Turning Active TLS Scanning to Eleven

Wilfried Mayer, Martin Schmiedecker

▶ **To cite this version:**

Wilfried Mayer, Martin Schmiedecker. Turning Active TLS Scanning to Eleven. 32th IFIP International Conference on ICT Systems Security and Privacy Protection (SEC), May 2017, Rome, Italy. pp.3-16, 10.1007/978-3-319-58469-0_1 . hal-01649020

HAL Id: hal-01649020

https://inria.hal.science/hal-01649020v1

Submitted on 27 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Turning Active TLS Scanning to Eleven

Wilfried Mayer and Martin Schmiedecker

SBA Research
Vienna, Austria
{wmayer,mschmiedecker}@sba-research.org

Abstract. Transport Layer Security (TLS) is the fundament of today's web security, but the majority of deployments are misconfigured and left vulnerable to a phletora of attacks. This negatively affects the overall healthiness of the TLS ecosystem, and as such all the protocols that build on top of it. Scanning a larger number of hosts or protocols such as the numerous IPv4-wide scans published recently for a list of known attacks in TLS is non-trivial. This is due to the design of the TLS handshake, where the server chooses the specific cipher suite to be used. Current scanning approaches have to establish an unnecessary large number of connections and amount of traffic. In this paper we present and implemented different optimized strategies for TLS cipher suite scanning that, compared to the current best practice, perform up to 3.2 times faster and with 94% less connections used while being able to do exhaustive scanning for many vulnerabilities at once. We thoroughly evaluated the algorithms using practical scans and an additional simulation for evaluating current cipher suite practices at scale. With this work full TLS cipher suite scans are brought to a new level, making them a practical tool for further empiric research.

Keywords: Cipher Suite Scanning · SSL · TLS · Network Security

1 Introduction

Transport Layer Security (TLS) is the fundament of today's web security and provides confidentiality and authentication for application layer protocols like HTTPS, e-mail-related protocols or smartphone applications. Successful attacks against TLS are irritating the security community on a regular basis. Many of these attacks exploit vulnerabilities in the underlying cryptographic primitives, which, when grouped together, form so-called cipher suites. Often the mitigation of these vulnerarbilites is achieved by simply discontinuing the use of insecure cipher suites. Although easily done, this is a *manual* configuration step, which results in a slowly adopting TLS ecosystem. This progress is only observable through Internet-wide measurements.

Full cipher suite scans are important in order to understand in-depth the TLS ecosystem and the impact of discovered vulnerabilities, as demonstrated

recently [6,19]. Only with detailed information it is possible to thoroughly assess the state of online security, ranging from the security of a single host up to the security of the whole ecosystem. With the recent advent of fast-paced scanning tools it has become possible to proactively scan the entire range of IPv4 on a regular basis. This data is invaluable when reacting to newly released attacks.

In this work, we developed three new scanning algorithms that efficiently test TLS configurations in detail. These full cipher suite scans can then be used to cluster configurations based on different cipher suites, identifying common misconfigurations and facilitate TLS stack fingerprinting. We evaluated these algorithms and estimated the performance gain for Internet-wide full cipher suites scans. We then used these algorithms to scan parts of the IPv4-wide Internet and analyzed the results. The specific contributions of this paper are:

- We introduce highly optimized scanning methodologies to perform TLS scanning at scale.
- We evaluate our improved methodologies against the top-10k websites, and are on average 3.2 times faster.
- We show that current cipher suite recommendations are hardly used.
- We publicly release the source code and collected data from our experiments under an open source license[1].

The remainder of this paper is organized as follows: In Section 2 we present the relevant background as well as the body of related work. Section 3 introduces our optimized scanning methodologies and the data inputs used for our evaluation. Section 4 illustrates the achievable gain in overall performance and provides insights into the current TLS deployment. We discuss the results in Section 5 before we conclude in Section 6.

2 Background

TLS itself is specified in a variety of RFCs. The most important one is RFC5246 [7]. It defines the most modern version of TLS, version 1.2, introduced in 2008. Version 1.3 contains significant changes, but is still a working draft [24]. One of the goals of TLS is extensibility, i.e., the possibility of exchanging the used cryptographic functions. This is accomplished through the concept of cipher suites. Cipher suites are combinations of cryptographic primitives, defined as a two-byte value [17]. The used cipher suite and the TLS version are negotiated in the first two exchanged TLS messages (`client_hello`, `server_hello`). First, the client sends a `client_hello` message including a list of supported cipher suites. Second, the server replies with a `server_hello` choosing one of these cipher suites. This cryptographic primitives are subsequently used. A large number of cipher suites exists (over 140), and they can be used in different TLS versions (SSLv3,

[1] The patterns, the mappings and the source code are available online at: `https://github.com/WilfriedMayer/turning-active-tls-scanning-to-eleven`

TLSv1, TLSv1.1, TLSv1.2). This results in approx. 550 different combinations that can be tested.

Many security problems are caused by the use of old and deprecated features of TLS, e.g., the support of export-grade algorithms, old TLS versions or insecure ciphers. Two examples are POODLE [20] which is caused by the use of deprecated SSLv3 and DROWN [6] which is based on the active support of SSLv2. Some attempts to get rid of old cryptography were made, e.g., the ban of export-grade crypto in modern TLS versions or RFC7465 [22] that forbids the use of the insecure RC4 cipher. A secure deployment is non-trivial, therefore several guidelines give recommendations on how to (i) configure cipher suite settings and (ii) improve the configuration of TLS-enabled server applications [2,25]. However, these methods all rely on the administrator to actively improve the setup by changing the supported cipher suites manually – hence, the ecosystem is adopting slowly.

In the early days, *nmap* was used to perform these types of scan on a larger scale, but it is rather slow and does not scale to a larger number of hosts in reasonable time. A breakthrough was achieved with the development of *zmap* [11] and *masscan* [13]. Both tools use new methods to optimize large-scale scanning and are so far mainly used for port and vulnerability scanning. However, these improved methods are not applicable to fine-grained TLS scanning. With *zgrab* it is possible to establish TLS connections, but it is still not feasible for examining full cipher suite configurations. A more intense scanning behavior is necessary due to the design of TLS. Tools like *SSLyze* implement naive algorithms that conduct a full scan of all cipher suites by using one TLS handshake for each cipher suite. This is slow and produces a lot of traffic, thus a huge potential for optimization exists. With the results of this work, we are able to efficiently measure cipher suite configurations for TLS, also for large-scale studies.

2.1 Related Work

Prior studies that measured the TLS ecosystem focused primarily on the certificate ecosystem, the overall security was rarely evaluated. An early study was conducted by Lee et al. in 2007 [18]. With only 19,000 evaluated servers, this is a long way from an Internet-wide scale. Nevertheless, the size of measurement studies increased constantly, with larger studies conducted by the EFF [12] a few years later. Also, additional passive data was taken into account (Amann et al. [4,5]). With new scanning methods (e.g., *zmap* [11]), studies were suddenly able to cover the IPv4-wide Internet. These methods implemented new ideas, e.g., no per-connection state. This improved the speed and quality of large-scale scans. Studies that used this new scanning behavior are, e.g., the certificate ecosystem study by Durumeric et al. [9], that doesn't cover supported cryptographic primitives, and studies on vulnerabilities like Heartbleed [10] that solely

examine one exclusive issue.

Most of these studies focused on specific details in the configuration, e.g., the properties of a certificate. Fewer studies scanned all cryptographic primitives at once, i.e., all supported cipher suites. Huang et al. [15] describes the results of a complete cipher suite scan to measure perfect forward-secrecy support, but scanned only hosts from the Alexa top 1 million list [3]. Mayer et al. [19] performed cipher suite scans for all e-mail-related ports at an IPv4-wide scale. Both studies used naive algorithms to perform the scan. Other projects like the Qualys SSLTest [23] also scan full TLS configurations, but these projects are designed for single host configuration tests and not for Internet-wide studies. Newer studies tried to draw a complete picture of the certificate ecosystem [26] while missing the underlying security primitives, others decided against scanning full TLS cipher suites, because it would require to establish too many connections [14].

3 Methodology

To improve the scan rate for TLS-specific scanning, we defined the following requirements: First, *time* as the overall time consumption of the scanning process; second, the support for *parallelization* – can different scans be executed in parallel or do they rely on partial results and therefore require a sequential execution? These two requirements are especially important for large-scale scans. Third, the number of *connections* necessary for a scan: How many connections are necessary for a full configuration scan? Also, the generated traffic is derived from the number of connections. Lastly, the *completeness* of the scan, or how much information we can gather from the results: Is it possible to draw a complete picture of the TLS configuration or is it just one specific detail?

The anticipated use cases range from an interested system administrator or CISO who wants to scan infrastructure for security vulnerabilities up to Internet-wide scans for either specific questions or complete ecosystem analysis.

We identified three existing approaches to scan and identify TLS configurations: The *naive approach* establishes one connection for each cipher suite, starting at the same time. For each connection the server replies with either this cipher suite or with an alert that the cipher suite is not supported. This method is currently implemented by the command line tool *SSLyze* [1]. It highly parallelizes all requests, which results in a fast execution time, especially for non-delaying networks. The number of connections and produced traffic is rather large. This can lead to errors for some hosts, because the number of parallel connections may exceed their limit. This disadvantage leads to error-prone results and affects the completeness in a negative way. *SSLyze* (version 0.12) produces exactly 543 connection attempts to test all cipher suite/TLS version combinations and approx. 500KB of traffic (inbound and outbound) per tested host. These numbers

clearly don't scale for Internet-wide scans, making *SSLyze* impractical for this task.

The second approach is implemented by *zmap*. This command line tool, created by Durumeric et al. [11], is primarily used for Internet-wide port scans. With *zgrab* they also implemented an application layer scanner capable of scanning TLS configurations. To minimize the number of connections to exactly one per server, the cipher suites in the `client_hello` message are fixed to a specific research question, e.g., in order to test if RC4 is supported, all cipher suites that use RC4 are included. The server then responds with a `server_hello` message (showing the support of RC4) or an alert (showing that RC4 is not supported). This one-connection-based approach minimizes traffic and performs fast. The downside is that it does not completely evaluate all cipher suites. It is limited in its expressiveness, since only one question per scan can be evaluated.

The third approach is used by the *SSL Server Test* [23]. This web service is designed to test and evaluate one specific web server configuration. Therefore it utilizes the cipher suite settings from different browsers and browser versions as well as settings to test common misconfigurations. It then establishes one TLS connection per setting to completely evaluate one server configuration. It also includes HTTPS-specific settings and security features, e.g., HSTS or HPKP. The information collected is comprehensive, but the service's design is not suitable for large-scale ecosystem studies.

3.1 Introducing New Approaches

We propose the following new approaches for cipher suite scanning:

Connection-Optimal Approach This approach tests all cipher suites per TLS version in a serialized way. The process is illustrated in Figure 1. It starts with one TLS `client_hello` message that includes all available cipher suites for this TLS version. The server then responds with one cipher suite that it accepts. The next handshake includes all cipher suites except the one that was accepted earlier. This procedure is repeated until the server does not accept any of the offered cipher suites and responds with an alert. All remaining cipher suites are then evaluated as rejected. This approach uses the optimal, lowest number of connections necessary, but is not parallelizable for one host. Therefore it needs more time, especially for networks with a delayed round trip.

Grouped by Cryptographic Primitives The second approach, presented in Figure 2, groups cipher suites according to their used cryptographic primitives. It is based on the assumption that server operators disable or enable all cipher suites with a common primitive (e.g., deactivate all RC4-based cipher suites). After the cipher suites are split up in groups, the process follows the methodology of the connection-optimal approach. We currently use groups based on keywords in the cipher suite name, i.e., `SRP, PSK, EXP, NULL, (DSA, DSS), (ADH, AECDH), (CAMELLIA, SEED, IDEA, DES-CBC-), RC4`. Primitives that are

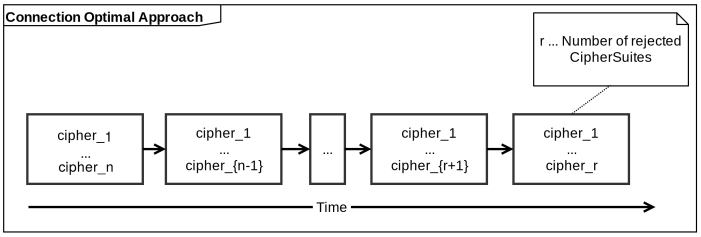


Fig. 1. Connection-Optimal Approach

not supported can be filtered out in the very first round. This approach supports parallel execution of the group tests so that it works with fewer round trips than the connection-optimal approach.

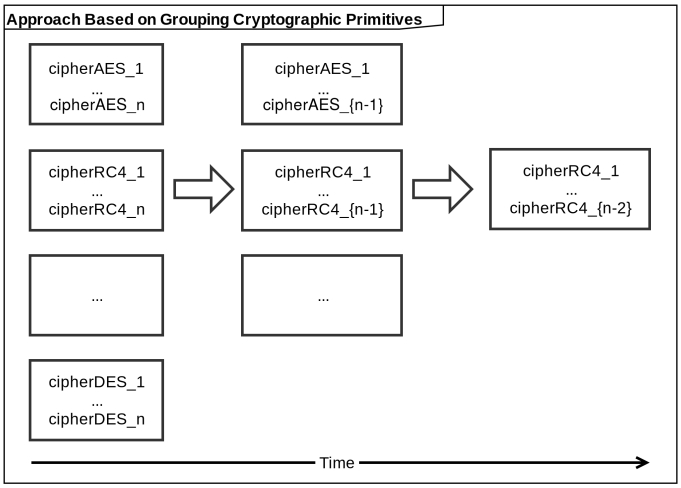


Fig. 2. Approach Based on Grouping Cryptographic Primitives

Based on Existing Results The third approach, as presented in Figure 3, combines the ideas from the former approach with data from already conducted cipher suite scans. It is based on the fact that many server operators use the same configuration, e.g., a default configuration. The most likely configuration based on former results is calculated before a `client_hello` is sent. After the first round of concurrent handshakes, an intermediary result is evaluated. Based

on this result, the next, most probable configuration is computed. The cipher suites used in the next round of parallel sent `client_hello` messages are then adjusted. This goes on until all cipher suites are either rejected or accepted. This approach is based on data described in the next paragraph.

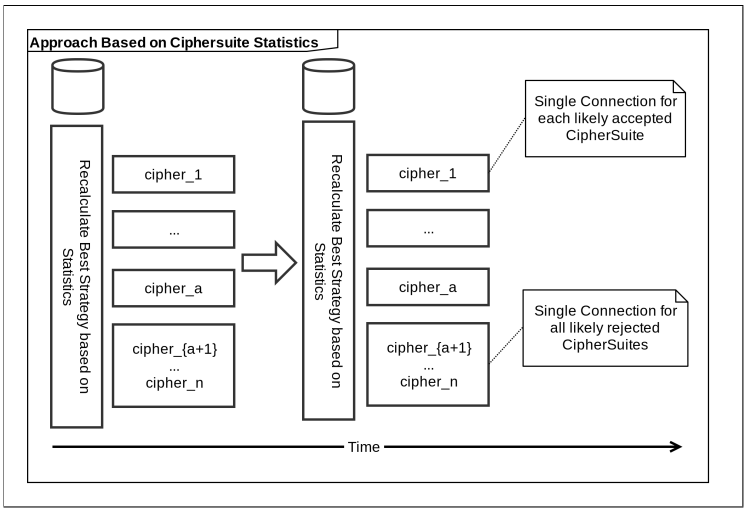


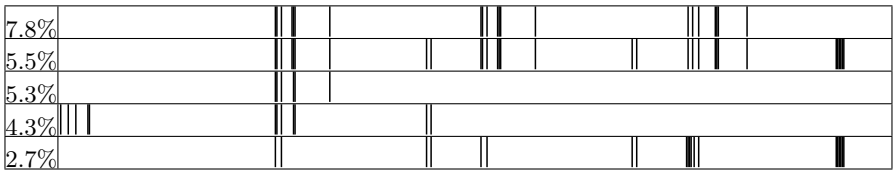
Fig. 3. Approach Based on Cipher Suite Statistics

Existing Data For the last algorithm, we rely on the dataset of an Internet-wide study we conducted from April to August 2015 [19]. We additionally use cipher suite scans of the HTTPS ecosystem, performed in August 2015. These datasets are very extensive w.r.t. the number of scanned cipher suites. Because of the large dataset (approx. 12 million error-free results), we transformed each result for each single host/port combination to a string. This string has a length of 551 characters[2]. This represents the total number of TLS version and cipher suite combinations. Each TLS version/cipher suite is either accepted or rejected, which is represented by the characters a respectively r. We did not take different behaviors of key exchange algorithms or different error messages (for rejected cipher suites) into account. In Table 1, the five most-used combinations for HTTPS are shown (a black bar represents an accepted cipher suite, a white bar a rejected cipher suite). We see that 7.8% of all hosts share one configuration in which all cipher suites for SSLv2 and SSLv3 are rejected and the supported cipher suites for TLSv1 and TLSv1.1 are identical. As an example, the first two bars represent

[2] 551 cipher suites were tested with *SSLyze* version 0.11. Because the underlying TLS implementation changed, version 0.12 does not test two specific cipher suites for four TLS versions, thus only 543 connections. Existing results for these cipher suites are ignored in the algorithm.

the accepted `AES128-SHA` and `AES256-SHA`, whereas the next cipher suites are rejected (`CAMELLIA128-SHA`, `CAMELLIA256-SHA`).

Table 1. Most-used cipher suite patterns for HTTPS, Internet-wide scan in Aug. 2015



When we take a closer look at the number of existing patterns per TCP port and the percentage of hosts that use these patterns, we see that a small number of patterns are used for most of the hosts. This is especially true for SMTP, where we see that the two most-used patterns cover more than 50% of all SMTP-enabled hosts. In Figure 4 the percentage of hosts that is covered by an increasing number of patterns for various TCP ports is shown. We assume that it is possible to optimize scanning methods by using this information. Also, the raw data of this patterns is publicly available.[3]

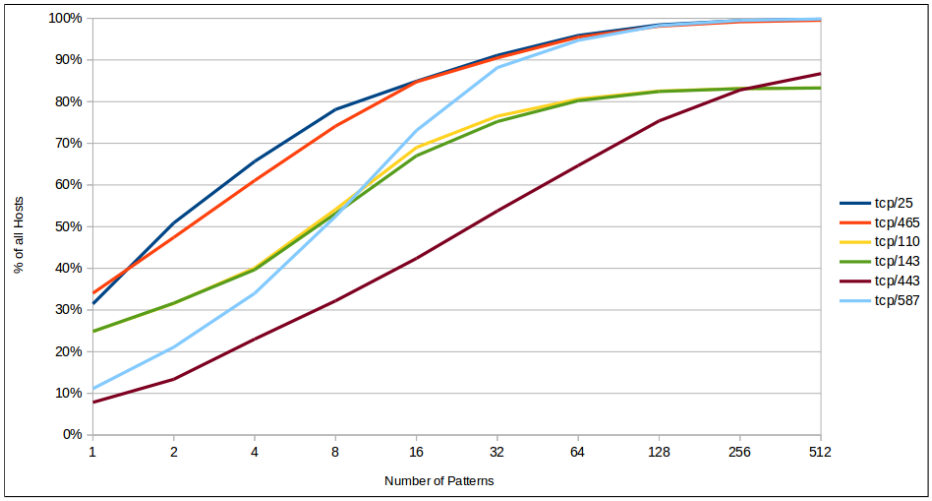


Fig. 4. Host Coverarage By Number of Patterns

[3] `https://scans.io/study/sba-email`

3.2 Implemented Approaches

We implemented all approaches by creating an additional mechanism to store partial results. Based on this partial result, the next requests are computed and executed, adding information to the partial result until it is complete. Users are able to choose the algorithm by specifying a command line argument (e.g., `--algorithm=connopt`). The required connections and time are logged for every run. Based on the existing data, we also implemented a simulation that calculates the number of necessary connections and rounds per approach. The complete source code is publicly available.

4 Results

We evaluated the proposed improvements by simulating an Internet-wide scan on IPv4 with existing scan data. We computed two performance values: The number of average established connections necessary to scan one host (C) and the number of average rounds (round trips) to scan one host (R). These two parameters are a good indicator for the defined requirements. Generated traffic and connections are directly mapped to C, the degree of parallelization and therefore the time needed is mapped to R. The results of this simulation are shown in Table 2. We can see that all new approaches use fewer connections than the naive approach. The optimum is achieved with the connection-optimal approach, although this method uses a lot of rounds and is therefore not parallelizable (and probably the slowest of all algorithms), except of the different TLS versions. Thus, the number of connections is five times bigger than the number of rounds. The group-based algorithm lies in between, with further potential to optimize the chosen groups. The algorithm based on existing data shows a low number of connections as well as a low number of rounds. For HTTPS on port 443 it minimizes the number of connections to an average of 37.3 (6.8%) with an average of 1.8 rounds. The simulation is based on the same dataset as the algorithm, so the expressiveness of this method will decrease in the future (as configurations change), but can be easily readopted with newer results.

Table 2. Comparison of Simulation Results with Existing Scan Data

	Port 25		Port 110		Port 143		Port 443	
	C	R	C	R	C	R	C	R
Naive	551.0	1.0	551.0	1.0	551.0	1.0	551.0	1.0
Connection-optimal	110.0	22.0	42.7	8.5	42.7	8.5	28.2	5.6
Crypto-group-based	252.0	7.3	199.9	3.8	199.8	3.8	187.7	2.4
Existing-data-based	141.0	1.5	52.3	1.5	51.4	1.4	37.3	1.8

4.1 Experimental Results

We tested the performance of our algorithms with scans in the wild. We used *SSLyze* version 0.12 and scanned a predefined set of hosts out of the Alexa Top10k list. We shuffled it and created batches of 100 hosts. With each algorithm we scanned 25 batches and measured the time needed and connections performed. We restrained from changing other aspects of *SSlyze*, like multiprocessing, multithreading or the general process. We also did not optimize kernel settings or other parameters on operating system level in order to compare only the algorithms with the default behavior. We used commodity hardware with an 100MBit/s uplink. The results are presented in Table 3. The naive approach performs worst in terms of speed. Also, connection-wise every new approach performs better than the naive approach. Although it has a more complex implementation, the approach based on existing data performs only slightly better than the algorithm based on crypto groups. Also listed in Table 3 are scans with a slightly larger set of hosts, used in the Section 4.2.

Figure 5 visualizes the large performance gain we can achieve with our approaches. It shows the average time for one host and the average number of connections per valid, scanned host of every tested batch.

Table 3. Experimental Results of the Different Approaches

	# Scans	Valid	Time (s)				Connections			
Approach	(Hosts)	Results	Total	Min	Avg	Max	Total	Min	Avg	Max
Naive	25 (100)	1,866	14,356	0.92	7.69	15.42	1,012,976	542.6	542.9	543.0
Connection-optimal	25 (100)	1,896	4,473	0.92	2.36	4.70	60,723	28.7	32.0	34.7
Crypto-group-based	25 (100)	1,914	5,462	0.64	2.85	4.96	351,534	182.1	183.7	185.8
Existing-data-based	25 (100)	1,870	4,672	0.50	2.50	5.98	268,814	126.1	143.8	156.4
Connection-optimal	5 (2,000)	9,262	5,951	0.56	0.64	0.75	314,398	33.3	33.9	34.4
Connection-optimal	5 (2,000)	7,534	9,493	1.13	1.26	1.38	244,644	31.7	32.5	33.5

These results show a large improvement in TLS cipher suite scanning algorithms. The connection-optimal algorithm is 3.2 times faster than the naive implementation (avg. connection-optimal compared with avg. naive) and uses only 6% of the connections (avg. connection-optimal compared with avg. naive) to execute a full TLS cipher suite scan in the wild. The connection-optimal approach and the group-based approach are correctly simulated, but we see that the results of the method based on existing data differ from the simulated results. We argue that this is due to two reasons: First, the algorithm and the simulation are based on the same data. If configurations change, the algorithm gets slower. The second reason is that we practically evaluated top-10k web services and not random hosts, whereas the simulation also considers a large number of small hosts.

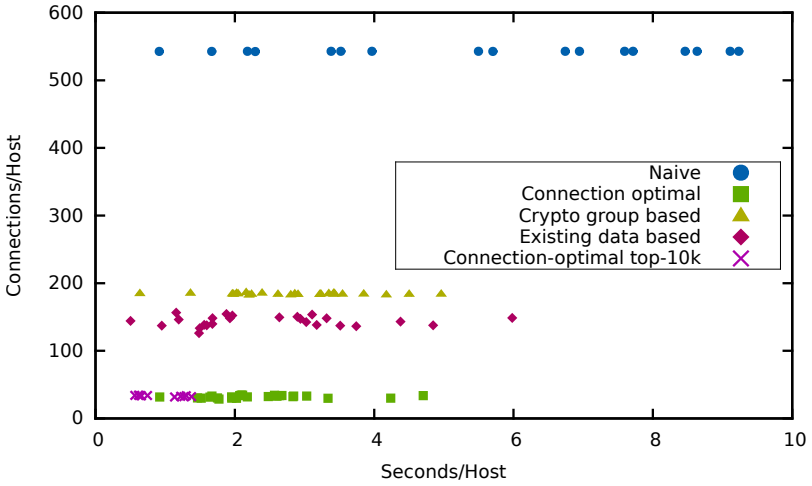


Fig. 5. Experimental Results of Different Approaches

Table 4. Cipher Suite Patterns

	Umbrella Top10k		Alexa Top10k	
e.g., xx.fbcdn.net	18.53%	1716	8.51%	641
e.g., google.com	13.43%	1244	6.15%	463
e.g., configuration.apple.com	7.63%	707	1.87%	141
Mozilla Modern Conf.	0.02%	2	0.05%	4
Mozilla Intermediate Conf.	0.98%	91	3.28%	247
Mozilla Old Conf.	0.35%	32	0.15%	11

4.2 Cipher Suite Results of Top-10k Domains

We used the connection-optimal algorithm to perform an additional cipher suite scan on the Alexa top-10k domains [3]. Cisco Umbrella recently proposed an alternative to the often-used Alexa Top 1 million list [16], so we decided to scan these top-10k domains as well. We analyzed which patterns occur, if these patterns are secure and if we can find a trend to common and secure TLS configurations. First, we looked at the most-used patterns in the Umbrella top-10k list. Although the three most-used patterns are used by 39.6% of the Umbrella top 10k resp. 26.9% of the Alexa top-10k, 524 and 954 (Umbrella/Alexa) different configurations exist. This indicates a highly diverse ecosystem. Second, we analyzed proposed cipher suite settings. Mozilla introduced a tool, the *Mozilla SSL Configuration Generator* [21], to generate secure configurations for various compatibility requirements, i.e., modern, intermediate and old. We see that the cipher suite pattern for a modern configuration is only used by 2 resp. 4 hosts in the top-10k lists. Their intermediate configuration is used by a recognizable number (91, 247). The exact numbers are also shown in Table 4. Third, we looked at differences between these patterns. All patterns disabled SSLv2 and SSLv3.

In contrast to the modern Mozilla configuration (only TLSv1.2), the other configurations support TLSv1 to TLSv1.2. In contrast to the intermediate configuration, TripleDES with DH key exchanges is not supported. The *xx.fbcdn.net* configuration is supporting more cipher suites (CAMELLIA, non-elliptic-curve Diffie-Hellman), whereas configurations like *google.com* support only one cipher suite more than configurations like *configuration.apple.com*, i.e., `AES256-SHA`. Finally, we tried to compare the results with pattern statistics we used for our simulation. We see that there are differences in the pattern usage, and we argue that the average top-10k host is differently configured than the average host from an Internet-wide scan.

5 Discussion

Internet scanning is not only a technical challenge. It also has to deal with ethical issues. Other studies already pointed out current best practices [11] which include to "scan no larger or more frequent than is necessary". This discouraged studies from performing a full scan, e.g., Holz et al. [14]. They stated that a full TLS cipher suite scan "is a poor trade-off in terms of good Internet citizenship versus lessons that can be learned". With our work, full TLS cipher suite scans can be conducted with less than 6% (approx. 32) of the connections compared to the currently used naive algorithm (543 connections). This minimizes the load each target host has to handle to a manageable minimum and makes the trade-off in terms of good Internet citizenship absolutely arguable. *Good Internet citizenship* is not only about minimizing the impact of one scan. It is also about avoiding unnecessary scans at all. One solution are publicly available results of scans, for which *Censys* [8], a search engine for Internet-wide scans, is a good example. They use the scanning approaches mentioned in Section 3, but an integration with the results of full TLS cipher suite scans is possible. We publish all our datasets and the source code.

In this work we optimized the methodology for full TLS cipher suite scans. For the practical evaluation we didn't change important factors of *SSLyze* to speed up the process. Important factors to optimize the bandwidth usage are, e.g., TCP port reuse, optimal settings for the TCP/IP stack or TCP connection reuse. The most influential factor is the parallelization of the scanning infrastructure. *SSLyze* (version 0.12) uses a maximum of 12 processes with 15 threads for all hosts; if the number of hosts is larger than that, the hosts are queued internally. This behavior is not optimal, since the server is idling. The solution is to split up all hosts amongst a large number of concurrent processes to minimize idling. With some optimizations applied, we were able to scan 27K hosts per hour with the naive approach on commodity hardware (100MBit/s uplink). We did not bundle these optimizations with our new approaches in order to focus on our comparison.

The approaches are created for TLS versions up to TLS v1.2. With TLSv1.3, which is currently a working draft, many things will change. Many insecure features are dropped, e.g., static RSA or DH key exchanges, insecure ciphers or hash-functions like MD5. Also, the handshake mechanism will be changed, so only one round trip is necessary to establish a full TLS connection. This – and also the question how TLSv1.3. is going to be deployed in the wild – affects the problem of how to efficiently scan full TLSv1.3 configurations.

6 Conclusion

In this paper we presented existing and new approaches for cipher suite scanning which is an important tool to evaluate the current status of the TLS ecosystem. Until now, naive approaches were used which are not optimal in terms of connections, scanning time or traffic transmitted over the wire. We introduced three new approaches that make use of the TLS protocol specification, common configurations and existing results. We evaluated the performance gain of these methods and found that we were able to perform scans 3.2 times faster with only 6% of the connections. We implemented a version of the described methods to work with a commonly used tool, simulated them and then evaluated them in practice by conducting a cipher suite scan for Alexa and the Umbrella top-10k hosts, describing the results and common patterns.

Acknowledgments. The research was funded by COMET K1 and by grant 846028 (TLSiP) from the Austrian Research Promotion Agency (FFG).

References

1. SSLyze - Fast and full-featured SSL scanner. `https://github.com/nabla-c0d3/sslyze`.
2. Applied Crypto Hardening. Online at `https://bettercrypto.org`, 2015.
3. Alexa Internet Inc. Top 1,000,000 sites. `http://s3.amazonaws.com/alexa-static/top-1m.csv.zip`.
4. B. Amann, R. Sommer, M. Vallentin, and S. Hall. No Attack Necessary: The Surprising Dynamics of SSL Trust Relationships. In *29th Annual Computer Security Applications Conference*, pages 179–188. ACM, 2013.
5. B. Amann, M. Vallentin, S. Hall, and R. Sommer. Revisiting SSL: A Large-Scale Study of the Internet's Most Trusted Protocol. Technical Report TR-12-015, ICSI, Dec. 2012.
6. N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, et al. DROWN: Breaking TLS using SSLv2. In *25th USENIX Security Symposium*, 2016.
7. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.
8. Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A Search Engine Backed by Internet-wide Scanning. In *22nd Conference on Computer and Communications Security*, pages 542–553. ACM, 2015.

9. Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS Certificate Ecosystem. In *13th ACM Internet Measurement Conference*, pages 291–304, Oct. 2013.
10. 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*, Nov. 2014.
11. Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *22nd USENIX Security Symposium*, Aug. 2013.
12. P. Eckersley and J. Burns. An Observatory for the SSLiverse. DEF CON 18 `https://www.eff.org/files/defconssliverse.pdf`, July 2010.
13. R. Graham. Masscan: the entire Internet in 3 minutes. `https://github.com/robertdavidgraham/masscan/`, `http://blog.erratasec.com/2013/09/masscan-entire-internet-in-3-minutes.html`.
14. R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kaafar. TLS in the wild: an Internet-wide analysis of TLS-based protocols for electronic communication. In *Network and Distributed System Security Symposium*, 2016.
15. L.-S. Huang, S. Adhikarla, D. Boneh, and C. Jackson. An Experimental Study of TLS Forward Secrecy Deployments. *Internet Computing, IEEE*, 18(6):43–51, 2014.
16. D. Hubbard. Cisco Umbrella 1 Million. 2016. `https://blog.opendns.com/2016/12/14/cisco-umbrella-1-million/`.
17. IANA. Transport Layer Security (TLS) Parameters. `https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4`.
18. H. K. Lee, T. Malkin, and E. Nahum. Cryptographic Strength of SSL/TLS Servers: Current and Recent Practices. In *7th ACM Internet Measurement Conference*, pages 83–92, Oct. 2007.
19. W. Mayer, A. Zauner, M. Schmiedecker, and M. Huber. No Need for Black Chambers: Testing TLS in the E-mail Ecosystem at Large. In *International Conference on Availability, Reliability and Security*, 2016.
20. B. Möller, T. Duong, and K. Kotowicz. This POODLE bites: exploiting the SSL 3.0 fallback. Security Advisory, 2014.
21. Mozilla. Mozilla SSL Configuration Generator. `https://mozilla.github.io/server-side-tls/ssl-config-generator/`.
22. A. Popov. Prohibiting RC4 Cipher Suites, Feb. 2015. RFC 7465.
23. Qualys SSL Labs. SSL Server Test. `https://www.ssllabs.com/ssltest`.
24. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-18. 2016.
25. Y. Sheffer, R. Holz, and P. Saint-Andre. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS), 2015.
26. B. Van der Sloot, J. Amann, M. Bernhard, Z. Durumeric, M. Bailey, and J. A. Halderman. Towards a Complete View of the Certificate Ecosystem. In *Internet Measurement Conference*, pages 543–549. ACM, 2016.