Measuring the Security Harm of TLS Crypto Shortcuts

Drew Springall[†] Zakir Durumeric[†‡] J. Alex Halderman[†]
[†] University of Michigan [‡] International Computer Science Institute
{aaspring, zakir, jhalderm}@umich.edu

ABSTRACT

TLS has the potential to provide strong protection against network-based attackers and mass surveillance, but many implementations take security shortcuts in order to reduce the costs of cryptographic computations and network round trips. We report the results of a nine-week study that measures the use and security impact of these shortcuts for HTTPS sites among Alexa Top Million domains. We find widespread deployment of DHE and ECDHE private value reuse, TLS session resumption, and TLS session tickets. These practices greatly reduce the protection afforded by forward secrecy: connections to 38% of Top Million HTTPS sites are vulnerable to decryption if the server is compromised up to 24 hours later, and 10% up to 30 days later, regardless of the selected cipher suite. We also investigate the practice of TLS secrets and session state being shared across domains, finding that in some cases, the theft of a single secret value can compromise connections to tens of thousands of sites. These results suggest that site operators need to better understand the tradeoffs between optimizing TLS performance and providing strong security, particularly when faced with nation-state attackers with a history of aggressive, large-scale surveillance.

1. INTRODUCTION

TLS is designed with support for perfect forward secrecy (PFS) in order to provide resistance against *future* compromises of endpoints [15]. A TLS connection that uses a *non*-PFS cipher suite can be recorded and later decrypted if the attacker eventually gains access to the server's long-term private key. In contrast, a forward-secret cipher suite prevents this by conducting an ephemeral finite field Diffie-Hellman (DHE) or ephemeral elliptic curve Diffie-Hellman (ECDHE) key exchange. These key exchange methods use the server's long-term private key only for authentication, so obtaining it after the TLS session has ended will not help the attacker recover the session key. For this reason, the security community strongly recommends configuring TLS servers to use forward-secret ciphers [27, 50]. PFS deployment has increased substantially in the wake of the OpenSSL Heartbleed vulnerability — which potentially exposed the private keys for 24–55% of popular websites [19] — and of Edward Snowden's disclosures about mass surveillance of the Internet by intelligence agencies [36, 38].

Despite the recognized importance of forward secrecy, many TLS implementations that use it also take various cryptographic shortcuts that weaken its intended benefits in exchange for better performance. Ephemeral value reuse, session ID resumption [13], and session ticket resumption [52] are all commonly deployed performance enhancements that work by maintaining secret cryptographic state for periods longer than the lifetime of a connection. While these mechanisms reduce computational overhead for the server and latency for clients, they also create important caveats to the security of forward-secret ciphers.

TLS performance enhancements' reduction of forward secrecy guarantees has been pointed out before [33, 54], but their real-world security impact has never been systematically measured. To address this, we conducted a nine-week study of the Alexa Top Million domains. We report on the prevalence of each performance enhancement and attempt to characterize each domain's *vulnerability window* — the length of time surrounding a forward-secret connection during which an adversary can trivially decrypt the content if they obtain the server's secret cryptographic state. Alarmingly, we find that this window is over 24 hours for 38% of Top Million domains and over 30 days for 10%, including prominent Internet companies such as Yahoo, Netflix, and Yandex.

In addition to these protocol-level shortcuts, many providers employ SSL terminators for load balancing or other operational reasons [39]. SSL terminators perform cryptographic operations on behalf of a destination server, translating clients' HTTPS connections into unencrypted HTTP requests to an internal server. We find that many SSL terminators share cryptographic state between multiple domains. Sibling domains' ability to affect the security of each other's connections also adds caveats to forward secrecy. We observed widespread state sharing across thousands of groups

IMC 2016 November 14–16, 2016, Santa Monica, CA, USA

ACM ISBN 978-1-4503-4526-2/16/11.
DOI: http://dx.doi.org/10.1145/2987443.2987480

of domains, including tens of thousands of sites that use CloudFlare and thousands operated by Google.

The widespread use of TLS performance enhancements may make them an attractive target for nation-state adversaries. Our findings show that a relatively small attack against an SSL terminator (to recover cryptographic state) could be leveraged to trivially decrypt up to months worth of connections to many different web sites. The cryptographic state could conceivably also be obtained by legal compulsion, such as a warrant, subpoena, or national security letter.

To our knowledge, we are the first to quantify this attack surface and its dangers, and the first to show that real-world TLS security benefits far less from forward secrecy than statistics about support for PFS ciphers would suggest.

2. BACKGROUND

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that operate below the application layer and provide end-to-end encrypted channels for diverse applications, including HTTPS, IMAPS, and SMTP. This section explains how TLS provides forward secrecy and facilitates session resumption. We refer readers to RFC 5280 [12] for a detailed description of the protocol.

2.1 Forward Secrecy in TLS

In TLS, perfect forward security [15] protects the confidentiality of connections in the event that the server is *later* compromised by an attacker. Its threat model is an adversary who passively observes and records the TLS handshakes and encrypted traffic between a victim client and server. At some point after the connection has ended, the attacker gains access to the server's secret internal state — perhaps by exploiting a memory leak like Heartbleed [11], by seizing the hardware and performing live-memory forensics, or by computing the server's private key by factoring its public RSA modulus [25]. If the server correctly provides forward secrecy, the attacker will not be able to decrypt connections recorded in the past.

In order to achieve forward secrecy, TLS supports using Diffie-Hellman key exchange to negotiate temporary symmetric keys for the session. The protocol supports two main flavors of Diffie-Hellman: finite-field ephemeral Diffie-Hellman (DHE) and elliptic curve ephemeral Diffie-Hellman (ECDHE). In DHE handshakes, the server selects a finite cyclic group G and a generator g. It picks a random value a and sends g^a mod G to the client, while the client picks a random b and sends g^b mod G to the server. Both sides then compute g^{ab} and use it to derive the session keys. Per RFC 5246 [14], both the client and server should generate a fresh a and b for each handshake. ECDHE functions similarly but over an elliptic curve group. The client generates a random d_A and sends $d_A G$ to the client, while the client generates d_B and sends $d_B G$ to the server. Both then derive session keys from $d_A d_B G$.

Whether the handshake uses DHE or ECDHE, the server still needs to *authenticate* itself to the client in order to prevent man-in-the-middle attacks, and it does so using its long-term private key and certificate. However, a successful attack on the authentication would require compromising the private key before the TLS handshake completes. After that, as long as the client and server both discard the session state, the connection data should be infeasible to decrypt.

Using forward secret TLS handshakes is considered a security best practice [50], and all modern browsers support them. However, many server implementations, including Apache and Nginx, must be manually configured to use them.

2.2 Session Resumption

In order to reduce connection overhead, TLS allows subsequent sessions to resume a prior session without completing a full handshake. The protocol provides two mutually exclusive mechanisms to do this: session ID resumption and session tickets. Both mechanisms allow the server to skip a costly public-key operation on later connections, and they save one network round trip of latency. As we will show, server support for these resumption methods is pervasive — 50% of Mozilla Firefox TLS sessions are resumptions[1] — and of the Alexa Top Million websites that support HTTPS, 83% support session ID resumption and 76% support session tickets.

Session ID Resumption Session ID resumption was introduced in SSL 2.0 [26] and allows a client and server to quickly resume an existing session. During the initial handshake, the server provides a random session ID, which both the client and server maintain in a table that maps IDs to session keys and connection states from recent connections. Upon reconnection, the client provides this session ID in its first protocol message, Client Hello. If the server recognizes the session, it will respond with a Server Hello message containing the same session ID, after which both sides immediately resume an encrypted connection using the original session keys. RFC 5246 suggests a maximum 24-hour session lifetime, after which the server should discard the cached key and state.

Session Ticket Resumption TLS session tickets were introduced in RFC 4507 [51] and redefined in RFC 5077 [52]. They allow session resumption without requiring the server to maintain per-connection state. Instead, the server provides the client with an opaque encrypted "ticket" containing the session keys and other data necessary to resume the session. The client includes this ticket in later connections as an offer to resume without the full handshake. More precisely, when the client first connects, it includes an empty session ticket extension in its Client Hello. The server includes a corresponding extension in the Server Hello message and, after the key exchange completes, sends the client an opaque ticket and a lifetime "hint" in a New Session Ticket message. The client then stores a mapping of the server's identity to the session ticket and cryptographic state required for the client to resume the connection. On subsequent connections, the client includes the ticket in its Client Hello. If the server accepts the ticket, the pair completes an abbreviated handshake, like in session ID resumption. During this process, the server

---

[1] As seen by Mozilla Firefox Telemetry [43] from March 3 to March 15, 2016.

can reissue the client a fresh session ticket, but the cipher and session keys remain constant.

The ticket can contain arbitrary data, but RFC 5077 recommends a structure consisting of a randomly generated key name (identifying the symmetric keys used to encrypt the ticket), an IV, the encrypted state, and a MAC. The RFC recommends that the server encrypt the state using AES-CBC and a 128-bit key and construct the MAC using HMAC-SHA-256 with a 256-bit key. (Note that these keys are never revealed to the client, which merely stores the encrypted ticket and returns it in later connections.) Throughout this work, we refer to the symmetric encryption key as the "Session Ticket Encryption Key" (STEK). Common server implementations, including Nginx and Apache, support both loading pregenerated STEKs from the filesystem and generating random STEKs upon server initialization.

Impacts on Forward Secrecy Both of these performance enhancements degrade the protection achieved by forward-secret TLS handshakes [33, 54]. The client and server will store the same symmetric key for use in future sessions, extending the lifetime of the ephemeral handshake. More importantly, for session tickets, compromising the server's STEK would allow decryption of all prior connections for which that STEK was used. If a server's STEK never changes, the site provides no effective forward secrecy to connections that use TLS session tickets, regardless of the key exchange mechanism used.

2.3 Reusing Ephemeral Values

While not a session resumption technique, servers will oftentimes reuse DHE and ECDHE values to reduce computation for each initial handshake. For instance, with DHE, a server might repeatedly use the same value a so that it does not have to keep computing g^a. As we will discuss later, we empirically find that at least 7.2% of HTTPS domains in the Alexa Top Million reuse DHE values and 15.5% reuse ECDHE values.

Since the client will generate its own unique values (b, g^b), the session keys derived from g^{ab} will differ for every connection. However, an attacker who obtains the server's a can compute the session keys for any observed connection that uses it. Thus, forward secrecy is not actually achieved until the server stops reusing this value and securely erases it. If the server's a never changes, then a PFS key exchange does not provide any effective forward secrecy.

We discuss how session resumption and ephemeral value reuse affect the TLS ecosystem's attack surface — and attacker incentives — in Section 6.

2.4 Changes in TLS 1.3

Although still in the draft stage, TLS 1.3 [48] makes many changes to session resumption and other security properties. Session IDs and session tickets are nominally obsoleted, but the mechanisms persist via the pre-shared keys (PSKs).

A PSK identifier is issued by the server in a New Session Ticket message after the first handshake is complete and then included in the second connection's Client Hello. The identifier itself may contain a database lookup key (analogous to a session ID resumption) or an encrypted and authenticated copy of the TLS resumption state (analogous to a session ticket resumption). Unlike the current TLS versions, version 1.3 explicitly derives a separate resumption secret.

DHE	Alexa 1M domains (14Apr2016)	957,116
	Non-blacklisted domains	952,991
	Browser-trusted TLS domains	427,313
	Support DHE ciphers	252,340
	$\geq$ 2x same server KEX value	18,113
	All same server KEX value	12,461
ECDHE	Alexa 1M domains (15Apr2016)	958,470
	Non-blacklisted domains	954,338
	Browser-trusted TLS domains	438,383
	Support ECDHE ciphers	390,120
	$\geq$ 2x same server KEX value	60,370
	All same server KEX value	41,683
Session Tickets	Alexa 1M domains (17Apr2016)	956,094
	Non-blacklisted domains	951,978
	Browser-trusted TLS domains	435,150
	Issue session tickets	354,697
	$\geq$ 2x same STEK ID	353,124
	All same STEK ID	334,404

Table 1: **Support for Forward Secrecy and Resumption**

This resumption secret can be used in two ways for session resumption. The first is for a direct resumption for a secondary session via the "`psk_ke`" mechanism. The second is to be used as authentication for resumed connection that conducts a second (EC)DHE key exchange via the "`psk_dhe_ke`" mechanism.

In addition to these two, the resumption secret can also be used for QUIC-like [47] 0-RTT communication. In this case, "early data" is sent by the client while awaiting the completion of a resumed or new TLS handshake. The data is encrypted to the resumption secret and can stream until the client receives the server's Finished message.

3. DATA COLLECTION

To assess the impacts of session resumption and ephemeral value reuse, we measured HTTPS behavior of Alexa Top Million domains [2] over a 9-week period in the Spring of 2016. We repeatedly connected to each server on TCP/443 using a version of the ZMap tool chain [16, 20] that we modified to support session ID and ticket resumption. In all cases, we restricted our analysis to websites that presented browser-trusted certificates that chain to the NSS root store[2]. Table 1 gives high level metrics from conducting 10 TLS connections in quick succession to each Alexa Top Million domain on the days given.

As with any active scanning research, there are many ethical considerations at play. We followed the best practices defined by Durumeric et al. [20] and refer to their work for more detailed discussion of the ethics of active scanning. All scans were completed from the University of Michigan

---

[2] Durumeric et al. find that 99.5% of certificates trusted by NSS are valid in all major browsers and can be used to estimate browser trusted websites [18].

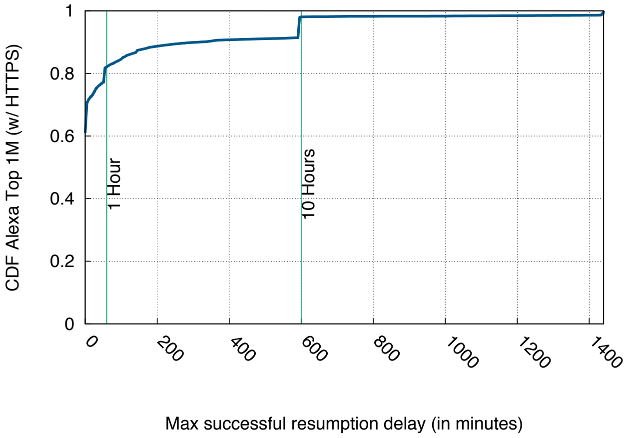

Figure 1: **Session ID Lifetime**—We measured how long Session IDs were honored by HTTPS websites in the Alexa Top Million.

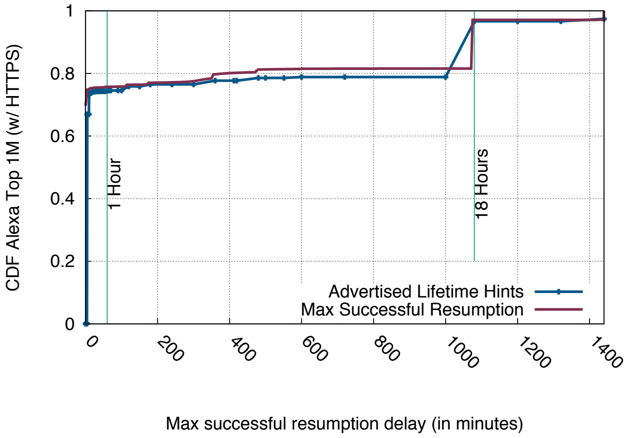

Figure 2: **Session Ticket Lifetime**—We measured advertised session ticket lifetime and how long tickets were honored by Alexa Top Million websites.

campus and followed the institutional blacklist. For experiments that required multiple connections in a single day, we restricted our measurements to popular sites in the Alexa Top Million for which this load should be negligible. When possible we used existing data from the Censys Project [16] instead of running redundant scans. We are publishing all of the data we independently collected on Scans.io [17], and our modifications have been merged into the main ZMap project.

Alexa Top Million Dataset Our measurements occurred within a 9-week period from March 2, 2016 to May 4, 2016 and used the Alexa Top Million as the target domains. We saw a surprising amount of churn within the Top Million domains from day to day. In total, we scanned 1,527,644 unique domains including over 155K which were in ≤ 7 polls of the Top Million. Only 539,546 domains remained in the Top Million for the whole 9 weeks. Of these, 369,034 (68%) ever supported HTTPS, 291,643 (54%) ever presented a browser-trusted certificate, and 288,252 (53%) ever issued a session ticket, completed a DHE or ECDHE key exchange, or resumed a session. To prevent churn in the Top Million from biasing our results, we restrict measurements over multiple days to domains that remained in the list for the entire period.

4. TLS SECRET STATE LONGEVITY

In this section, we describe HTTPS domains' behavior in practice with regard to the lifetime of cryptographic state, including how long session ID and session ticket resumption is allowed, the lifetime of session ticket encryption keys, and the reuse of key exchange values. We find that while session IDs and session tickets are generally only honored for under an hour (82% and 76%, respectively), session ticket encryption keys (STEKs) persist much longer.

4.1 Session ID Lifetime

To measure how long session IDs are accepted, we initiated a TLS handshake with each of the Alexa Top Million domains on April 27, 2016. We attempted to resume each session one second later and then every five minutes until either the site failed to resume the session or 24 hours had elapsed. Of the 433,220 domains that supported HTTPS and presented a browser trusted certificate, 419,302 (97%) indicated support for session ID resumption by setting a session ID value in the Server Hello message, and 357,536 (83%) resumed the session after a one second delay.

As shown in Figure 1, the distribution of lifetimes is somewhat discrete: 82% of domains that supported session ID resumption allowed resumption for one hour or less, and 61% did for less than five minutes. Only 2,845 domains (0.8%) resumed sessions for 24 hours or longer; 86% of those domains belong to or are hosted by Google. We also note that Facebook's CDN honored session IDs for more than 24 hours.

These empirical results align with the default configuration of population web server implementations. Apache enables session ID resumption by default and sets the lifetime to five minutes [41]. Nginx issues session IDs but does not allow resumption unless it is explicitly configured; session IDs expire after five minutes when enabled unless the administrator sets a different lifetime [42]. Microsoft IIS expires session IDs after ten hours [40], corresponding to the jump seen in Figure 1.

4.2 Session Ticket Lifetime

We use a similar experiment to measure how long domains allowed session tickets to resume TLS connections and the hinted lifetime. We initiated a TLS handshake with each site in the Alexa Top Million on April 29, 2016. We attempted to resume each connection one second later, then every five minutes until either the domain failed to resume the session or 24 hours had elapsed. If the domain reissued a session ticket during any of the connections, we continued to attempt resumption with the ticket issued from the first connection. We found that 366,178 out of the 461,475 domains with a browser-trusted certificate (79%) issued a session ticket and 351,603 (76%) resumed the session after one second.

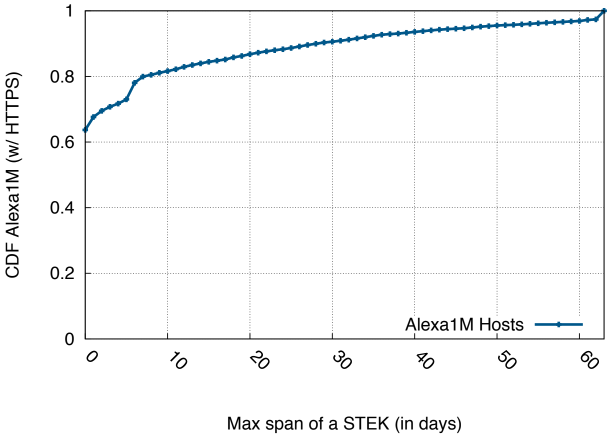

Figure 3: **STEK Lifetime**—TLS connections cannot achieve forward secrecy until the STEK (the key used by the server to encrypt the session ticket) is discarded.

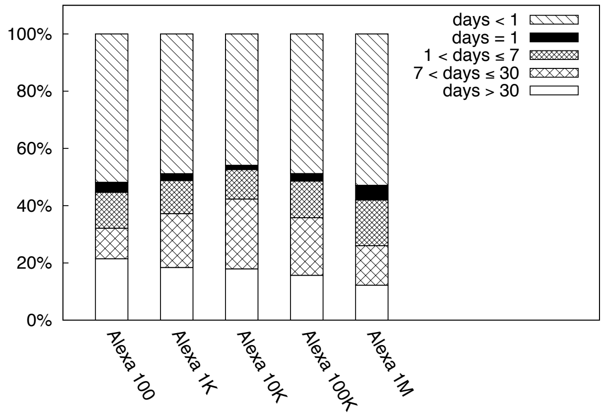

Figure 4: **STEK Lifetime by Alexa Rank**—We found 12 Alexa Top 100 sites that persisted STEKs for at least 30 days.

Similar to session ID resumption, 67% of domains accepted a session ticket for less than five minutes and 76% for one hour or less as seen in Figure 2. The indicated ticket lifetime closely follows the advertised lifetime hint, with the exception of 14,663 domains that leave it unspecified and up to the client's policy [52]. At the extreme end, we found that two domains specified a lifetime hint longer than ten days: `fantabobworld.com` and `fantabobshow.com`, both of which specified a 90 day hint. 54,522 unique domains hosted by CloudFlare resumed for 18 hours, causing the steep increase in Figure 2. As with session ID resumption, 8,969 domains accepted tickets for 24 hours, of which 8,535 were hosted by Google (95%), which specified a 28 hour lifetime hint.

This behavior also agrees with the known defaults for popular web server implementations. Apache and Nginx both enable session ticket resumption by default with a three minute lifetime.

4.3 STEK Lifetime

While the time span that domains will accept previously issued session tickets is an important metric, it reflects only the ticket's lifetime (set by policy) and not necessarily the time period for which the associated STEK exists and is used to issue new session tickets. As discussed in Section 2, the content of a historical session can be decrypted using a site's STEK regardless of whether a PFS handshake occurs and regardless of whether the ticket's lifetime has expired or not. In other words, a "forward secret" session is not actually forward secret while the STEK that encrypted the associated ticket persists.

While it is not possible to directly detect that the key used to encrypt the session state has changed, popular server implementations include a 16-byte STEK identifier in the ticket, as prescribed in RFC 5077 [52]. We reviewed popular open-source TLS implementations, including OpenSSL, LibreSSL, GNUTLS, mbedTLS, and NSS, and found that all follow this recommendation except for mbedTLS, which uses a 4-byte STEK identifier. We also tested Microsoft's SChannel implementation and found it to use an ASN.1 encoded object containing a DPAPI object [7]. For the measurements below, we parsed this object and extracted the Master Key GUID to use as the STEK identifier.

Between March 2, 2016 and May 4, 2016, we connected to the Alexa Top Million domains daily and recorded the session ticket that was issued by the server, if one was issued. We were able to determine the lifetime of each STEK by looking for the first and last time that the (STEK identifier, domain) pair was seen. As opposed to measuring the number of sequential days that a domain issues tickets with identical STEK identifiers, this metric accounts for much of the real-world jitter seen in Internet scanning. This could be due to the ZMap tool-chain's choice of A-record entries between days, a poorly configured load balancer which does not maintain client-server affinity, or simply the server failing to respond to one our connections. It is highly unlikely that an administrator would switch static STEKs only to switch back or that a randomly generated STEK identifier would collide within the bounds of our study. Therefore, we can safely assume that a STEK was in use between the first and last time that its identifier was seen and that any intermediate STEK identifiers seen were the result of fluctuations connecting to different servers.

Of the 291,643 browser-trusted sites always in the Alexa Top Million, 66,941 (23%) never issued a session ticket. 118,835 (41%) used different issuing STEKs for each day. 63,976 domains (22%) reused the same STEK for at least 7 days, and 28,210 domains (10%) reused for at least 30 days. We show the CDF of these lifetimes in Figure 3.

We found a surprising collection of websites, including those of major Internet companies, that fall within the 30+ day reuse. Table 2 shows the ten most popular domains according to their average Alexa ranking that reused a STEK for at least 7 days. While there are many other notable domains, we note that there are a total of eight `yandex.[tld]` domains, each of which showed 63 days of reuse, `slack.com` (a popular team communication service) showed 18 days of reuse, and

Rank	Domain	# Days	Rank	Domain	# Days
5	yahoo.com	63	31	netflix.com	54
19	qq.com	56	35	imgur.com	63
20	taobao.com	63	41	tmall.com	63
21	pinterest.com	63	53	fc2.com	18
28	yandex.ru	63	55	pornhub.com	29

Table 2: **Top Domains with Prolonged STEK Reuse**—We show the most popular domains (by average Alexa rank) that reused a STEK for at least 7 days.

Rank	Domain	# Days	Rank	Domain	# Days
31	netflix.com	59	580	kayak.com	13
53	fc2.com	18	592	cbssports.com	60
392	ebay.in	7	626	gamefaqs.com	12
456	ebay.it	8	633	overstock.com	17
528	bleacherreport.com	24	730	cookpad.com	63

Table 3: **Top Domains with Prolonged DHE Reuse**—We show the most popular domains (by average Alexa rank) that reused a DHE value for at least 7 days.

Rank	Domain	# Days	Rank	Domain	# Days
31	netflix.com	59	353	paytm.com	27
74	whatsapp.com	62	464	playstation.com	11
158	vice.com	26	527	woot.com	62
221	9gag.com	31	528	bleacherreport.com	24
322	liputan6.com	28	615	leagueoflegends.com	27

Table 4: **Top Domains with Prolonged ECDHE Reuse**—We show the most popular domains (by average Alexa rank) that reused an ECDHE value for at least 7 days.

`mail.ru` showed 63 days of reuse. 63 days indicates that it was seen on both the first and last day of our study and was likely in use both before and after our study.

Figure 4 depicts how STEK lifetimes varied with Alexa rank tiers according to the average rank of each domain over the 9-week period. We observed 56 domains which issued session tickets in the Alexa Top 100, 494 in the Top 1K, 4,154 in the Top 10K, 37,224 in the Top 100K, and 224,702 in the Alexa Top Million. Again, these are only domains which remained within the Alexa Top Million for the entire span of our study.

The longevity of STEK lifetimes can be largely explained by the the popular implementations. Apache 2.4.0 and Nginx 1.5.7 and later allow an administrator to configure the server to read 48 bytes of randomness from a file path on disk. This file contains the STEK identifier, encryption key, and MAC key in order to synchronize STEKs across servers. This configuration can only be changed via direct interaction from the administrator and restarting the server process. If this option is not available, or if a key file is not configured, the server randomly generates a STEK on startup and uses it for the lifetime of the process.

While there is a worrying set of websites that appeared to never rotate STEKs, we note that many have more reasonable configurations. Google, Twitter, YouTube, Baidu, and many others never reused an issuing STEK across days. However, as we will discuss in Section 7, that is not always the sole indicator of a secure configuration.

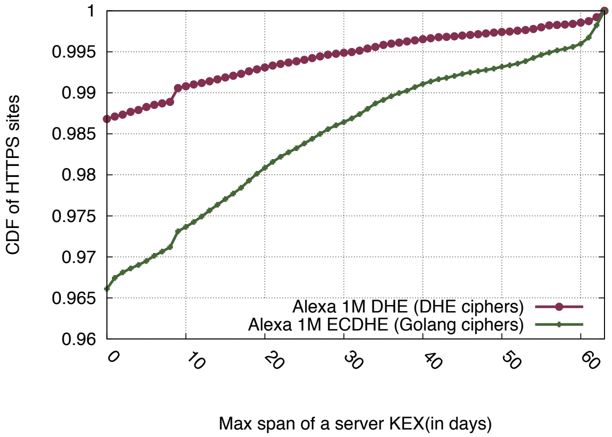

Figure 5: **Ephemeral Exchange Value Reuse**—We measured how long Alexa Top Million websites served identical DHE and ECDHE values (note vertical scale is cropped).

4.4 EC(DHE) Value Lifetime

As described in Section 2, TLS servers can cache and reuse ephemeral handshake values (a, g^a in a finite-field Diffie-Hellman exchange or d_A, $d_A G$ in elliptic curve Diffie-Hellman) to reduce the computational cost of public key cryptography. Table 1 shows that 7.2% of domains in a single Alexa Top Million list reuse a DHE value for some amount of time and 15.5% reuse an ECDHE value for amount of time.

To determine how long these ephemeral values persist, we analyzed two sets of daily scans for the Top Million Domains. One set, obtained from the Censys project [16], offered only DHE ciphers and the other offered ECDHE and RSA ciphers, with ECDHE as the first priority.

DHE Of the 291,643 domains consistently in the Alexa Top Million and who support HTTPS with a valid certificate, only 166,608 (57%) ever connected successfully when the client offered only DHE ciphers. 12,824 domains (4.4%) reused a DHE value for some amount of time in the 10 connection scans referenced in Table 1. The Censys project scans show that 3,849 (1.3%) reused a DHE value for at least one day, 3,347 (1.2%) for at least 7 days, and 1,527 (0.52%) for 30 or more days. Figure 5 shows this visually.

Table 3 shows the top ten domains which reused a DHE value for more than 7 days as determined by their average Alexa rank. We also find `commsec.com.au` (an Australian brokerage firm) with 36 days of reuse and 32 `kayak.[tld]` domains with between 6 and 18 days of reuse.

ECDHE 234,302 domains 80% of those consistently in the Alexa Top Million who support HTTPS with a valid certificate, completed an ECDHE handshake. 42,029 domains (14.4%) reused an ECDHE value for some amount of time in our 10 connection scans referenced in Table 1. In our daily scans, we saw 9,886 domains (3.4%) that reused an ECDHE value for at least one day, 8,710 (3.0%) reused for at least 7 days, and 4,071 (1.4%) reused for 30 or more days. This is shown visually in Figure 5.

Table 4 shows the top ten domains that reused an ECDHE value for more than 7 days. Notable domains beyond the top ten include `betterment.com` (an online investing service) with 62 days of reuse, `mint.com` (a budgeting website that connects to banks and investment services) with 62 days of reuse, and `symantec.com`, `symanteccloud.com`, and `norton.com` with 41, 16, and 19 days of reuse respectively.

As seen in Figure 5, the ephemeral value longevity metrics are fairly consistent with one another, but are substantially different from the STEK longevity rates seen in Figure 3.

5. TLS SECRET STATE SHARING

When measuring the increased attack surface resulting from stored TLS secrets, it is also important to consider cases where secrets are shared across domains, servers, or data centers. If a shared TLS secret is extracted from a single site, it can be used to compromise connections to all the other sites regardless of whether they use different long-term SSL certificates.

We found many "service groups" in which multiple domains shared a session cache, STEK, or Diffie-Hellman value, making these secrets particularly valuable targets for attack. While it would be logical for a single domain to use this technique to allow sessions to be resumed across multiple servers, the magnitude of sharing across domains was surprising. The root cause of this behavior is likely that domains share an SSL terminator, whether it is a separate device such as a Cavium card [8] or multiple domains running on the same web server.

5.1 Shared Session ID Caches

To establish a lower bound on how many websites share session ID caches, we conducted a cross-domain probing experiment where we attempted to resume a TLS connection to domain b with a session that originated from domain a. If performed exhaustively, this would require hundreds of thousands of connections to each domain. However, we made the experiment tractable by limiting groups to a small number of domains from each AS and by transitively growing the graph. That is, if we observed that id_a was valid on domain b and id_b was valid on domain c, we conclude that id_a would have also been valid on domain c and group domains a, b, and c together.

For each site, we randomly selected up to five other sites in its AS and up to five sites that shared its IP address and tested whether its session ID allowed connection to these other sites. We note that because servers can expire session IDs at any time, there is no harm to the server to provide an invalid session ID; the server will simply complete a typical TLS handshake as if no session ID had been presented.

Of the 357,536 domains that supported session ID resumption in Section 4.1, we found 212,491 service groups, of which 183,261 (86%) contained only a single domain. The largest service group we found belonged to CloudFlare and contained 30,163 domains (66% of the 45,520 Alexa Top Million domains in their AS). We show the ten largest session cache service groups in Table 5.

Operator	# domains	Operator	# domains
CloudFlare #1	30,163	Blogspot #2	743
CloudFlare #2	15,241	Blogspot #3	732
Automattic #1	2,247	Blogspot #4	648
Automattic #2	1,552	Shopify	593
Blogspot #1	849	Blogspot #5	561

Table 5: **Largest Session Cache Service Groups**

Operator	# domains	Operator	# domains
CloudFlare	62,176	GoDaddy	1,875
Google	8,973	Amazon	1,495
Automattic	4,182	Tumblr #1	975
TMall	3,305	Tumblr #2	959
Shopify	3,247	Tumblr #3	956

Table 6: **Largest STEK Service Groups**

Operator	# domains	Operator	# domains
SquareSpace	1,627	Atypon	167
LiveJournal	1,330	Affinity Internet	146
Jimdo #1	179	Line Corp.	114
Jimdo #2	178	Digital Insight	98
Distil Networks	174	EdgeCast CDN	75

Table 7: **Largest Diffie-Hellman Service Groups**

As shown in the table, we observed cases where a single logical provider (such as a CDN or cloud services company) had multiple service groups even within the same /24 CIDR block. We manually confirmed that this was not an artifact of our grouping methodology and in fact reflected the remote configuration. While we believe that this measurement technique is effective, it provides only a lower bound on the true number of domains that share session caches. Our ability to provide a tighter estimate is limited, since TLS does not provide the client any information about the session cache or saved session state other than the random session ID.

5.2 Shared STEKs

To track how STEKs are shared across servers, we connected to each April 17, 2016 Alexa Top Million domain ten times over a six hour window and grouped sites together that shared at least one STEK identifier during the scans. Since some providers rotate session tickets at smaller intervals than six hours, we repeated the experiment with one connection over a 30 minute window, similarly grouped domains, and then joined the two groups.

Of the 354,697 sites that supported session tickets, we found 170,634 STEK service groups, of which 140,715 (83%) contained only a single domain. As with session IDs, the largest group belonged to CloudFlare; it contained 62,176 domains. The next largest belonged to Alphabet (Google's parent company) and contained 8,973 hosts sharing a STEK. We show the top ten largest STEK service groups in Table 6.

5.3 Shared (EC)DHE Values

Lastly, we looked for Alexa Top Million domains that shared DHE or ECDHE key-exchange values. To do this, we completed 10 TLS handshakes with each Alexa Top Million domain over a five-hour window. As with the shared STEK experiment, we also performed a scan that made a single connection to every domain during a 30 minute window. Both scans were conducted twice, once with only DHE ciphers and once with only ECDHE ciphers, for a total of four scans.

We found that Diffie-Hellman values were shared in fewer instances and by somewhat smaller groups than either session caches or STEKs. The most widely shared DHE value was one we saw 1,368 times across 137 domains and 119 IP addresses, all within AS 20401 (Hostway Corporation). We also found a single ECDHE value shared 1,790 times across 179 domains on a single IP, which appeared to be a Jimdo hosting server [31] on Amazon EC2.

We labeled servers that ever presented the same DHE or ECDHE key-exchange value to be part of the same service group. We found 421,492 Diffie-Hellman service groups, of which 417,397 (99%) contained only a single domain. The largest group belonged to SquareSpace and contained 1,627 domains. We identify the largest ephemeral value service groups in Table 7.

6. CRYPTO SHORTCUT DANGERS

As of May 2016, we find that 90.2% of Top Million domains with trusted HTTPS use forward secret key exchanges for connections from modern browsers. Prior to our study, we — the authors — would have assumed from this that connections would be forward secret shortly after the connection has ended. However, when we consider the interaction of crypto shortcuts and cross-domain secret sharing, we see that this is not the case and that many popular domains remain susceptible to retrospective decryption.

As opposed to the naive understanding, forward secrecy is not a binary concept being either forward secret or not forward secret. Forward secrecy is a gradient where the confidentiality of the data is forward secret after some passage of time dependent on many different factors. At one extreme, an arbitrarily complex key-ratcheting mechanism could protect data confidentiality even if an endpoint is compromised while the connection is in progress. The attacker would be able to decrypt the connection's content after the compromise, but not before. At the other extreme, a TLS connection that uses RSA key exchange is effectively never forward secure. Due to the long-term nature of most SSL certificates as well as the likelihood that they are stored on disk, recovery is likely possible even long after the certificate has expired.

To quantify the amount of forward secrecy, we can attempt to establish lower bounds for each site's *vulnerability window*. This is the span of time during which an attacker could recover the session keys for an observed TLS connection by compromising secret values stored by the server. Our measurements from the previous sections allow us to estimate lower bounds for this window, but the true exposure may be much greater. While we can detect that a server refuses to resume older sessions, we cannot tell whether it has securely erased the corresponding secrets or whether the secrets may be vulnerable to forensic recovery.

In addition to quantifying the amount of forward secrecy, we also wish to account for the concentration of the secrets themselves. In a secure world, a compromise of one server would affect as few connections on as few domains as possible. But as shown in Section 5, this is far from the case and that the compromise of a small number of SSL endpoints could endanger an out-sized number of domains' content.

The interaction of these two factors presents an enticing target for an attacker who wishes to decrypt large numbers of connections for a comparatively small amount of work.

6.1 Exposure from Session Tickets

The long-term usage of session ticket encryption keys (STEKs) is the most worrisome practice we observed. Since the session ticket contains the session keys encrypted with the STEK, and since it is sent as part of each TLS connection outside of the TLS tunnel (initially by the server and subsequently by the client), an attacker who obtains the associated STEK can decrypt the ticket, recover the session keys, and decrypt the connection contents.

The vulnerability window begins when the STEK is generated (potentially before the victim connection) and ends when it is securely erased from all servers. As reported in Section 4.3, 36% of the ticket-issuing domains we considered reused the same STEK for at least a day, 22% for more than a week, and 10% for more than a month.

In Figure 6, we visualize the interaction of session ticket service groups and the median STEK reuse for each service group. The two largest service groups (CloudFlare and Google) account for 20% of Top Million HTTPS sites and are shown in the far-left column, and both reused STEKs for less than 24 hours. On the opposite end of the longevity spectrum were TMall (a Chinese online retailer) and Fastly (a CDN), which are represented by the largest red elements in the second column of Figure 6. Together, they accounted for 1,208 domains. Fastly, which controlled domains such as `foursquare.com`, `www.gov.uk`, and `aclu.org`, always issued session tickets with the same STEK throughout our 9-week study.

While not one of the largest service groups, we note a concerning cluster of sites controlled by Jack Henry & Associates. This service group contains 79 bank and credit union domains which issued session tickets for 59 days using a single STEK and then all rotated to a different — but still shared — STEK for the final 4 days of our study.

While we are pleased that many of the largest service groups rotate their STEKs at least daily, the magnitude of reliance on a small number of secret values is disconcerting. Current versions of Chrome, Firefox, IE, and Microsoft Edge all offer the session ticket extension by default and an attacker who could collect the traffic as well as obtain the STEK within

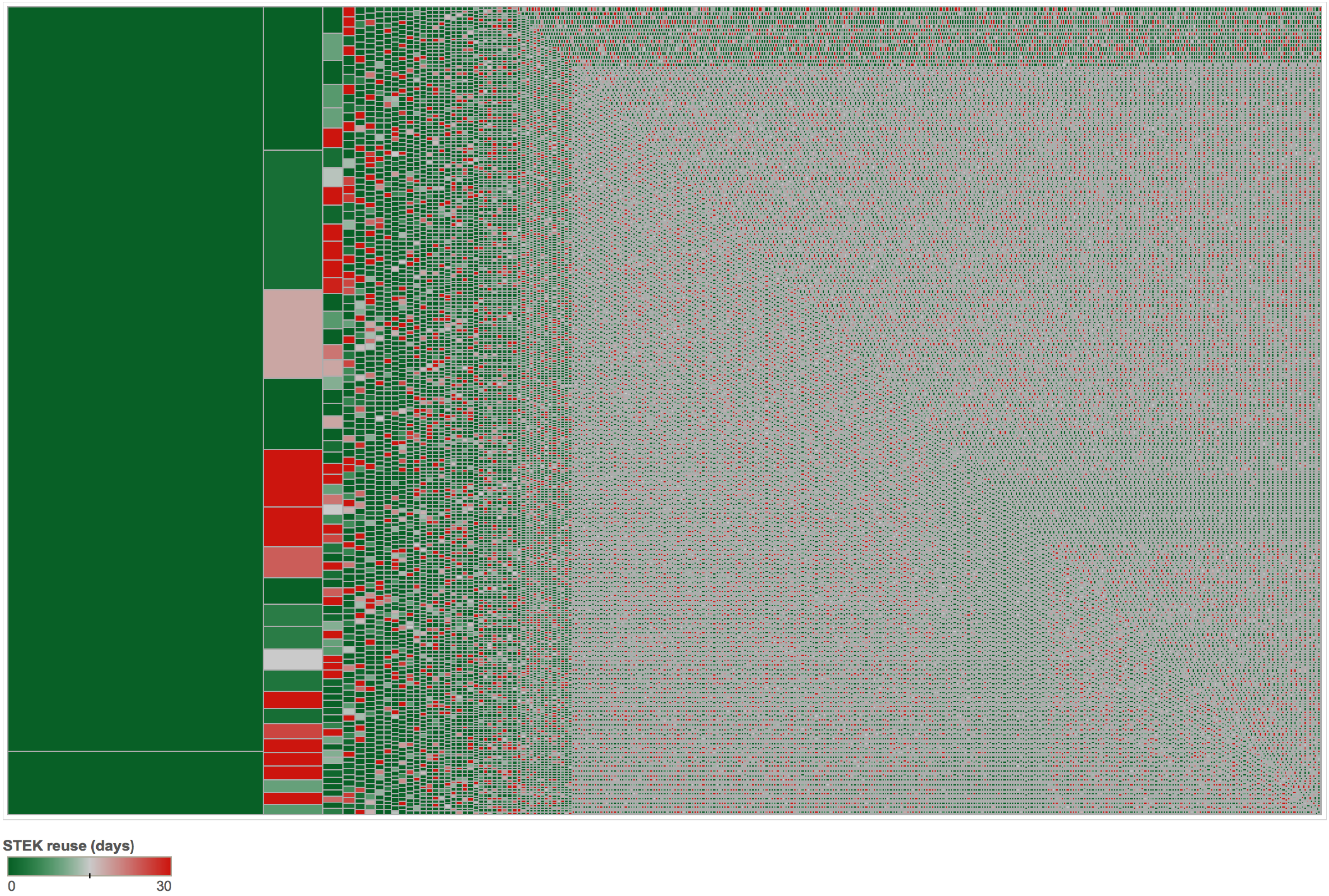

Figure 6: **STEK Sharing and Longevity Visualization**—Each box in this illustration is sized proportionally to the number of domains in that service group and colored according to the observed longevity of the key. Solid red boxes represent groups of domains that shared a key that persisted for at least 30 days.

the vulnerability window would be able to decrypt and access the millions of victims' connection content with ease.

6.2 Exposure from Session Caches

When a server supports session ID resumption, an attacker can potentially recover keys for past sessions as long as they reside in the server's session cache. As such, the vulnerability window begins when the victim connection completes its handshake and ends when the server implementation securely discards the session state.

Our experiments in Section 4.1 show that at least 83% of Top Million sites employ session caching and retain state for some amount of time after a connection, and at least 18% do so for more than 60 minutes. Section 5.1 shows that session cache sharing is widespread, with 49% of Top Million domains sharing a cache with at least one other popular domain. Figure 7 shows the interaction of these measurements.

The combined effect of session caching and cache sharing makes large interdomain session caches a particularly attractive target for attackers. The ten largest shared caches (Table 5) account for 15% of Top Million domains and exhibited median vulnerability windows of 5 and 1,440 minutes (24 hours). Of these, the five longest-lived all belonged to Google Blogspot and exhibited median cache lifetimes ranging from 4.5 hours to 24 hours (the maximum we tested). An attacker who could access the contents of any one of these caches would be able to decrypt hours' worth of TLS traffic for hundreds of popular sites.

Compared with Figure 6, Figure 7 shows a similar distribution within the largest service groups. Although the maximum vulnerability windows are orders of magnitude different, the proportional distribution is similar.

6.3 Exposure from Diffie-Hellman Reuse

When a server reuses Diffie-Hellman ephemeral values (contrary to the advice of RFC 5246 [14]), this also leads to an extended vulnerability window. The window last from the time the server generates its random Diffie-Hellman value (a or d_A) until that value is securely erased. Like session tickets, an attacker who leaks the server's Diffie-Hellman value can also decrypt *future* TLS connections until the server ceases using that value as well as any previous connections using that value.

Figure 7 shows combined effect of longevity and interdomain sharing was significantly smaller for Diffie-Hellman reuse than for session resumption, but it still resulted in a few

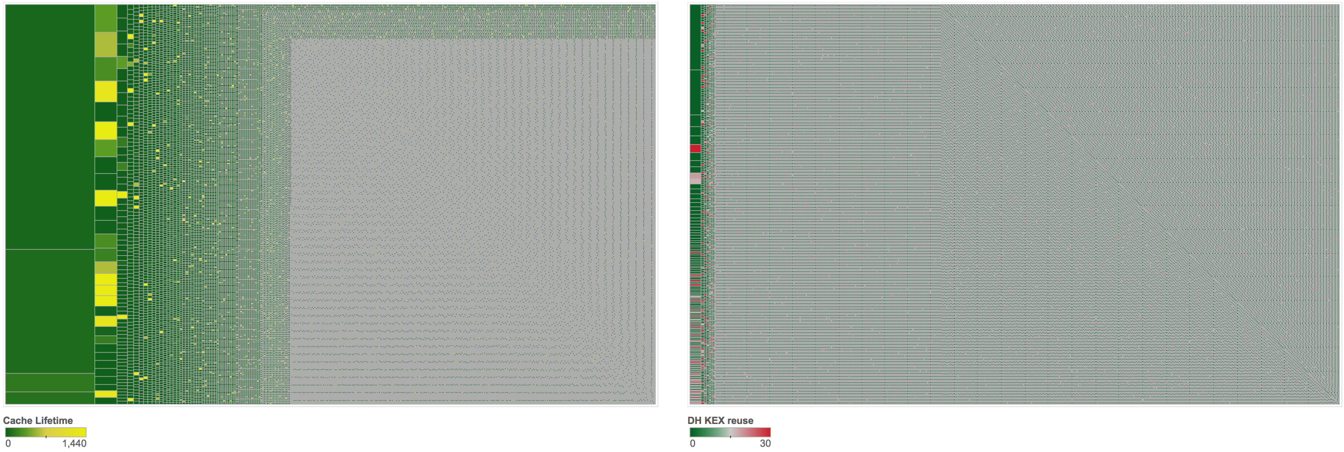

Figure 7: **Visualizing Session Caches and Diffie-Hellman Reuse**—For comparison with Fig. 6, we show similar illustrations of the longevity and cross-domain sharing exhibited by session caches (*left*) and repeated Diffie-Hellman values (*right*).

notable high-value targets. Affinity Internet shared a single Diffie-Hellman value across 91 domains for 62 days, and Jimdo shared one value for 19 days across 64 domains and another value for 17 days across a different 60 domains (seen as the red blocks in the far left column).

6.4 Combined Exposure

Since session tickets, session caches, and Diffie-Hellman reuse all lead to an extended vulnerability window, an attacker with some way of accessing the server's internal state could choose to exploit any of them to compromise forward secrecy. A domain's overall exposure is determined by the longest vulnerability window it exhibits for any of these mechanisms.

Of the 291,643 domains that were in the Alexa Top Million for the duration of our measurements and supported HTTPS with a browser-trusted certificate, 288,252 (99%) issued a session ticket, resumed a session, or conducted a DHE or ECDHE key exchange. Figure 8 shows the distribution of the maximum vulnerability window found for every domain.

About 90% of browser-trusted Top Million domains with browser-trusted certificates are configured to support forward-secrecy with modern browsers, which, as commonly thought of, would result in a vulnerability window that lasts no longer than the connection. Due to combined effects of the TLS crypto shortcuts we have discussed, we find that 110,788 domains (38%) have a maximum vulnerability window of more than 24 hours, 65,028 (22%) of more than 7 days, and 28,880 (10%) of more than 30 days.

7. NATION-STATE PERSPECTIVE

As seen above, our results indicate that TLS crypto shortcuts leave popular HTTPS sites significantly less well protected than we thought in the face of server-side information leaks such as Heartbleed. However, the risks of these mechanisms appear even more severe if we consider threats from nation-state attackers such as the NSA. In particular, the "shape" of the vulnerability windows created by session tickets is ideally suited for exploitation by intelligence agencies for surveillance purposes. In this section, we consider how a nation-state attacker might seek to exploit TLS crypto shortcuts and we assess the potential impact on Internet security of such a compromise against one particular high-value target, Google. Due to the availability of information regarding the NSA and other "Five Eyes" agencies, we focus on the *modi operandi* of these groups.

Recent TLS vulnerabilities—such as FREAK [6], Logjam [1], and DROWN [4]—require active interference with each connection, making them unsuitable for stealthy, retroactive, or wide-scale surveillance. Some researchers believe that NSA can currently defeat TLS encryption when used with 1024-bit RSA [35] or DHE [1]. In either case, specific non-standard configurations would be required in cipher selection (preferring RSA client write and DHE ciphers with specific DH constants respectively) to enable passive decryption. However, there is no credible evidence that they can break the higher-strength cryptography now used by most popular sites.

7.1 The STEK as an Enabling Vector

It is well known that the NSA and other intelligence agencies have the ability to passively collect vast amounts of Internet traffic. Some collection is "targeted" at a specific person, website, or IP address, but other collection involves indiscriminately storing all network traffic in large circular buffers, such as XKEYSCORE [36] and TEMPORA [38], for *ex post facto* analysis [55].

These capabilities are almost certainly challenged by the growth of TLS, which has accelerated following increased public awareness of surveillance [37] and the availability of free browser-trusted certificates [30]. Faced with these constraints, nation-state adversaries might find that session tickets provide an appealing mode of attack. Exfiltrating one 16-byte STEK from a server would allow the adversary to decrypt every passively collected connection which uses the TLS session ticket extension during the vulnerability window, including connections within the window but before the STEK was leaked. As seen in Figures 6, stealing a small

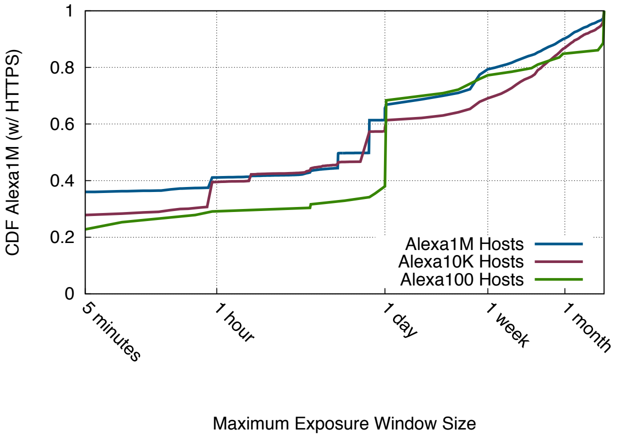

Figure 8: **Overall Vulnerability Windows**—This CDF depicts the combined effects of exposure from session tickets, session caches, and Diffie-Hellman reuse.

number of STEKs would enable decryption of content from a large number of domains.

Although obtaining a STEK may require attacking the provider and not the end-target, intelligence agencies have been known to conduct sophisticated intrusions in order to facilitate later passive surveillance. GCHQ infiltrated SIM card manufacturer Gemalto to steal the encryption keys used by millions of cellphones [10, 45, 53]. They also attacked engineers at Belgacom, the largest ISP in Belgium, in order to gain access to traffic from its core routers [21]. An unknown adversary—thought to be a nation state—infiltrated Juniper Networks' code repository and inserted a cryptographic backdoor into the company's VPN products [9]. Similar operations could be used to access STEKs from high-value targets.

It is likely that some domains synchronize STEKs across servers in many network locations and jurisdictions. A nation-state attacker could attempt to compromise the synchronization mechanism, or they could convince a hosting facility to grant them access to the equipment for physical attacks [3]. Within its national borders, such an attacker might use the court system to compel an organization to turn over the STEKs, as Lavabit was ordered to do with its TLS private key [46]. However obtained, the STEK would provide global decryption capabilities.

7.2 Target Analysis: Google

To provide a concrete example, we simulate a nation-state attacker's possible analysis of an attack against Google—a large tech company with experience being attacked by [23,56] and defending against [22] nation-state adversaries. As the attacker, our goal is to leverage our existing passive collection systems—which currently only see TLS ciphertext—in order to gain insight into a large swath of network communication.

As seen in Table 6, a single STEK is shared by nearly all Google web services, including Search, GMail, Drive, Docs, Hangouts, and many more. We find that Google also uses the same STEK for other TLS-based protocols, including SMTP+STARTTLS, SMTPS, IMAPS, and POP3S. We experimentally determined that Google's STEK is rolled over every 14 hours, but issued tickets are accepted for up to 28 hours, indicating that each key is maintained at least that long. This implies that only two 16-byte keys must be stolen every 28 hours in order for the attacker to be able to decrypt all Google TLS connections that use the session ticket extension.

By requesting the MX records for the Alexa Top Million domains, we find that over 90,000 domains (9.1%) point to Google's SMTP servers. This is likely a reflection of the Google for Work program in which more than 2 million businesses (including 60% of Fortune 500 companies) use Google's service for their internal and external e-mail [24]. So in addition to the e-mail communications and web-app data from `@gmail.com` addresses, the content of any company which relies on Google's cloud service for intracompany e-mail or web-apps would be decryptable.

The intelligence value from the resulting decryption ability would extend far beyond Google's own properties. Google supplies analytics, ads, and APIs to many websites whose requests would likely send the user's Google cookies. We have confirmed that browser connections to these Google dependencies use the same STEK as other Google sites. Obtaining the Google STEK would allow tracking users even when they are not directly accessing Google sites.

As this analysis shows, Google's STEK would be an immensely valuable target, as it would enable the decryption of a huge amount of encrypted traffic and provide intelligence on targeted and untargeted individuals. Even if the exploitation required the use of sophisticated, persistent hardware or software implants, the trade off between the possibility of their discovery and the rich intelligence that would be gained likely falls within the acceptable risks category for many nation-state adversaries.

Google's is the case of a well protected organization with a highly talented security team. As shown in Section 4.3, many other organizations—including large tech and cloud service companies—appear to be far less cognizant of the risks of TLS performance enhancements. As an example, Yandex is a Russian Internet company that mirrors Google's offerings in search, e-mail, and cloud storage and enjoys a 57% domestic market share [32]. Like Google, Yandex appears to use a single STEK for almost all of its properties, but unlike Google, this STEK has been in use continuously since at least January 10, 2016—eight months prior to this writing. A single operation to recover this STEK would immediately allow decryption of months' worth of connections.

8. DISCUSSION

While we've notified the domains and companies that we explicitly point out above, there are other ways to address the ecosystem-wide issues we found. In this section we step back and view the problems found with (EC)DHE values, session caches, and session tickets from a community level. We draw lessons from our measurements and make recommendations for server operators.

8.1 Security Community Lessons

The security community's advocacy for the adoption of TLS forward secrecy has shown clear gains, with over 90% of Top Million HTTPS sites now using forward secret key exchanges for modern browsers. And the use of forward secret key exchanges is undoubtedly a vast security improvement from non-forward secret exchanges. However, our results are a reminder that cipher selection is only one part of the story.

Forward secrecy comes with many critical caveats and nuances of implementation [49]. The security community needs to do a better job of monitoring implementation behavior—through measurements like the ones we present here—in order to have a realistic understanding of the threats we need to guard against.

The security community also needs to more clearly communicate such caveats to TLS server operators and implementers so that they can make informed choices about security/performance tradeoffs. Absent such knowledge, there is a risk that forward-secret TLS handshakes can create a false sense of security. In the aftermath of the Heartbleed vulnerability, security experts urged administrators to enable PFS ciphers in order to guard against retrospective decryption as a result of future server-side memory leaks [19, 57]. However, only a few experts ever noted that performance enhancements like session resumption undermine that protection [49], and the fact seems to have been largely overlooked. The next time there there is such a vulnerability, administrators who enabled PFS as a defense might mistakenly believe they are safe.

One opportunity to begin such education is protocol standards. As described in Section 2.4, the TLS 1.3 draft proposes changes that have direct consequences for the protocol's vulnerability window. Draft 15 briefly addresses the changes to forward secrecy cased by PSK connections and 0-RTT, but simply sets a 7 day maximum for PSK lifetimes without discussion. As shown above, PSKs honored for 7 days (whether database lookups or encrypted state) require TLS secrets to exist for the same amount of time and may be a significant risk for high-value domains.

8.2 Server Operators Recommendations

For maximum security, server operators should disable all session resumption and Diffie-Hellman reuse. And while we are aware that many operators will be understandably unwilling to do so due to the bandwidth, computation, and latency advantages, there is a middle-ground between the two that limits vulnerability windows as well as allows the performance enhancements.

Use HTTP/2 Using HTTP/2 [5] drastically reduces the computation, bandwidth, and latency of loading a website without requiring any crypto shortcuts. An entire domain's contents (base page and all dependencies) can be loaded over a single TLS connection. This results in the time-to-first byte on the first request being identical to standard HTTP over TLS, but all follow-on requests are significantly faster without expanding the PFS vulnerability window.

Rotate STEKs frequently Reducing the time period that a STEK is used to encrypt session tickets is the simplest way to reduce the vulnerability window when using session ticket resumption. While Figure 3 shows that many domains are already doing this, it also shows that many are not. Twitter, CloudFlare, and Google have all created their own custom key rotation solutions [27, 33, 34], but, to our knowledge, no popular server software does this, with the exception of the most recent release of Caddy [28].

Use different STEKs for different regions Rather than sharing a single session ticket key globally, large sites should seek geographical diversity by using different keys in different regions. In addition to limiting exposure if a single server is compromised or physically attacked, this practice would help constrain the effects of legally mandated STEK disclosure to connections within a particular jurisdiction.

Reduce session cache lifetimes Specific to session ID resumption, quickly expiring cached session state is also useful. The number of connections that are at risk of decryption at any time grows proportionally with the lifetime of the server-side state. By measuring the duration of a typical user visit, operators can use that to ensure that a user only has to conduct one full handshake per visit but also refrain from retaining the session state longer than necessary.

Store, distribute, and erase secrets securely TLS implementations need to ensure that TLS secrets handled securely before, during, and after their use. For a small site, these details should be handled by the TLS implementation. But for more complicated deployments that involve synchronizing caches or STEKs across multiple servers, operators need to be more directly involved. Whatever mechanism they design to synchronize STEKs needs to ensure that these keys are transmitted securely and maintained only in memory (rather than persistent storage), so that they can be reliably discarded.

9. RELATED WORK

The HTTPS ecosystem has been widely studied. Previous work has tracked the configuration and deployment of HTTPS [18, 29] and community projects exist to provide up-to-date Internet-wide measurements of HTTPS servers and certificates [16, 17]. However, none of these works has directly measured support for TLS session resumption, and none has attempted to quantify the lifetimes of cached sessions, STEKs, or repeated Diffie-Hellman values. We build upon this prior research in a focused effort to understand and quantify the impact of these performance enhancements on forward secrecy and their effect on the overall attack surface of the HTTPS ecosystem.

Previous work has shown that some TLS implementations generate a single DHE value and reuse it for period of time. In May 2015, Adrian et al. [1] found that 17% of randomly sampled IPv4 hosts that had browser-trusted certificates reused a DHE g^a value at least once over 20 connections, and they noted that server-side DHE reuse was the default behavior in OpenSSL and Microsoft SChannel. In January 2016, OpenSSL entirely removed support for DHE reuse following CVE-2016-0701 [44]. Our work expands on the prior measurements by providing updated metrics following the OpenSSL change, by characterizing the lengths over which Diffie-Hellman values are repeated, and by also measuring reuse for ECDHE (now the most popular key exchange).

We are not the first to recognize the impact of TLS session resumption techniques on forward security. Representatives from Mozilla, Google, Twitter, and CloudFlare have all written about aspects of this issue [27, 33, 34, 54]. Instead, our work seeks to provide an empirical foundation for future discussions, system designs, and operator configuration choices by providing the first detailed global measurements about the use and impacts of session resumption and related TLS crypto shortcuts. To our knowledge, we are also the first to examine the effect of widespread inter-domain sharing of session secrets.

10. CONCLUSION

We conducted a 9-week study of HTTPS within the Alexa Top Million with a focus on understanding both the prevalence and characteristics of TLS performance enhancements such as (EC)DHE value reuse, session ID resumption, and session ticket resumption. Through this study, we were able to characterize the effects of cryptographic shortcuts on the promises associated with the use of forward-secret ciphers. Our findings show that the TLS ecosystem achieves much weaker protection from forward secrecy than statistics about support for forward-secret handshakes would suggest. They also emphasize the need for the security community to clearly communicate the relevant tradeoffs between security and performance to server operators.

Acknowledgments

The authors thank Adam Langley, Vern Paxson, Nick Sullivan, and our shepherd, Christo Wilson, for insightful discussions and feedback. We also thank the exceptional sysadmins at the University of Michigan for their ongoing help and support. This material is based upon work supported by the U.S. National Science Foundation under grants CNS-1345254, CNS-1409505, CNS-1518888, and CNS-1530915, by the NSF Graduate Research Fellowship Program under grant DGE-1256260, by the Post-9/11 GI Bill, by the Google Ph.D. Fellowship in Computer Security, and by an Alfred P. Sloan Foundation Research Fellowship.

11. REFERENCES

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[2] Alexa Internet, Inc. Alexa Top 1,000,000 Sites. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.

[3] J. Angwin, C. Savage, J. Larson, H. Moltke, L. Poitras, and J. Risen. AT&T helped U.S. spy on Internet on a vast scale. The New York Times, Aug. 16, 2015. http://www.nytimes.com/2015/08/16/us/politics/att-helped-nsa-spy-on-an-array-of-internet-traffic.html.

[4] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS with SSLv2. In *25th USENIX Security Symposium*, Aug. 2016. https://drownattack.com.

[5] M. Belshe, R. Peon, and M.Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540 (Proposed Standard), May 2015.

[6] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *36th IEEE Symposium on Security and Privacy*, May 2015.

[7] E. Burzstein and J. M. Picod. Recovering Windows secrets and EFS certificates offline. In *4th USENIX Workshop on Offensive Technologies*, Aug. 2010.

[8] Cavium. Intelligent network adapters. http://www.cavium.com/Intelligent_Network_Adapters_NIC4E.html.

[9] S. Checkoway, J. Maskiewicz, C. Garman, J. Fried, S. Cohney, M. Green, N. Heninger, R.-P. Weinmann, E. Rescorla, and H. Shacham. A systematic analysis of the Juniper Dual EC incident. In *23rd ACM Conference on Computer and Communications Security*, Oct. 2016.

[10] CNE access to core mobile networks. Media leak. https://theintercept.com/document/2015/02/19/cne-access-core-mobile-networks-2/.

[11] Codenomicon. The Heartbleed bug. http://heartbleed.com/.

[12] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. RFC 5280 (Proposed Standard), May 2008.

[13] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC 2246 (Proposed Standard), Jan. 1999.

[14] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. http://www.ietf.org/rfc/rfc5246.txt.

[15] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and cryptography*, 2(2):107–125, 1992.

[16] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. Censys: A search engine backed by Internet-wide scanning. In *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[17] Z. Durumeric, J. A. Halderman, et al. Internet-wide scan data repository. https://scans.io.

[18] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *13th ACM Internet Measurement Conference*, IMC '13, pages 291–304, 2013.

[19] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The matter of Heartbleed. In *14th ACM Internet Measurement Conference*, IMC '14, pages 475–488, 2014.

[20] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium*, Aug. 2013.

[21] R. Gallagher. Operation Socialist. The Intercept, Dec. 13, 2014. https://theintercept.com/2014/12/13/belgacom-hack-gchq-inside-story/.

[22] S. Gallagher. Googlers say "f*** you" to NSA, company encrypts internal network. Ars Technica, Nov. 2013. http://arstechnica.com/information-technology/2013/11/googlers-say-f-you-to-nsa-company-encrypts-internal-network/.

[23] B. Gellman and A. Soltani. NSA infiltrates links to Yahoo, Google data centers worldwide, Snowden documents say. The Washington Post, Oct. 30, 2013. https://www.washingtonpost.com/world/national-security/nsa-infiltrates-links-to-yahoo-google-data-centers-worldwide-snowden-documents-say/2013/10/30/e51d661e-4166-11e3-8b74-d89d714ca4dd_story.html.

[24] Google. Google for work: Enterprise solutions to work the way you live. https://www.google.com/work/.

[25] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Security Symposium*, Aug. 2012.

[26] K. E. Hickman. The SSL protocol, Apr. 1995. https://tools.ietf.org/html/draft-hickman-netscape-ssl-00.

[27] J. Hoffman-Andrews. Forward secrecy at Twitter, Nov. 2013. https://blog.twitter.com/2013/forward-secrecy-at-twitter.

[28] M. Holt. Caddy 0.8.3 released, Apr. 2016. https://caddyserver.com/blog/caddy-0_8_3-released.

[29] R. Holz, L. Braun, N. Kammenhuber, and G. Carle. The SSL landscape: a thorough analysis of the X.509 PKI using active and passive measurements. In *11th ACM Internet Measurement Conference*, IMC '11, pages 427–444, 2011.

[30] Internet Security Research Group. Let's Encrypt certificate authority. https://letsencrypt.org/.

[31] Jimdo. Website builder: Create a free website. http://www.jimdo.com/.

[32] D. Korobov. Yandex worker stole search engine source code, tried selling for just $28k. Ars Technica, Dec. 2015. http://arstechnica.com/business/2015/12/yandex-employee-stole-search-engine-source-code-tried-to-sell-it-for-just-27000-2/.

[33] A. Langley. How to botch TLS forward secrecy, June 2013. https://www.imperialviolet.org/2013/06/27/botchingpfs.html.

[34] Z. Lin. TLS session resumption: Full-speed and secure, Feb. 2015. https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/.

[35] I. Lovecruft. Twitter, Dec. 2015. https://twitter.com/isislovecruft/status/681590393385914368.

[36] M. Marquis-Boire, G. Greenwald, and M. Lee. `XKEYSCORE`: NSA's Google for the world's private communications. The Intercept, July 2015. https://theintercept.com/2015/07/01/nsas-google-worlds-private-communications/.

[37] J. McLaughlin. Spy chief complains that Edward Snowden sped up spread of encryption by 7 years, Apr. 2016. https://theintercept.com/2016/04/25/spy-chief-complains-that-edward-snowden-sped-up-spread-of-encryption-by-7-years/.

[38] media-34103. Media leak. http://www.spiegel.de/media/media-34103.pdf.

[39] P. Membrey, D. Hows, and E. Plugge. SSL load balancing. In *Practical Load Balancing*, pages 175–192. Springer, 2012.

[40] Microsoft. TLS/SSL settings, Nov. 2015. https://technet.microsoft.com/en-us/library/dn786418.aspx.

[41] mod_ssl: Apache HTTP server version 2.4. https://httpd.apache.org/docs/2.4/mod/mod_ssl.html.

[42] Module ngx_http_ssl_module. http://nginx.org/en/docs/http/ngx_http_ssl_module.html.

[43] Mozilla Telemetry. https://telemetry.mozilla.org/.

[44] OpenSSL security advisory, Jan. 2016. https://www.openssl.org/news/secadv/20160128.txt.

[45] PCS harvesting at scale. Media leak. https://theintercept.com/document/2015/02/19/pcs-harvesting-scale/.

[46] K. Poulsen. Snowden's email provider loses appeal over encryption keys. Wired, Apr. 2014. https://www.wired.com/2014/04/lavabit-ruling/.

[47] QUIC, a multiplexed stream transport over UDP. https://www.chromium.org/quic.

[48] E. Rescorla. The Transport Layer Security (TLS) protocol version 1.3 draft-ietf-tls-tls13-15, Aug. 2016. https://tools.ietf.org/html/draft-ietf-tls-tls13-15.

[49] I. Ristic. Twitter, Apr. 2014. https://twitter.com/ivanristic/status/453280081897467905.

[50] I. Ristic. SSL/TLS deployment best practices, Dec. 2014. https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices.pdf.

[51] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport layer security (TLS) session resumption without server-side state. RFC 4507 (Proposed Standard), May 2006. Obsoleted by RFC 5077.

[52] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport layer security (TLS) session resumption without server-side state. RFC 5077 (Proposed Standard), Jan. 2008.

[53] J. Schahill and J. Begley. The great SIM heist. The Intercept, Feb. 19, 2015. https://theintercept.com/2015/02/19/great-sim-heist/.

[54] T. Taubert. Botching forward secrecy: The sad state of server-side TLS session resumption implementations, Nov. 2014. https://timtaubert.de/blog/2014/11/the-sad-state-of-server-side-tls-session-resumption-implementations/.

[55] N. Weaver. In defense of bulk surveillance: It works, Sept. 2015. https://www.lawfareblog.com/defense-bulk-surveillance-it-works.

[56] K. Zetter. Google hack attack was ultra sophisticated, new details show. Wired, Jan. 2010. https://www.wired.com/2010/01/operation-aurora/.

[57] Y. Zhu. Why the web needs perfect forward secrecy more than ever. EFF Deeplinks Blog, Apr. 2014. https://www.eff.org/deeplinks/2014/04/why-web-needs-perfect-forward-secrecy.