TLScompare: Crowdsourcing Rules for HTTPS Everywhere

Wilfried Mayer
SBA Research, Austria
wmayer@sba-research.org

Martin Schmiedecker
SBA Research, Austria
mschmiedecker@sba-research.org

ABSTRACT

For billions of users, today's Internet has become a critical infrastructure for information retrieval, social interaction and online commerce. However, in recent years research has shown that mechanisms to increase security and privacy like HTTPS are seldomly employed by default. With the exception of some notable key players like Google or Facebook, the transition to protecting not only sensitive information flows but all communication content using TLS is still in the early stages. While non-significant portion of the web can be reached securely using an open-source browser extension called *HTTPS Everywhere* by the EFF, the rules fueling it are so far manually created and maintained by a small set of people.

In this paper we present our findings in creating and validating rules for HTTPS Everywhere using crowdsourcing approaches. We created a publicly reachable platform at tlscompare.org to validate new as well as existing rules at large scale. Over a period of approximately 5 months we obtained results for more than 7,500 websites, using multiple seeding approaches. In total, the users of TLScompare spent more than 28 hours of comparing time to validate existing and new rules for HTTPS Everywhere. One of our key findings is that users tend to disagree even regarding binary decisions like whether two websites are similar over port 80 and 443.[1]

Keywords

HTTPS, TLS, HTTPS Everywhere, crowdsourcing, privacy, security

1. INTRODUCTION

One of the major problems in today's Internet is the lack of wide-spread HTTPS deployment: while it is a well-understood concept and many websites protect portions of their communication content with clients using HTTPS, e.g., login forms, only a minority of websites serve their entire content over HTTPS. While protocol extensions like HTTP Strict Transport Security [11] can be used to enforce HTTPS for specified domains, only few website operators deploy it [17]. This leads to numerous security- and privacy-related problems, as an active adversary can read, modify or redirect traffic at will during a man-in-the-middle (MITM) attack. This can be either a malicious ISP [13], a local attacker on the same network using ARP spoofing [14], or a malicious certificate authority [19]. In a recent paper it was demonstrated by Gmail that not only HTTPS is commonly targeted by active attackers. Also attacks against other protocols which use TLS [2], e.g., SMTP are encountered in the wild [9].

Server administrators are responsible for deploying HTTPS for a given website, and there is little that users can do to protect their communication content if servers do not support HTTPS. However, many websites employ HTTPS for a fraction of their pages, resulting in a valid certificate and proper HTTPS deployment – many times serving the same webroot over both protocols. The browser extension *HTTPS Everywhere* redirects otherwise unencrypted communication content to the encrypted counterpart where available, with a set of rules written in regular expressions. So far these rules are manually maintained, and while everyone can create his/her own rules, this approach clearly doesn't scale to the current size of the web. Therefore we present our platform in this paper, which tries to scale rule-creation and -verification using crowdsourcing.

The remainder of this paper is divided into the following sections: Section 2 gives the essential background on current HTTPS deployment and HTTPS Everywhere. Section 3 introduces our approach for crowdsourcing rule generation. Section 4 presents our implementation for the collection of these rules. Section 5 discusses our findings before we conclude in Section 6.

2. BACKGROUND

Today Transport Layer Security (TLS) is responsible for the majority of encrypted online communication, not only for HTTPS but also numerous other protocols. Compared

[1] This is a preprint of the paper to appear at the Workshop on Empirical Research Methods in Information Security (ERMIS) 2016, to be held with WWW'16.

[2] In this paper we use the term TLS to refer to all incarnations of the TLS and SSL protocols, if not specified otherwise.

ACM ISBN 978-1-4503-2138-9.
DOI:

to plaintext communication it provides confidentiality, authenticity and integrity. Currently, TLS 1.2 [8] is the most recent version of the SSL/TLS protocol family, with TLS 1.3 on the horizon. Many of the problems of TLS have been discussed in the literature [7, 21], and multiple guidelines on how to securely deploy TLS using best-practice methods have been published [22, 3].

While HTTPS Everywhere is a solution that has to be installed in client browsers, there are also server-side solutions available to prevent cleartext communication despite server-side redirects from TCP port 80 to TCP port 443. HTTP Strict Transport Security (HSTS) [11] for example was standardized to enforce the use of HTTPS. Some browsers go even one step further by shipping the software with predefined HSTS lists to enforce HTTPS before the first use [16].

2.1 HTTPS Everywhere

HTTPS Everywhere[3] is a browser extension for Firefox, Chrome and Opera which is developed by the EFF and the Tor project. It ensures that a secure connection is used if available by rewriting URLs in the browser before attempting to start a connection over HTTP. This is helpful in the particular use case: a secure channel is available, but not used by default, and the server operator does not by default employ HTTPS for all users and communication content. The rules for HTTPS Everywhere are written using regular expressions which specify the target domains as well as the secure counterparts that ought to be used instead. Listing 1 presents a simple rule for the domain torproject.org. Not only does it rewrite all connections to http://torproject.org, but also all valid subdomains with the * wildcard.

```
<ruleset name="Tor Project">
  <target host="torproject.org" />
  <target host="*.torproject.org" />
  <!-- excluded content -->
  <rule
    from="^http://([^/:@\.]+\.)?torproject.org/"
    to= "https://$1torproject.org/" />
</ruleset>
```

Listing 1: HTTPS Everywhere Rule for torproject.org[4]

These rules are manually developed and maintained using a public git repository. At the time of writing the repository contains approx. 18,400 files that specify rewrite rules. The process of manually creating rules is justified by the possible complexity of rewrite rules for particular deployments, yet simple rewrite algorithms do sometimes not know how to handle these deployments correctly. During the compilation of the browser extension, these xml files are merged into a single SQLite file. Table 1 shows that the majority of the existing rules of HTTPS Everywhere are in fact really simple. 13,871 files contain only one rewrite rule and of all rules 73% do not reference the http URL (i.e., are not using $ in the regular expression). 28% use the trivial rewrite rule “http:” to “https:” as listed in Listing 2 and only 1.6% use a more complicated rule with two references. The “exclusion” element is used very seldom.

```
<ruleset name="example.org">
  <target host="example.org" />
  <rule from="^http:" to="https:" />
</ruleset>
```

Listing 2: Trivial rewrite rule

Files with 1 rule	13,871
Files with 2–10 rules	4,496
Files with more than 10 rules	44
Total rules	26,522
Trivial rules	7,528
Rules without reference ($)	19,267
Rules with more than one reference ($)	417
Files with no exclusion	17,005
Files with one exclusion	1,136
Files with more exclusion	272
Files with no securecookie	8,597
Files with one securecookie	9,215
Files with more securecookie	601

Table 1: Ruleset statistics

2.2 Related Work

Recent publications regarding TLS in general focus firstly on the insecure deployment of TLS in the field, and secondly on new cryptographic attacks and their possible impact. Problems range from the use of old and deprecated SSL/TLS versions [12], to the use of insecure cryptographic primitives like export-grade ciphers [18, 6] or weak RSA keys in certificates [4], to the use of TLS implementations with known bugs like Heartbleed [10]. To the best of our knowledge we are the first to discuss the creation of rules for HTTPS Everywhere using automatic and semi-automatic means.

3. DESIGN

We split the design of automated rule generation in two parts. First, suitable rules are generated with a rather simple algorithm. Second, these rules are validated by matching the similarity of the encrypted and unencrypted content with crowdsourcing methods.

3.1 Automated Rule Generation

Many domains are deployed with a simple HTTPS configuration. For these domains rules can be automatically generated with tool support. HTTPS Everywhere provides a script to generate trivial rules for these domains (called `make-trivial-rule`). Rulefiles generated with this script include one or more targets and the simple rule as listed in Listing 2. We can identify hosts that support HTTP and HTTPS via Internet-wide scanning techniques. Hosts that use HSTS or server-side redirects can also be efficiently identified and excluded from further processing. We then use a rather easy approach of rule generation by creating trivial

[3] https://www.eff.org/https-everywhere

[4] https://github.com/EFForg/https-everywhere/blob/master/src/chrome/content/rules/Torproject.xml

rules for all domains. We also include often-used subdomains such as “www” and “secure”. Also, the rank from the Alexa Top 1 Million ranking [1] is taken into consideration.

The most important requirement for new HTTPS Everywhere rules is that we want to avoid corrupt rules under all circumstances. If error-prone rules reduce user satisfaction, the plugin may get uninstalled. Of course the assumption that two pages serve the same content is naive, although often true. We want to avoid corrupt rules, we have to further prove the similarity of encrypted and unencrypted content.

3.2 Rule Validation

The question whether the content of two pages is similar is not easy to answer. We can compare the equality of two pages by comparing their hash values, but if, e.g., only the server name differs in http and https, the pages will not share the same hash value despite being similar. This problem continues with dynamically generated content. The pages can be similar, indicating a valid rule, but they are not equal, e.g., if they display different ads or other dynamically loaded content. To measure the similarity and thus conclude if a rule is valid or not, we identify two possibilities: First, similarity-matching algorithms. These algorithms can be based on the HTML source code, visual recognition, or other metrics. Second, crowdsourcing approaches, i.e., we use the input from a large group of people who voluntarily compare pages. To facilitate this approach we developed a web service that is easy to use and publicly available. We can then use crowdsourcing results to measure the correctness of rules and analyze the reasons why rules are invalid.

4. CROWDSOURCING

In order to crowdsource the validation of rules we created the project *TLScompare.org*. This web service facilitates the rewrite rule validation process by providing an easy-to-use interface. As illustrated in Figure 1, this minimalistic interface offers three buttons: one to start the comparison and two to choose the result. After clicking the button “Compare!” two separate browser windows appear. The left window displays the non-secured (HTTP) version of a web page, the right window displays the secured (HTTPS) version. These two different URLs represent the “from” and “to” value of the regular expression. The browser windows are opened without a menubar or a scrollbar. The windows are also equally sized and do not overlay the buttons of the main window. After comparing the two pages the user has two choices in the normal operation mode. This is illustrated in Figure 2.

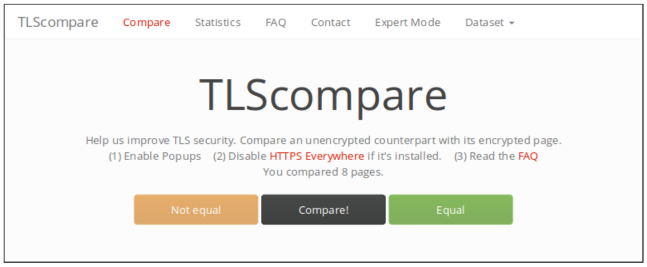
Not equal The two pages differ in some way.

Equal The two pages are visually equal.

The meaning of these choices is explained in the FAQ section. After the user chooses one result, both windows close automatically and the next comparison can start.

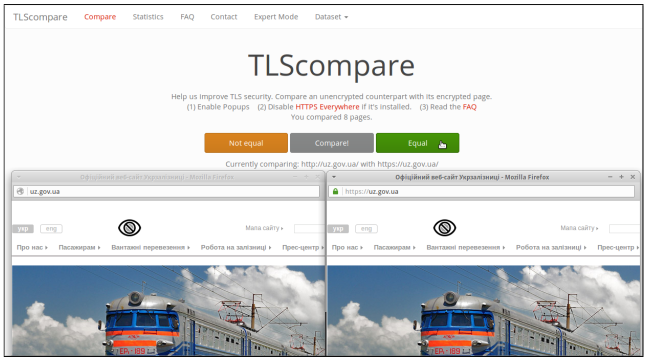
As shown in Figure 3, we also introduced an “expert mode” to analyze the differences on a more fine-grained level. The expert mode enables a variety of result choices, specifying the detailed reason. Possible choices with short explanations – also shown in the corresponding tooltip – are:

Certificate Mismatch Certificate URL is different to URL.



Help us improve TLS security. Compare an unencrypted counterpart with its encrypted page.
(1) Enable Popups (2) Disable HTTPS Everywhere if it's installed. (3) Read the FAQ
You compared 8 pages.

Figure 1: Normal operation mode



Help us improve TLS security. Compare an unencrypted counterpart with its encrypted page.
(1) Enable Popups (2) Disable HTTPS Everywhere if it's installed. (3) Read the FAQ
You compared 8 pages.

Currently comparing: http://uz.gov.ua/ with https://uz.gov.ua/

Figure 2: Comparison screen

Untrusted Certificate Expired certificate, or untrusted by browser.

Mixed content Pages do not look similar due to mixed content (CSS, JS).

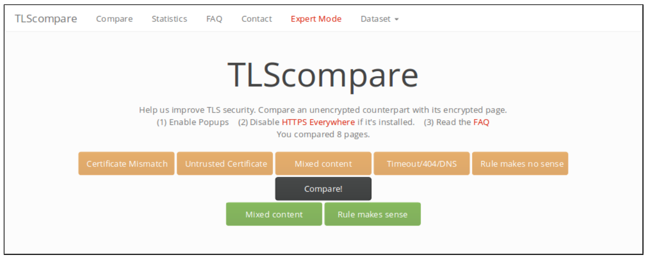
Timeout/404/DNS Network related errors on either page.

Rule makes no sense No use in upgrading: pages are not similar or either page redirects, e.g., from HTTP to HTTPS (or vice versa).

Mixed content Pages look similar, but mixed content warning.

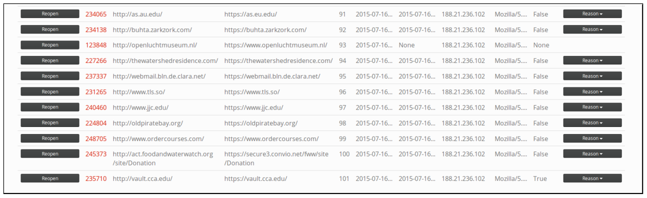
Rule makes sense Pages look similar, no HTTPS warnings.

The user is able to compare pages for different datasets, while we are able to set different default datasets. We also included an administration panel as illustrated in Figure 4.



Help us improve TLS security. Compare an unencrypted counterpart with its encrypted page.
(1) Enable Popups (2) Disable HTTPS Everywhere if it's installed. (3) Read the FAQ
You compared 8 pages.

Figure 3: Expert operation mode

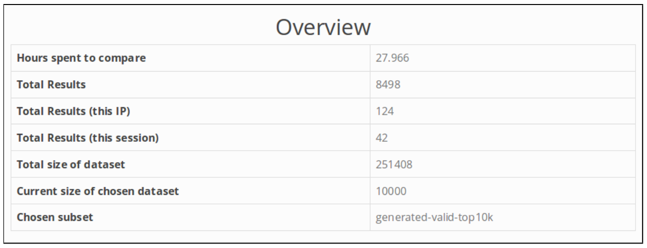


Reopen	234065	http://as.au.edu/	https://as.eu.edu/	91	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	234138	http://buhta.zarkzork.com/	https://buhta.zarkzork.com/	92	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	123848	http://openluchtmuseum.nl/	https://www.openluchtmuseum.nl/	93	2015-07-16...	None	188.21.236.102	Mozilla/5....	None	
Reopen	227266	http://thewatershedresidence.com/	https://thewatershedresidence.com/	94	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	237337	http://webmail.bln.de.clara.net/	https://webmail.bln.de.clara.net/	95	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	231265	http://www.tls.so/	https://www.tls.so/	96	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	240460	http://www.jjc.edu/	https://www.jjc.edu/	97	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	224804	http://oldpiratebay.org/	https://oldpiratebay.org/	98	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	248705	http://www.ordercourses.com/	https://www.ordercourses.com/	99	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	245373	http://act.foodandwaterwatch.org /site/Donation	https://secure3.convio.net/fww/site /Donation	100	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	False	Reason ▾
Reopen	235710	http://vault.cca.edu/	https://vault.cca.edu/	101	2015-07-16...	2015-07-16...	188.21.236.102	Mozilla/5....	True	Reason ▾

Figure 4: Administration panel

Technically TLScompare is developed as a Python WSGI application behind a nginx webserver. Flask [20] is used as web framework and the results are stored in a relational SQLite3 database via SQLAlchemy [5]. All important methods are exposed via REST endpoints. This setup is hosted at https://tlscompare.org. For every result we store the IP, the User Agent, the result, the reason (if classified in expert mode) as well as the timestamps of the original request and original result. Each validation attempt is identifiable via an unique id and connected to a session identifier. We are able to filter out bogus results via wrong session identifiers and a too short duration between request and result.

To improve the user motivation we included basic statistics for the current session and the global scope as depicted in Figure 5. The user is able to compare the number of results to the global scope or other users.



Overview

Hours spent to compare	27.966
Total Results	8498
Total Results (this IP)	124
Total Results (this session)	42
Total size of dataset	251408
Current size of chosen dataset	10000
Chosen subset	generated-valid-top10k

Figure 5: Statistics on global and session scale

4.1 Acquired Data

At the time of writing we have acquired 8,523 validation attempts and 7,616 actual results. An overview of the acquired data is presented in Table 2.

Unfinished attempts	907
Actual results	7,616
Number of different UserAgents	125
Number of different sessions	268
Min results per session	1
Max results per session	456
Average results per session	28.4
Median results per session	6
Sum of time between request and result	27.98h

Table 2: Results statistics

We introduced different datasets from which to choose:

Existing rules Comparisons of unencrypted pages that are currently rewritten by HTTPS Everywhere. We computed these URLs by "reversing" the regular expressions of rules. For simplicity we only generated rules out of regular expressions that include different subdomains, but don't contain more complicated regular expressions.

Generated rules We also generated values to compare domains of the Alexa Top 1 Million domain ranking. We used comparisons for second-level domains and for common third-level domains, e.g., www, secure, or ssl. We also filtered out a subset with the 10,000 highest-ranked domains.

We expect these datasets to yield completely different values. For the existing rule set, we expect the majority of results to have similar pages. Statistics for this dataset are presented in Table 3. We see that there are, against our expectations, more "Non Equal" results. We therefore checked all 226 "Not Equal" results that were set without a reason. We found that 106 of these domains deployed a default server-side redirect and were set before we changed "Not Equal (Rule makes no sense)" in expert mode to explicitly save this reason. We assume that these results were generated by users that realized that additional client-side rules make no sense although the pages look similar. The remaining results must be checked separately.

Total attempts	543
Total results	527
Equal	169
Not Equal (Total)	358
Not Equal (Without reason)	226
Not Equal (Timeout)	58
Not Equal (Certificate Mismatch)	40
Not Equal (Untrusted certificate)	14
Not Equal (Rule makes no sense)	15

Table 3: Dataset for existing HTTPS Everywhere rules

We also created a dataset of pages for the Alexa Top 1 Million ranking. We used a rather naive algorithm to create new rules which were then compared. The results of this dataset presented in Table 4. We see that 74% of all rules were marked with "equal". This number is conform to our assumption that many hosts serve similar pages on the encrypted and unencrypted channel.

Total attempts	2,600
Total results	2,267
Equal	1,688
Not Equal	579
Not Equal (mixed-content)	87

Table 4: Dataset for similar Alexa Top 10k domains

Table 5 presents the validity results per created rule. Each "1" represents one comparison where the pages were marked as equal, each "0" represents a non-equal result. It also shows comparisons where more than one result was submitted. For comparisons with two results, 87% have the same result and 13% differ in the result. Apart from users that submitted wrong results, this can have multiple reasons, e.g., the pages look different in some browsers or the similarity changed over time. For comparisons with three results, 86% have the same result and 14% have one discordant value.

Combination	Number of Results
0	842
1	443
00	612
10	148
11	394
000	67
100	6
110	10
111	32
0000	9
1000	2
1100	2
1110	1
1111	5
00000	1

Table 5: Multiple results to ensure data quality

4.2 Deriving rules

With this data we can automatically generate HTTPS Everywhere rules from results that have been checked multiple times and yielded the same result. In the future, we have to find the correct threshold; currently we assume that comparisons that yield the result "Equal" for at least three times can be taken for sure. We currently have 38 results that fulfill this requirement. The creation realized by selecting all candidates, passing them to a generation script and creating rules as presented in Listing 3, where *BASEADDRESS* is the domain name of the unencrypted part, and *ENCRYPTED* is the domain name of the encrypted URL. These rules are then ready to be inserted in the HTTPS Everywhere repository.

```
<ruleset name="BASEADDRESS">
  <target host="BASEADDRESS" />
  <rule from="^http://BASEADDRESS"
        to="https://ENCRYPTED" />
</ruleset>
```

Listing 3: Trivial rewrite rule

5. DISCUSSION

We started our project as non-commercial, although several commercial crowdsourcing frameworks exist. The probably most famous variant is Amazon Mechanical Turk [2]. It commercializes so called "Human Intelligence Tasks" (HITs). Each HIT is solved by a human worker who gets paid with a small amount of money. One page comparison can be modelled as HIT, and with this platform we could easily enlarge our dataset. With our non-commercial project we were able to collect 7,616 results in a timespan of approx. 5 months. We announced our project solely on Twitter and held one lightning talk at a community-driven event. We assume that the quality of our data is comparable to data from other crowdsourcing frameworks. We also assume that we would have to include more sophisticated methods to ensure data quality if we also included money-driven results with commercial crowdsourcing providers.

To further increase the usage of encrypted connections for data transmission, other problems have to be solved first. This includes an easy way to deploy server-side HTTPS and ensure trust in the used certificates. A promising project currently dealing with this issue is Let's encrypt [15]. Server-side redirection is making the effort of client-side rewrite rules worthless, and together with the server-side enforcement of HTTPS via HSTS [11] it is clearly the favorable solution.

5.1 Future Work

We currently investigate the use of different algorithms to automatically measure similarity. We have to identify usable algorithms, but the question of similarity in the context of HTTPS support is not easy to define. Can an URL be rewritten to its encrypted counterpart if the content is not equal? If some content of the page is randomly displayed, but the functionality stays the same? Is the content similar if the code is similar or if the rendered images are similar? Different technical methods are possible: simple word matches, comparisons of the Document Object Model, algorithms that compare images of rendered pages amongst others. However, similarity matching algorithms will also yield false positives or create borderline cases. TLScompare can then be used to crowdsource fine tuning of these algorithms or crowdsource manual checking of the results.

We will also adopt TLScompare for a student exercise. After students manually create new rules for HTTPS Everywhere, TLScompare will be used to validate these rules.

Weighting of user quality is currently left out, but is another measure to ensure data quality. Users are not identifiable, but the results are stored with a session id, the User Agent and the IP address. So if one user misjudges more results, the weight of this session is reduced accordingly. Also, the average compare time, i.e. the duration between request and result, is one property that can be used to calculate the user's credibility.

Another future issue is to optimize the selection algorithm for the next comparison. An improved version can immediately choose the same comparison for another user to increase comparisons with more than one result. This is especially true for larger datasets where the probability of multiple results is low.

6. CONCLUSION

In this paper we presented a new approach to create and validate rules for HTTPS Everywhere. We showed that the currently used approach doesn't scale and does not use the large number of simple-configured HTTPS deployments. We presented the design of our webservice TLScompare and explained how crowdsource approaches can be used to generate and validate rules. We implemented the service, deployed it in publicly available way and collected over 7,500 results for different datasets. We took a look at the first results for existing comparisons of HTTPS Everywhere rules and identified problems of the user's understanding of different terms. We tested newly generated rules and showed that it is possible to consider a rewrite rule to be valid with the use of multiple results per comparison. These rewrite rules were than transformed in an XML representation that is to be submitted to HTTPS Everywhere. We identified problems with the quality of our data and suggested methods of ensuring high data quality with statistical methods. Other approaches of rule generation and additional use cases were considered. The fundamental problem regarding the lack of existing HTTPS deployments was discussed.

Thanks to our project it is possible to simplify the process of HTTPS Everywhere rule generation. This increases the content transmitted exclusively over HTTPS for users of this plugin and makes a small step in the direction of an encrypted Internet.

Acknowledgements

This work has been supported by COMET K1, FFG - Austrian Research Promotion Agency and under grant no. 846028.

7. REFERENCES

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

[2] Amazon Mechanical Turk. https://www.mturk.com/mturk/welcome.

[3] Applied Crypto Hardening. Online at https://bettercrypto.org, 2015.

[4] 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 Conference on Computer and Communications Security*. ACM, 2015.

[5] M. Bayer. SQLAlchemy - The database toolkit for python. http://www.sqlalchemy.org/.

[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 *Symposium on Security and Privacy*. IEEE, 2015.

[7] J. Clark and P. C. van Oorschot. SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *Symposium on Security and Privacy*, pages 511–525. IEEE, 2013.

[8] 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.

[9] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman. Neither Snow Nor Rain Nor MITM...: An Empirical Analysis of Email Delivery Security. In *15th Internet Measurement Conference*, pages 27–39. ACM, 2015.

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

[11] J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security (HSTS). RFC 6797 (Proposed Standard), Nov. 2012.

[12] 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*. Internet Society, 2016.

[13] R. Holz, T. Riedmaier, N. Kammenhuber, and G. Carle. X. 509 Forensics: Detecting and Localising the SSL/TLS Men-in-the-middle. In *European Symposium on Research in Computer Security*, pages 217–234. Springer, 2012.

[14] M. Huber, M. Mulazzani, and E. Weippl. Who on earth is "Mr. Cypher": automated friend injection attacks on social networking sites. In *Security and Privacy – Silver Linings in the Cloud*, pages 80–89. Springer, 2010.

[15] Internet Security Research Group. Let's Encrypt - Let's Encrypt is a new Certificate Authority. https://letsencrypt.org/.

[16] D. Keeler. Preloading HSTS. Mozilla Security Blog - https://blog.mozilla.org/security/2012/11/01/preloading-hsts, 2012.

[17] M. Kranch and J. Bonneau. Upgrading HTTPS in Mid-Air: An Empirical Study of Strict Transport Security and Key Pinning. In *Network and Distributed System Security Symposium*. Internet Society, Feb. 2015.

[18] W. Mayer, A. Zauner, M. Schmiedecker, and M. Huber. No Need for Black Chambers: Testing TLS in the E-mail Ecosystem at Large. *arXiv preprint arXiv:1510.08646*, 2015.

[19] J. Prins and B. U. Cybercrime. Diginotar certificate authority breach'operation black tulip', 2011.

[20] A. Ronacher. Flask - web development, one drop at a time. http://flask.pocoo.org/.

[21] Y. Sheffel, R. Holz, and P. Saint-Andre. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS(DTLS). RFC 7457 (Proposed Standard), 2015.

[22] Y. Sheffer, R. Holz, and P. Saint-Andre. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7525 (Proposed Standard), 2015.