Reproducing "Analysis of the HTTPS Certificate Environment"

CS 244 Final Project Report
Emily Marx and Timothy Le

Github URL: https://github.com/tle7/httpsCertEnv

1 INTRODUCTION

We aimed to reproduce "Analysis of the HTTPS Certificate Environment" [3] by Durumeric et al. At the time of this paper (2013), the authors produced the most comprehensive existing dataset of the HTTPS environment, with the aim of better understanding certificate authorities and leaf certificates. We found this paper interesting to reproduce because the scans were performed in 2013. In our 2019 scan, we explore changes in the certificate landscape over the past five years.

HTTPS relies on TLS to provide authenticated encryption. In turn, TLS uses certificates to securely associate a server's public key with its hostname. In order to convince the client that the key is correct, the certificate is signed by a chain of trusted entities known as certificate authorities (CAs). The chain begins with a certificate signed by one of a limited set of authorities known as root CAs contained in the client's browser or operating system. This root certificate in turn signs a certificate from an intermediate CA, which can then sign the next certificate in the chain. This continues until the leaf certificate, the one presented by the server.

While the list of root CAs can be easily obtained from the browser or OS, the identity of intermediate authorities is not publicly known. This has significant implications for TLS, given that the chain of trust is only as strong as its weakest intermediate CA. Durumeric et al. were motivated partly by this observation; several of the questions they explore concern the identity and characteristics of the intermediate CAs, and the certificates they sign.

In their 2013 paper, the authors investigate a wide range of questions, each broadly pertaining to one of two categories: Certificate authorities, and the leaf certificates themselves. They find that only two-thirds of hosts with port 443 open complete TLS handshakes, and that half of those responsive hosts presented trusted certificates. Additionally, according to their scans, a small set of CAs sign the vast majority of certificates, which is concerning given that the compromise of just one of these CAs would invalidate an enormous number of certificates. Moreover, the paper notes that half of browser-trusted certificates contain an RSA key shorter than the recommended 2048 bits in their chain.

We attempted to reproduce a subset of these results from both categories, namely the number of hosts responsive to SYN and TLS probes across the IPv4 space; the percentage of leaf certificates that are trusted; the share of trusted leaf certificates directly signed by each authority (Table 5); the distribution of RSA/DSA key lengths used in trusted leaf certificates and their chains (Table 8); and the signature algorithm used for trusted certificates (Table 10). In addition to these metrics from the original paper, we also recorded the TLS version used in the handshakes.

As in [3], we use ZMap [12] to perform a SYN scan of the IPv4 address space on port 443, storing the list of responsive hosts. We then attempt one TLS handshake with each responsive host using Zgrab2 [11]. Following each handshake, the leaf certificate, certificate chain, and verification results from Zgrab2 are parsed and the results stored.

We encountered an issue with Zgrab2, in which 96% of attempted TLS handshakes produced a "context-deadline-exceeded" error. The Zgrab2 developers indicated that this error had never previously been seen with the tool. We are still in talks with the developers to investigate the source. We performed a smaller scan, using the same ZMap/Zgrab2 commands, on a random sample of 100 hosts that had produced the error during the full IPv4 scan. In this random sample, none of the handshakes exhibited the error. Additionally, upon repeating our experiment by splitting the ZMap output of 70 IP addresses per file, the error was only encountered in 457 of 1,052,585 handshakes. Thus, it appears the error is a consequence of contacting a very large number of hosts in parallel. This issue also does not appear to be related to the computational power of our instances because we had this error appear on both a 4-vCPU instance and an 8-vCPU instance.

Based on discussions with another group (Andrew and William) that used the same tools for their project, we changed from running the tools in parallel to running them sequentially. From an earlier run of ZMap that we performed about two weeks ago, we had an earlier output from ZMap which listed 47,713,654 hosts which completed a TCP handshake on port 443. We divided this output into smaller files, and we passed each file sequentially through ZGrab2. From this experiment we collected (TODO: add number of responses and certificates), (TODO add number/percent of certificates received and number of context deadline exceed).
It should be noted that due to ZGrab2 errors, our sample size was significantly lower than the original paper’s. Overall, we find the TLS certificate ecosystem has undergone some changes since the original paper’s scans circa 2013, and in other ways has remained similar. The keys and signature algorithms used for browser-trusted certificates have become more secure, moving away from RSA 1024-bit and SHA-1, but there is still room for improvement. Both [3] and our scan found that key and signature algorithms lag behind the most up-to-date recommendations, namely ECDSA. However, we do not find the usage of blatantly outdated keys or algorithms, as [3] did with MD5/MD2 and 1024-bit RSA keys. In contrast to the improvement in keys and signature algorithms, the diversity of certificate authorities has remained concerningly low. As in 2013, the vast majority of certificates are signed by the most popular ten CAs. Also similar to the original paper was the proportion of hosts open on port 443 who completed a TLS handshake (around two-thirds). The proportion of certificates that were browser-trusted, however, has increased from 48% to 60%.

2 PRIOR WORK

In addition to Durumeric et al., there have been several investigations into the TLS certificate landscape. Most comparable is a 2011 study by Holz et al. [4], which gathered certificates via both active and passive measurement, the former via nmap and OpenSSL. However, the number of certificates gathered by Holz et al. was approximately 1.3% of that acquired by Durumeric et al., and Holz et al. focused on the most popular web servers, rather than the entire IPv4 space. As pointed out in [3], the most popular servers are not necessarily representative of the entire certificate ecosystem, since these hosts may be more likely to adhere to TLS standards.

Since [3] was published in 2013, additional investigations have taken place around the TLS certificate ecosystem. Much of this recent work has focused on security vulnerabilities in certificates [5], [2], perhaps motivated by recent high-profile TLS attacks such as FREAK and LogJam [10].

3 SYSTEM DESIGN

For this and following sections, we will call the scan that resulted in mostly “context-deadline-exceeded” errors as our initial scan. We will additionally refer to our smaller scan we did, after talking with another group, as our subsequent scan.

3.1 TCP and TLS handshake tools

The SYN scan for TCP handshakes for both scans was performed across the entire IPv4 space on port 443 using ZMap 2.1.1, a highly parallelized single-packet network scanner [12]. ZGrab2 [11], a Go-based scanner capable of performing stateful handshakes, including TLS 1.0, 1.1, and 1.2 handshakes with server extension (SNI).

In their 2013 paper, Durumeric et al. used bufferevents [7] rather than ZGrab2. Bufferevents is an event-driven C library with capability to perform concurrent TLS handshakes. The authors opened 2,500 concurrent TLS connections, and defined a callback to initiate a new handshake upon completion of a previous one. Initially, we wrote a similar framework. While this implementation did produce certificates and verification results, we were not confident in its integrity, particularly with regard to concurrent usage of shared variables. OpenSSL uses a variety of internal global variables [6], in addition to the global variables we used for data recording. OpenSSL’s documentation on this topic is quite sparse; a previous version of OpenSSL required users to define explicit locking callbacks for use with concurrent libraries like libevent, but such functions were deprecated in later versions of OpenSSL, and it was unclear if there was a replacement [9].

Additionally, in our talks with the original authors, they strongly discouraged the use of OpenSSL both for the lack of documentation around issues like these, and the improvements around error-checking and parallelization made by tools like Zgrab2. In addition, unlike Zgrab2, the original paper does not support SNI, an extension required by 31% of the Alexa top 100k servers [8]. Moreover, the original paper only briefly describes their bufferevents implementation, so we were not confident our use of the library was similar to theirs. Thus, we concluded that Zgrab2 would allow us to produce more credible results while maintaining a similar level of fidelity to the original paper’s methods.

One possible concern in reproducing some of the original paper’s results is that the original paper and ZGrab2 could have similar flaws that we would not be able to detect since we are using a tool from one of the original authors. We developed an initial script using OpenSSL to perform TLS 1.2 handshakes with 200 hosts from the ZMap output. These 200 hosts were the first 200 IPv4 addresses from a partial scan of ZMap randomly scanning the IPv4 address space. ZGrab2 outputted 130 leaf certificates, 75 of which were browser trusted. Our OpenSSL script resulted in 115 leaf certificates, 74 of which were browser trusted. When comparing the outputs, it appears that ZGrab2 is able to retrieve more leaf certificates because it checks for other protocols other than TLS 1.2, such as TLS 1.1 and TLS 1.0. Other than differences based on ZGrab2 checking more TLS versions, based on randomly comparing 15 results of this 200 IP host scan, we see that our ZGrab2 results match the results of our bash script. Based on ZGrab2’s robustness to TLS versions and it matching our own OpenSSL results, we believe that using ZGrab2 will have accurate results for the TLS scan.
3.2 System setup

Out initial IPv4 scan consisted of 5 Ubuntu 18.04 Google Cloud instances, each comprising 4 vCPUs and a 15 GB memory. Our subsequent experimental setup consisted of 2 Ubuntu 18.04 Google Cloud instances, each comprising 8 vCPUs and a 30 GB memory.

In both the initial IPv4 scan and subsequent smaller scan, we split the scan over multiple instances in order to maximize bandwidth. For the subsequent scan, we chose two fairly large instances to avoid running into potential resource problems. This size also seemed to be similar to the instances Andrew and William used. Splitting the subsequent scan across more than 2 instances would have increased bandwidth for the subsequent scan. However, we did not want to use the other 5 instances from out initial scan. Two had configuration issues, likely from transferring the instances from Emily to Timothy. The other three instances significant output from the initial scan, so we did not want to run into memory issues on these machines for the subsequent scan. Therefore, for the subsequent scan, we opted to use our 2 8-vCPU instances, which had sufficient memory left to hold the output of the scans, rather than spend time configuring instances from the initial scan or configuring new instances. It was a priority to run the subsequent scan promptly in order to collect as many responses as possible.

3.3 Experiment scanning details

In the initial scan, we used Zgrab2’s sharding mechanism to assign a random subset of the IPv4 space to each instance, dividing the IPv4 space into 5 partitions. For the initial scan, the list of responsive hosts was concurrently piped into ZGrab2. The instances used for this experiment used Go 1.12.5. Upon ZMap and Zgrab2 completion, and the json results were parsed in Python. The certificates were verified using the Ubuntu 18.04 root CA store. The open files limit was raised to 100,000 for the initial scan, allowing the many concurrent connections to open the root CA files simultaneously.

In the subsequent scan, as mentioned above, rather than running Zgrab2 and ZMap in parallel, we passed partitions of an earlier ZMap scan to Zgrab2. We divided the 47,713,654 hosts from ZMap’s scan, which occurred about two weeks ago, into files, each containing 70 IP addresses. We selected 70 files to input to Zgrab2 because we found this value to be close to the maximum files we could output to Zgrab2 before we received the "context-deadline-exceeded" error appear. Each of the 8-vCPU instances was responsible for half of these IP files containing 70 IP addresses. Due to time limitations, we could not complete this scan.

Table 5 shows the number of browser-trusted certificates signed by the top ten certificate authorities. Durumeric et al. found that the 92.4% of trusted certificates are directly signed by the top ten CAs. Our scan indicates a similar 89.7%. This is concerning for the security of the certificate ecosystem, since compromise of any one of these popular CAs would result in a large proportion of certificates becoming invalid. According to Adrian et al., ECDSA is recommended for maximal security, followed by RSA keys of at least 2048 bits [1]. In Table 8, we enumerate the keys used in all root and intermediate certificates belonging to the chains of browser-trusted certificates (Authorities column), as well as the key used in browser-trusted leaf certificates (Signed Leaves column). In the original paper, no leaf certificates were signed by ECDSA keys, and only 0.3% of chains contained an ECDSA key. In contrast, our scan found 5.9% of leaves signed with ECDSA.

4 RESULTS

From the 47,713,654 hosts we had from our earlier ZMap scan, we were able to retrieve the first 1,052,585 (2.2%) responses from the ZMap output list, in which ZMap contacted the hosts randomly. We therefore assume a random sample, and this was the largest sample we had time for after changing out scanning approach. From the hosts we were able to contact, 723,526 (68.7%) hosts completed a TLS handshake. This is comparable to the original paper, which found that 67% of hosts completing a TCP handshake on port 443 also completed a TLS handshake. From the hosts that completed a TLS handshake, a total of 440,432 (60.9%) browser trusted certificates. This percentage is higher than the 48% in the paper; we note that this percentage may have increased in the past 6 years.

Table A (not present in the original paper) lists the TLS/SSL version used in the handshake. ZGrab2 supports TLS versions 1.0-1.2, and allows the server to select the version. We find 84.4% of handshakes use TLS 1.2, with the rest using older versions, or SSL. As mentioned in future work, it would be interesting to add ZGrab2 support for TLS 1.3. In conversations with the authors, they expected around 30% of the most popular servers to support TLS 1.3, with the fraction being much lower over the entire IPv4 space.
The top 10 commercial certificate authorities control 92.4% of trusted certificates present in our March 22, 2013 scan.

<table>
<thead>
<tr>
<th>Organization</th>
<th>Signed Leaf Certificates</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoDaddy. com, Inc.</td>
<td>913,416 (28.6%)</td>
</tr>
<tr>
<td>GeoTrust Inc.</td>
<td>586,376 (18.4%)</td>
</tr>
<tr>
<td>Comodo CA Limited</td>
<td>374,769 (11.8%)</td>
</tr>
<tr>
<td>VeriSign, Inc.</td>
<td>317,934 (10.0%)</td>
</tr>
<tr>
<td>Thawte, Inc.</td>
<td>228,799 (7.2%)</td>
</tr>
<tr>
<td>DigiCert Inc</td>
<td>145,232 (4.6%)</td>
</tr>
<tr>
<td>GlobalSign</td>
<td>117,685 (3.7%)</td>
</tr>
<tr>
<td>Startfield Technologies</td>
<td>94,794 (3.0%)</td>
</tr>
<tr>
<td>StartCom Ltd.</td>
<td>88,729 (2.8%)</td>
</tr>
<tr>
<td>Entrust, Inc.</td>
<td>76,929 (2.4%)</td>
</tr>
</tbody>
</table>

Marx/Le Table 5: Top Certificate Authorities - The top 10 commercial certificate authorities control 89.7% of trusted certificates present in our 2019 scan.

and 2.1% of chains contained an ECDSA key. Durumeric et al. found 7.3% of chains and 4.2% of leaves containing a 1024-bit RSA key, which was not recommended at the time of the paper, but our scan found no such RSA keys. Moreover, while the majority of RSA keys in our scan were 2048 bits, 10.5% of chains and 3.6% of leaves used longer RSA keys.

Table 10 indicates the signing algorithm used for browser-trusted certificates. In Durumeric et al.’s 2013 scan, a small number of certificates (<1%) were signed with the insecure MD5 and MD2 algorithms. We found no such certificates. Since the time of [3], SHA-1 has been broken, and we note no certificates signed using SHA-1, where as the original paper found 98.7% of certificates using it. Instead, 94.3% of the certificates found in our scan use SHA-256 with RSA, and 5.6% use SHA-256 with ECDSA.

5 LIMITATIONS

The main limitation is the number of successful TLS handshakes. While our second scan resulted in far more handshakes and certificates than the initial scan, we would like to scan the full IPv4 space as in the original paper. To do so,
we will need to resolve the context-deadline-exceeded error, in order to obviate the need for manual sharding of host IPs.

In addition to the smaller sample size, manual sharding has another consequence. As described, the subsequent scan used the static output of a previous ZMap scan in order to allow manual division across files, rather than dynamically piping ZMap output into Zgrab2. This was necessary since the dynamic piping produced the context-deadline-exceeded error. However, it may result in a slightly smaller proportion of successful TLS handshakes, since the list of hosts responsive on port 443 changes slightly over time. The original authors indicated they would expect the use of static ZMap output to result in a 5-10% decrease in the number of successful TLS handshakes.

6 FUTURE WORK

This topic provides ample opportunities for future work. Most salient is the number of successful TLS handshakes. Ongoing investigation into the context-deadline-exceeded error indicates there may be some incompatibility between Zgrab2 and the instances used for this experiment, preventing the massively parallel IPv4 scanning for which Zgrab2 is designed. As of right now, we believe that this error is a result of ZGrab2 receiving too many files as input. We will be contacting the developers of Zgrab2 about how we saw the frequency of this error decrease.

We chose to limit the number of metrics we gathered, given our relatively small sample size. However, the authors examined many traits of certificates and CAs in addition to the ones we reproduced, such as X.509 constraints on name and path length. Another area of potential involves the root store. We used Ubuntu’s default store, whereas the authors compared certification validation results across root stores from Apple Mac OS 10.8.2, Windows 7, and Mozilla Firefox. With a larger sample, such features could be incorporated into our existing experimental framework relatively easily. We designed our certificate parsing tool to be extensible, so the main limitation currently is the lower number of certificates.

The authors performed 110 scans over a period of 14 months. This allowed them to identify temporal changes in the certificate landscape, as well as examine questions such as the persistence of individual certificates over time. Our limited time did not allow for such methods, but it may be interesting to investigate whether the landscape changes more quickly over time today than it did at the time of the paper.

Currently, Zgrab2 does not support TLS 1.3. TLS 1.3 involves several changes to certificates, including the removal of PKCS#1 v1.5 [10]. Future efforts could port TLS 1.3 support to Zgrab2, and investigate the percentage of hosts supporting TLS 1.3, along with the impact of TLS 1.3 on the certificate ecosystem.

REFERENCES