Characterizing the Adoption of Security.txt Files
And their Applications to Vulnerability Notification

William P. Findlay
School of Computer Science
Carleton University
Ottawa, Canada
will@ccsl.carleton.ca

AbdelRahman Abdou
School of Computer Science
Carleton University
Ottawa, Canada
abdou@scs.carleton.ca

***Abstract*—While security researchers are adept at discovering vulnerabilities and measuring their impact, disclosing vulnerabilities to affected stakeholders has traditionally been difficult. Beyond public notices such as CVEs, there have traditionally been few appropriate channels through which to directly communicate the nature and scope of a vulnerability to those directly impacted by it. Security.txt is a relatively new proposed standard that hopes to change this by defining a canonical file format and URI through which organizations can provide contact information for vulnerability disclosure. However, despite its favourable characteristics, limited studies have systematically analyzed how effective Security.txt might be for a widespread vulnerability notification campaign. In this paper, we present a large-scale study of Security.txt's adoption over the top 1M popular domains according to the Tranco list. We measure specific features of Security.txt files such as contact information, preferred language, and RFC version compliance. We then analyze these results to better understand how suitable the current Security.txt standard is for facilitating a large-scale vulnerability notification campaign, and make recommendations for improving future version of the standard.**

I. Introduction

Encouraging affected stakeholders to patch security vulnerabilities has traditionally been a difficult task [24, 25, 26, 28]. Among the chief sources of friction for widespread vulnerability notification, security researchers have cited the problem of determining the best point of contact for impacted stakeholders [24, 25]. To drive positive security outcomes from vulnerability research, the security community needs a way to effectively communicate the presence and severity of vulnerabilities to affected stakeholders. The proposed Security.txt standard [16, 18] seeks to address this limitation by providing the community with an open standard for publishing vulnerability contact information. The file format is designed to be both human-readable and machine-parsable, enabling vulnerability notification on both a manual and automated basis.

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2022
28 April 2022, San Diego, CA, USA
ISBN 1-891562-78-9
https://dx.doi.org/10.14722/madweb.2022.23014
www.ndss-symposium.org

Security.txt is still a relatively young standard in its early draft stages. In particular, existing RFCs are currently being invalidated every six months, often with non-trivial changes [16]. To maximize utility for vulnerability notifications, standard authors great care must be taken to ensure that the evolving standard best meets the needs of various stakeholders, including the domains implementing the standard and the security researchers who will come to rely upon it. The evolving nature of this situation presents an opportunity to study Security.txt's early adoption, characterize its benefits for vulnerability notifications, and identify any sources of friction between the upstream standard and downstream stakeholders. Findings therein can then be used to drive improvements in the existing standard and promote best practices among early adopters.

In this paper, we present a large-scale measurement study characterizing the adoption of the Security.txt standard in the wild, emphasizing its utility for vulnerability notification campaigns. In particular, we develop a Security.txt crawler capable of rapidly iterating over the top 1M most popular domains and identifying, classifying, and storing information about any Security.txt files it encounters for subsequent analysis. Finally, we leverage this crawler and the data it produces to answer research questions about the effectiveness of Security.txt files for vulnerability notifications.

Our work is most similar to a study conducted concurrently by Poteat and Li, who examined the adoption of Security.txt in the Alexa top 100K sites over 15 months. While many of the findings presented by Poteat and Li are consistent with our own, we note significant differences in methodology, including scope and source of truth for seed domains, the timeline during which measurements were conducted, and the approach to parsing and validating Security.txt files. Moreover, our work considers additional research questions which were left out of Poteat and Li's study, including the differences between RFC versions deployed in the wild and how Security.txt contacts compare with other potential vulnerability contacts such as WHOIS abuse records. We argue that considering these two works together helps to build a clearer picture of the overall Security.txt landscape and lends further insight into the reproducibility of our respective results.

Specifically, we approach the following research questions:

1) **Measuring Adoption.** To what extent is the Security.txt standard currently adopted by major stakeholders?
2) **Measuring Consistency.** How consistent are existing Security.txt deployments? Which RFC versions are they compliant with, if any? What factors impact consistency?
3) **Measuring Utility.** How viable is the current Security.txt standard for a widespread, automated vulnerability notification campaign?
4) **Identifying Barriers.** What barriers, if any, exist that are inhibiting the adoption of the standard or decreasing its utility for vulnerability notifications? How can the current standard be improved to mitigate or eliminate these barriers?

Our findings indicate that Security.txt currently sees a meagre adoption rate of about 0.49% among the top 1M domains. Moreover, the vast majority of domains that do implement Security.txt files are following outdated versions of the RFC, some of which are over three years old. Many of these files incorporate fields that are no longer considered valid, were never considered valid, or are common misspellings of standardized fields. These common issues increase the risk that any automated software driving a notification campaign might either ignore the Security.txt file entirely or misinterpret it in such a way that its content is of little utility to security researchers.

Among valid Security.txt files, we find that a significant number provide contact information in the form of email addresses or URLs linking to a few large-scale bug bounty platforms. These contact methods are highly amenable to automated or semi-automated vulnerability notification campaigns, a promising early result despite the challenges highlighted above. However, very few of these files provide additional information which may be critical to the success of a notification campaign, such as preferred communication language, expiry date, a digital signature, and encryption keys.

Based on these findings, we recommend that several currently optional fields be mandatory in Security.txt. We also propose measures to mitigate potential sources of error, such as introducing regional spellings as alternative field names, making field formats more consistent with each other, and taking additional steps to ensure machine readability. We also encourage downstream implementers to read the latest versions of the standard carefully, employ (and abide by) shorter expiry dates, and keep up to date with the latest RFC versions. Finally, implementers should work to accommodate any automated software consuming their Security.txt files and follow security best practices such as providing a cleartext PGP signature to ensure authenticity.

In summary, this paper makes the following contributions:

1) We design and open source a **novel web crawler and parser for Security.txt files** that can accurately parse and classify all major, breaking RFC versions of the Security.txt standard. Leveraging the power and safety of Rust, our crawler can iterate over the top 1M domains (2M requests) in just under 2 hours on commodity hardware.
2) We present a **large-scale study of Security.txt's adoption** among the most popular domains, according to the Tranco list [23]. This study reveals several key factors inhibiting the practical adoption of Security.txt in its current form and offers early insights into how practical Security.txt files might be for a widespread, automated vulnerability notification campaign.
3) We leverage the findings from our study to make **recommendations to both Security.txt's authors and downstream stakeholders**. Our goal is to reduce current sources of friction between the Security.txt specification and its implementations.

II. Background

Security.txt [16, 18] is a proposed open standard for publishing vulnerability contact information. In particular, its goal is to be a machine-parsable and human-readable file format through which an organization can (1) list contact information for disclosing security vulnerabilities; (2) link to any relevant security policy; (3) thank security researchers and list previously found vulnerabilities; and (4) provide public encryption keys which can be used to digitally sign and encrypt communications with security researchers. The end goal is to promote and facilitate responsible vulnerability disclosure in the security community [18].

The standard specifies two possible locations for Security.txt files: `/.well-known/security.txt` or `/security.txt`. These files must be served over HTTPS and must follow a pre-determined, plaintext format. All lines must either be a comment, which must be delineated using the `#` symbol, or a field-value pair delineated by the field name followed by a colon. Listing 1 in the Appendix depicts an example Security.txt file compliant with RFC version 12 (the most recent version at the time of writing this paper). For illustrative purposes, the example file uses every possible field defined in the current standard, although organizations frequently use only a few. We briefly highlight the main components of the Security.txt file in the paragraphs that follow, but encourage the reader to consult the official RFC for a complete description of the full specification [16].

The `Contact` field is required and allows an organization to list a contact address in one of three acceptable formats: `https` for web URIs, `mailto` for email addresses, and `tel` for telephone numbers. The implementer may choose any of these three options and may list multiple contact fields in order of preference. A given contact address must begin with the name of the chosen format, followed by a colon.

Besides the `Contact` field, the only other required field as of RFC version 12 is the `Expires` field, which lists an ISO8601-compliant date by which the contents of the file should no longer be considered valid. The `Expires` field is one of only two fields that may not appear more than once (the other being `Preferred-Languages`).

The `Preferred-Languages` field is an optional field that allows the organization to enumerate a list of preferred contact languages for communication with security researchers. Unlike other fields, which allow multiple values to be defined simply by repeating the field, the `Preferred-Languages` field must be a comma-separated list of languages. Like the `Expires` field, `Preferred-Languages` may only be provided once in a given Security.txt file. Languages are specified using their respective IANA language tags as per RFC 5646 [32].

The `Canonical` field allows an organization to list the canonical URI where the Security.txt file can be found. When combined with a valid digital signature, this field gives the security researcher assurance that the file and its content is authentic. The `Encryption` field provides a URI where security researchers may locate the organization's public encryption key. If an organization chooses to digitally sign its Security.txt file, it may do so using a cleartext PGP signature. In this case, a link to the public encryption key used to generate the PGP signature should be provided via the `Encryption` field. An example of this is given in Listing 1.

Aside from the aforementioned fields, additional fields include `Acknowledgments`, `Policy`, and `Hiring`. These fields are used to acknowledge researchers who previously disclosed vulnerabilities, outline the organization's security policy, and perform hiring outreach to the security community.

TABLE I: Breaking RFC versions of Security.txt and their distinguishing features.

RFC	Distinguishing Feature
12 [16]	Uses ISO8601 dates for the `Expires` field.
11 [17]	Uses RFC5322 dates for the `Expires` field
09 [15]	Last version where the `Expires` field is optional.
05 [14]	Last version where `http` URIs are allowed.
04 [13]	Last version to use the `Signature` field instead of PGP cleartext.
02 [11]	Last version with inferred URI schemes for email and telephone contacts.

Since its inception in late 2017 [12], Security.txt has undergone 13 distinct RFC versions (numbered 00–12) [16]. Over time, significant changes have been made in terms of its scope, the number of fields it defines, and the nature of these fields. Working backwards from the latest version, we identify a total of six "breaking" versions, i.e., ones that are incompatible with a higher RFC version. For example, RFC 11 is incompatible with RFC 12, since RFC 11 uses RFC5322 dates, whereas RFC 12 uses ISO8601 dates. Table I highlights each breaking RFC version and their respective distinguishing features. We later use these distinguishing features to classify Security.txt files by their highest possible RFC version.

III. Methodology

A. SecMap and sectxt.rs

To scrape and parse Security.txt files, we design and implement two artifacts[1]: `sectxt.rs`, a Security.txt parsing library for Rust, and SecMap, a highly-concurrent web crawler that uses `sectxt.rs` to parse and validate candidate Security.txt files. Figure 1 depicts an overview of our methodology and SecMap's architecture. SecMap and `sectxt.rs` are written in just under 2K lines of code in the Rust programming language.

SecMap is inspired by and, in part, named after Zmap [7] an open source tool for Internet-wide scanning. Like ZMap, SecMap shares a similar goal of being able to rapidly iterate over candidate hosts, which we accomplish by massively parallelizing its scanning strategy. However, the similarities between the two tools stop here. ZMap is designed to iterate over the entire IP space and may probe a single domain many times in a single run. Conversely, SecMap only issues two requests per input domain in a given scanning run. This difference has ethical implications since it means SecMap will never overwhelm the domains it scans.

A few factors motivated the creation of custom software for this research, the most critical of which is the unique requirements imposed by the nature of this study. We want to investigate the adoption of Security.txt files across many domains and over a potentially wide variety of different RFC versions. This scale necessitates a crawler that is both fast and capable of handling many different versions of the Security.txt standard at the same time. While some Security.txt parsing libraries do exist,[2] they are neither fast enough for this study nor are they capable of handling more than one RFC version. Moreover, none of the existing libraries support the latest RFC, version 12, which means that we would potentially miss out on valuable early adoption metrics for the most recent components of the standard.

Rust is an ideal choice for implementing our custom tooling for several reasons. First, the `tokio` asynchronous programming framework provides powerful abstractions for writing correct, maximally concurrent code to iterate over the top 1M domains as fast as possible. Due to this highly concurrent implementation, SecMap can rapidly take snapshots of the Security.txt landscape as it evolves. Second, Rust's powerful type system is highly conducive to supporting multiple Security.txt versions simultaneously, which was a primary goal of this research. Using Rust's enum variants, we can express each valid Security.txt version as a single data structure and define custom parsing logic for each version as needed.

We leverage the MongoDB NoSQL database to store Security.txt data. Using Rust's MongoDB driver and the `serde` library, inserting records into the database is as simple as annotating our Security.txt data structure with `#[derive(Serialize)]` and invoking the appropriate

[1] Available: https://github.com/willfindlay/secmap

[2] An official list of Security.txt related projects is available at https://securitytxt.org/projects.

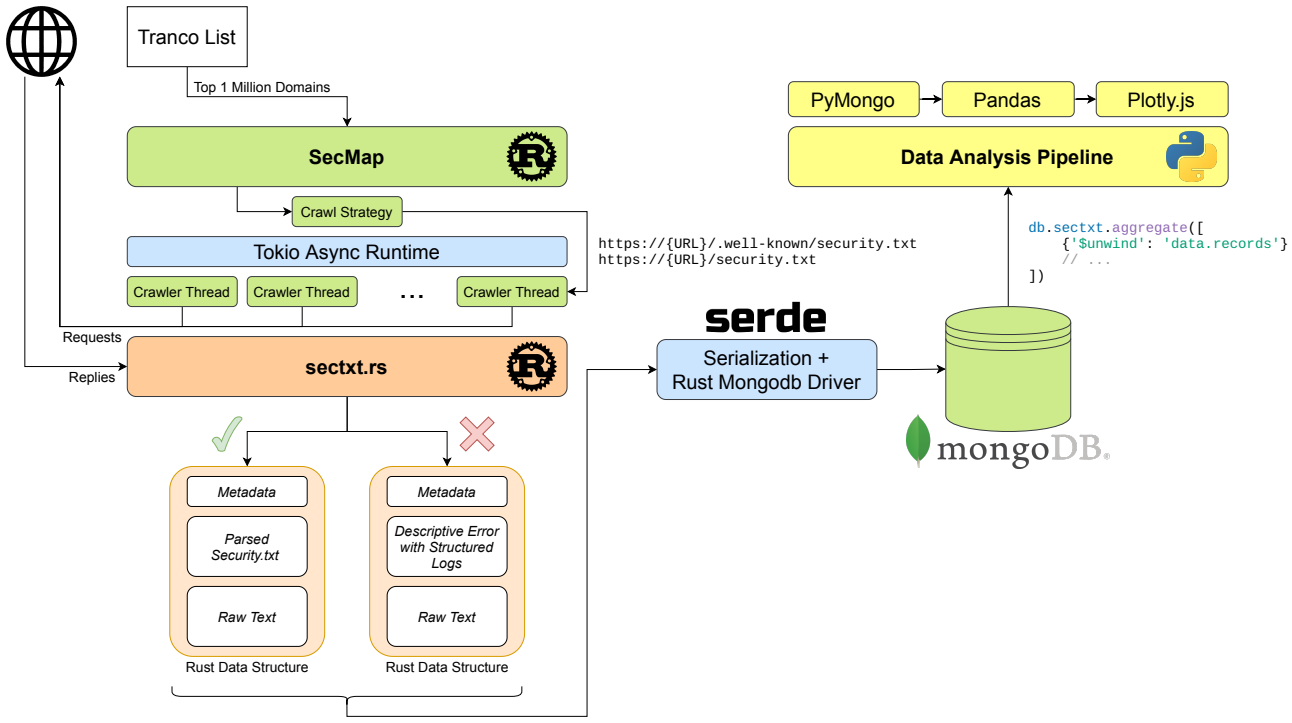

Fig. 1: An overview of our methodology. SecMap ingests the top 1M domains from the Tranco list and mutates them into relevant URLs using a crawl strategy. These URLs are then passed to several crawler threads which concurrently issue HTTP GET requests. Valid replies are forwarded to the `sectxt.rs` library which parses and validates Security.txt files, then stores them in a database for subsequent analysis.

method call at runtime. This approach saves significant engineering effort and is largely responsible for the small number of lines of code needed to implement SecMap. An added benefit to this approach is that a smaller codebase makes the source code easier to test and audit and reduces the probability of a mistake in the tooling impacting results.

Although a smaller codebase helps to reduce the risk of developer error, we elect to store the raw replies associated with each request. This is a precautionary measure, enabling subsequent human analysis to guard against potential errors in data collection and improving our confidence in the accuracy of SecMap's results. SecMap also stores as much metadata about each request and reply as possible, including a unique identifier for the experimental run, the time of day at which the request was sent, the domain and full URL being checked, and any errors encountered while fetching or parsing the Security.txt file. Finally, we maintain a verbose debug log of each action SecMap performs over a given run, storing as much information as possible in case further investigation is needed.

To encourage reproduction of our results and provide maximum utility to the research community, we are releasing both SecMap and `sectxt.rs` as free and open-source software under the MIT license. Beyond their utility for measuring Security.txt's adoption, we expect that SecMap and `sectxt.rs` will also be valuable tools for driving future vulnerability notification campaigns (see Section V). To support reproducibility, we have also open sourced our raw dataset and data analysis toolchain[3].

B. Experimental Configuration

1) Hardware and Network Connection: We run our experiments on commodity hardware running version 5.15.7-arch1-1 of the Linux kernel. The system has an eight core 4.5GHz Intel i7-7700K Kaby Lake CPU and 16GiB of DDR4 RAM at 3GHz. Finally, the NIC is an Intel I219-V ethernet controller running the latest e1000e driver. We issue requests from a local Internet connection provided by Rogers communications with advertised speeds of 1Gbps download and 20Mbps upload.

2) Tuning SecMap: In addition to binding to a network interface, SecMap supports several configuration options that control its speed and impact its accuracy. In particular, we support command-line flags to tune the number of concurrent requests SecMap buffers at any given time and the timeout duration for filtering out dead connections. Getting these settings right is imperative to the success of our study. A lower level of concurrency and high connection timeout will improve accuracy by ensuring that all connections are handled from start to finish. However, tuning these values too low would result in prohibitively slow crawl times. Similarly, higher concurrency and lower timeout duration will significantly improve crawl speeds but could impact the accuracy of results by dropping some valid connections before they can complete.

We determine the optimal SecMap configuration empirically by repeating preliminary trials under different settings and

[3]Available: https://gitlab.com/willfindlay/secmap-dataset

measuring both the time it takes to complete each trial and the number of successful attempts to locate a Security.txt file. Specifically, we start at a low concurrency level and high timeout and gradually increase concurrency and lower the timeout until we note diminishing returns in speed versus the number of found Security.txt files. Repeating this process iteratively, we arrive at a concurrency level[4] of 512 and a connection timeout of 10 seconds.

Another factor to consider is whether SecMap will go too fast for the operating system to handle. In our early experiments, we noted a large number of requests being dropped due to too many open file descriptors after running SecMap under `strace`. The Linux kernel imposes a limit on the number of open file descriptors to prevent a single process from exhausting system resources — open network sockets are one such resource. To alleviate this difficulty, we implement a `--ludakris` command-line flag, which serves two purposes. First, it bumps its file descriptor resource limit to the theoretical maximum using the `rlimit(2)` system call. Second, it increases the maximum concurrent TCP connections supported by the kernel using a `sysctl(2)` call. We run all of our SecMap experiments under this "ludakris"[5] mode and note no further bottlenecking by the kernel.

3) Seeding and Crawling Domains: We seed candidate domains using the Tranco list [23], an aggregation of the Alexa [1], Majestic [21], and Umbrella [20] lists. Among its most desirable properties, Tranco is hardened against attacker manipulation, mitigating potential sources of error and making it an attractive target for this kind of research. Since its initial publication in 2019, the Tranco list has been consistently used by other measurement studies published in top security venues [10, 27, 30].

We consider the top 1M domains on the Tranco list for our study. For reproducibility, we note that we used version 5WYN[6] of the Tranco list from October–November 2021. To determine candidate URLs, SecMap ingests the list of 1M domains and produces a new list of 2M candidate URLs — one for each canonical path outlined in the Security.txt RFC [16]. It then uses the `reqwest` Rust library to issue an HTTP GET request to each URL in parallel, checking for the presence of a Security.txt file.

To improve confidence in the results of the study, we repeat our crawling activity once per day over a period ranging from November 28^{th} to December 5^{th} 2021, and verify no significant difference in the collected data across any of the experimental runs. We include only the most recent data from December 5^{th} in our results.

4) Collecting Security.txt Files: After issuing a GET request for a Security.txt file, SecMap applies four filters to discard invalid replies. We enumerate each filter as follows:

F1 **Request and Protocol Errors.** Request and protocol errors comprise any error that prevents an HTTP request from completing. In general, this includes any error encountered by the `reqwest` library (the client used to make the GET requests). Examples of request errors include failed DNS queries, unreachable hosts, and connection timeouts. Examples of protocol errors include invalid SSL certificates or HTTP responses.

F2 **Bad Status Codes.** Of the successful requests, we filter the resulting HTTP responses by status code, discarding any response without a status code of "success" (200–299) as per HTTP RFC 7231 [9]. This step filters out known-bad responses and spurious replies returned by misbehaving servers.

F3 **Non-Plaintext Responses.** Of the successful HTTP responses, we discard any responses which are not tagged with the `text/plain` MIME type. This is done in accordance with the Security.txt standard, which stipulates that all Security.txt files must be returned as plaintext responses with the appropriate MIME type set in the response header [16]. Filtering out such spurious replies helps to improve crawling speeds and reduces the volume of data in the database.

F4 **Parse Errors.** All responses which have passed the above checks are passed into the Security.txt parser implemented in `sectxt.rs`. The parser then returns any errors encountered while parsing a Security.txt file, and flags the corresponding file as invalid.

IV. Results

A. Adoption Rate and Taxonomy of Security.txt Files

Of the 2M requests, SecMap received 1.38M valid HTTP responses. HTTP 400-series status codes (client errors) accounted for 83% of all responses, with Error 404 (Not Found) at 78% (1.08M) and Error 403 at 3% (46K) of all responses. Successful status codes (200-series) comprised 15.6% of all responses (214K), with status code 200 being the most common. Interestingly, 0.3% of replies contained non-standard status codes (considered invalid by SecMap). These were generally vendor-specific error codes for the given web server or invalid status codes returned by malfunctioning web servers. Unless otherwise specified, the remaining analysis in this section considers only the set of *valid* Security.txt files as determined by SecMap.

Figure 8 in the Appendix classifies the responses filtered out by F1, F3, and F4. In total, 237K requests (11.9%) result in a protocol error, with TLS errors being the most common at 11.2%. Another 383K requests (19.2%) result in a request error, with DNS errors, connection timeouts, and refused connections being the most common. We received 206K replies (96% of all "successful" responses and 10% of all requests) that are not plaintext. The reasons underlying this large number of spurious replies are similar to the problems with HTTP status codes outlined above. Many web servers are configured to serve incorrect responses (most commonly an HTML page accompanied by a 200-series status code, but we also note a few instances of binary blobs) to invalid requests.

[4]This means that SecMap will buffer up to 512 requests at a time.

[5]The misspelling of "ludicrous" is an intentional reference to the name of the largest possible map size in the video game Age of Empires II.

[6]Available at https://tranco-list.eu/list/5WYN.

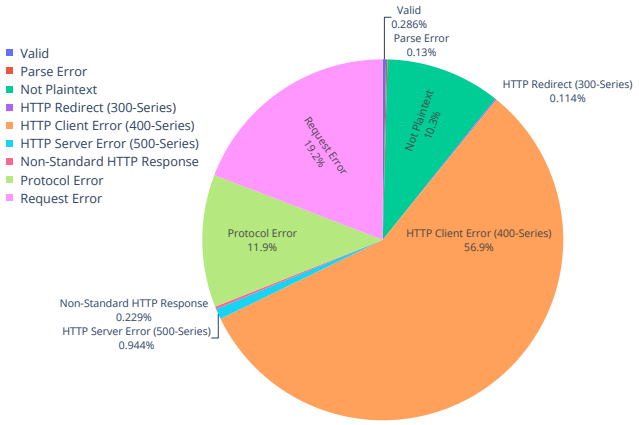

Fig. 2: A summary of all responses to the 2M requests. Protocol errors and request errors prevented a request from successfully completing. Valid files, parse errors, and non-plaintext responses account for all 200-series HTTP status codes. Note that HTTP client errors make up nearly 57% of all responses.

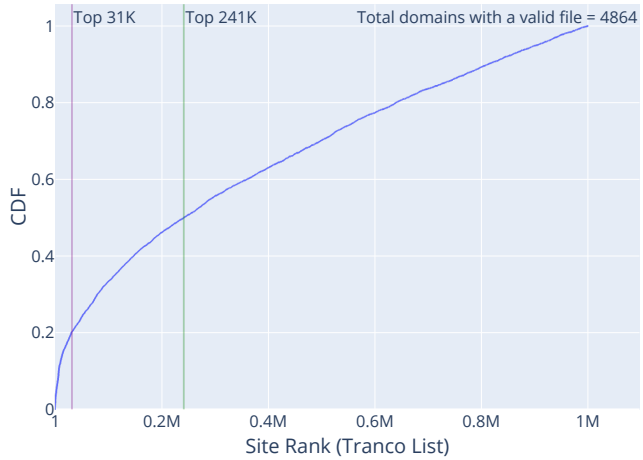

Fig. 3: The cumulative percentage of valid Security.txt files by site rank according to the Tranco list [23]. Note that the top 31K domains account for 20% of all valid files and the top 241K account for 50% of all valid files. In total, only 4864 of the top 1M domains provide a valid Security.txt file.

Of the 8.3K plaintext replies, 2.5K result in parse errors for a total of 5.8K valid Security.txt files (or 0.29% of all requests). After accounting for domains with two files (recall that there are two valid canonical paths for a Security.txt file), we are left with 4864 domains with at least one valid Security.txt file (0.49% of the top 1M domains). If we narrow our scope to only the top 100K domains, we note a total of 1624 unique domains (1.6% of the top 100K). These numbers might seem shockingly low but are, in fact, in line with our expectations outlined at the beginning of this section.

The most common parse error was including a line without any field (i.e. any line that is not a comment and that does not begin with a field name followed by a colon). Many of these can be attributed to plaintext responses that did not contain anything resembling a Security.txt file. Upon manual inspection of a random sample of invalid files, we also note another common occurrence: the presence of lines that should have been comments but were missing a #. The most prominent example of this was amazon.com/security.txt (number 18 on the Tranco list), which invalidated itself according to our parser by listing a bare URL without any comment or field name. The second more frequent parse error was a missing contact field (considered invalid for all RFC versions). In total, 900 files did not provide any contact field, effectively making the Security.txt file unusable.

Figure 3 shows the cumulative percentage of valid Security.txt files over the top 1M domains (without double-counting). Interestingly, while the distribution of Security.txt files is more-or-less uniform over the long tail of domains, the highest-ranked domains are significantly more likely to have a Security.txt file. In particular, the top 31K domains account for 20% of all valid Security.txt files, while the top 241K account for 50% of all valid Security.txt files. These numbers make intuitive sense given the context underlying site ranks — the top-ranked domains generally correspond to large organizations with a vested interest in security (to protect their clientele as well as their profit margins). Such organizations are, in turn, more likely to be following the bleeding edge of security best practices.

Recall from Section III that we are interested in measuring the RFC version of Security.txt files in addition to their validity. Accordingly, the number of valid files is the total of valid files across all RFC versions. This turned out to be a sound research decision since the vast majority of Security.txt files do not fall within even the three most recent RFC versions. Only 560 files (11.5%) are compliant with the most recent RFC (version 12 [16]). RFC 11 accounts for another 272 files (5.6%). RFC 9 was by far the most common version with 2283 files (46.9%). Shockingly, RFC 5 (which dates back to 2019 [14]) accounts for 1598 Security.txt files (32.9%). Even older RFCs account for the remaining 151 files.

B. Expiry Dates

To characterize expiry windows, we measure the expiry dates of Security.txt files against various factors such as site rank and RFC version. Note that RFC version 9 is not included in the data presented here. This omission is intentional. RFC 9 is the highest RFC version in which the `Expires` field is optional (and this is its only distinguishing characteristic). Since our parser uses the absence of an `Expires` field to test for RFC 9 files, including them in this data is an impossibility.

We find that the majority of Security.txt expiry dates (85.5%) are concentrated within a period of 1–4 years in the future. Other bands of activity occur in and around years with round numbers such as 2030. Of the files that expire in 2021, about half had already expired at the time of measurement. A small number of Security.txt files set expiry dates long enough that they might realistically outlive the person who wrote them. Example years include 2099 and 2222.

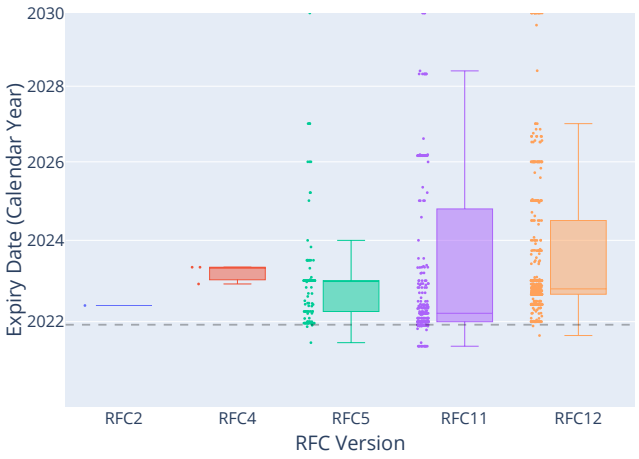

Fig. 4: The distribution of Security.txt expiry dates by RFC version. RFC 9 is not present here since its only distinguishing feature from RFC 11 is an optional `Expires` field. The y-axis is cut off at 2030 for readability. Any point outside of 1.5 IQRs from Q1 or Q3 is considered an outlier. The dashed grey line corresponds to today's date (as of writing this paper). Note that several RFC 11 files are currently expired, as well as some RFC 5 and RFC 12 files. RFC 11 and RFC 12 are also more skewed into the future compared with other RFC versions.

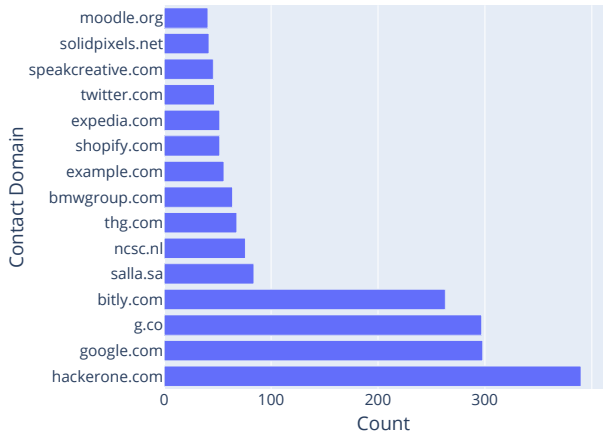

Fig. 5: Top 15 contact domains for `http(s)` and `mailto` contacts in Security.txt files.

Figure 4 depicts the distribution of expiry dates by RFC version. Expired files and those on the precipice of expiring are distributed among RFC versions 5, 11, and 12, although most belong to version 11, with two particularly large clusters concentrated around May and November 2021. Upon manual inspection of the files in these clusters shared common vulnerability contact domains, including 14 belonging to tech-day.com and 34 belonging to moodle.com. TechDay appears to be an aggregation of news sites from New Zealand, whereas Moodle is a popular learning management system platform. The former cluster is a collection of Security.txt files belonging to a single organization, while the latter appears to be a sensible default provided by Moodle. Curiously, none of the RFC 2 or RFC 4 files were expired. While this result may seem counter-intuitive at first, it can be explained by the fact that previously expired files may be more likely to have been updated to a later RFC, along with the minimal sample size for RFC 2 and RFC 4. RFC 12, RFC 11, and RFC 5 each had Security.txt files with expiry dates well into the future. In the case of RFC 11 and RFC 12, a significant number extended into the later 2020s and early 2030s — enough to skew both distributions into the future.

C. PGP Signatures

We find a total of 363 domains with a Security.txt file signed using PGP cleartext. Of these, only 310 are valid Security.txt files. Of the 53 invalid Security.txt files with digital signatures, 29 arose from incorrect handling of the OpenPGP specification or encryption keys. Nine Security.txt files contained a formatting error that invalidated their cleartext signature and caused parsing to fail. Another 20 files used the `Encryption` field incorrectly, placing a key ID tagged with the `openpgp4fpr` URI scheme. This URI is invalid according to the Security.txt spec, and so the file was invalidated by SecMap.

Among the 310 valid Security.txt files that include PGP signatures, we find that they all list at least one public encryption key. In total, we find 153 unique encryption keys, with a select few keys being far more common than others. A key belonging to soc.beiersdorf.com, the security division of a German beauty product corporation, accounted for 28 `Encryption` fields in total (9% of all `Encryption` fields). The second most common key belonged to Atlassian, a web security firm. Manual inspection of files containing these keys revealed that they all belong to domains owned by their respective corporations. In the case of Atlassian, this included unique domains for specific security products as well as status pages managed by Atlassian for popular services like Discord.

D. Vulnerability Contacts

A primary goal of this research was to gauge the viability of the Security.txt standard for automated vulnerability notification campaigns. As one can imagine, the success or failure of such a campaign would heavily depend on the reliable availability of contact information in a *consistent* format. The most desirable format would arguably be `mailto` links since these can be reliably contacted by automated software. Less desirable are `http` and `https` links since these are generally inaccessible to automated software without significant engineering effort (e.g., natural language processing, HTML parsing, web drivers to fill out forms). In this section, we examine the most common contact schemes and domains used by Security.txt files and compare them with a previously studied alternative: WHOIS abuse records [24].

`https` (26.5%) and `mailto` (72.8%) were the most popular contact schemes, with `tel` (0.6%) and `http` (0.1%) being the least popular. Note that `http` contacts are considered deprecated by the Security.txt standard and have not been supported since RFC 4 [13]. In total, 4824 `mailto` contacts were defined across 4100 Security.txt files, and 1560 `https` contacts were defined across 1440 Security.txt files. This

means that 84.3% of valid files support at least one email contact, while 29.6% of files support at least one HTTPS contact. Only 35 files provide a `tel` contact and four files provide an `http` contact.

Many contact domains exhibited disproportionate popularity. Figure 5 shows the respective frequencies of the top 15 most popular contact domains among crawled sites. HackerOne [19] which makes up 6.7% of all contact domains, is a bug bounty platform that partners with businesses to manage their coordinated disclosure policies. The second and third most popular domains, g.co and google.com are both used as umbrella contacts for all official domains owned and operated by Google. The fourth most popular contact domain bitly.com is a popular URL shortener, which may be used to obfuscate or otherwise shorten the actual contact URL. The salla.sa and shopify.com domains belong to e-commerce platforms, and the corresponding Security.txt files appear to be sensible defaults applied to websites operating atop these platforms. Other common umbrella domains were thg.com, used in several cosmetic product websites and bmwgroup.com, used in websites belonging to auto manufacturers operating under BMW.

Perhaps the most shocking finding among the most popular contact domains is the prevalence of example.com as a vulnerability contact. In total, we find 56 separate instances of mailto:security@example.com being used as a `mailto` contact. This is the example email address provided on the Security.txt informational site [18] and should not be treated as a legitimate contact. The end result is 56 Security.txt files (1.2% of all valid files) that essentially have no contact information, rendering them totally useless for security researchers hoping to perform vulnerability disclosure.

Security.txt Contacts Versus WHOIS Abuse Contacts: Li *et al.* [24, 25] have investigated the use of WHOIS abuse records as vulnerability contact information in longitudinal vulnerability notification campaigns. Their results were modest, indicating that WHOIS contacts may not be the best point of contact for vulnerability disclosure. In the context of this research, we are interested in verifying this hypothesis by checking to see whether WHOIS abuse records match the contact information given in Security.txt files. To do this, we instruct SecMap to fetch the WHOIS abuse contact information associated with every domain that also has a valid Security.txt file. In total, SecMap was able to retrieve 2477 WHOIS abuse contacts. Interestingly, not a single WHOIS abuse contact was found to match a Security.txt contact. Moreover, we found only three instances of an abuse contact whose domain matched the site's domain. This finding raises further questions about the utility of such schemes for vulnerability notification.

E. Contact Languages

Another critical piece of information for security researchers seeking to perform vulnerability notifications is knowing which language to use when communicating with vulnerability contacts [25]. The Security.txt standard provides such

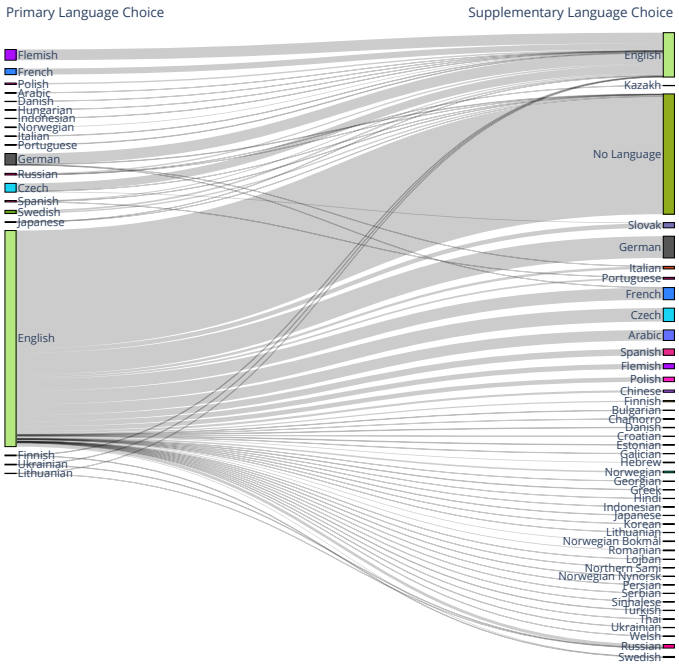

Fig. 6: Primary and supplementary language choices for vulnerability contacts. Only Security.txt files with at least one preferred language are considered. We define the primary language of a Security.txt file as the first language provided in its `Preferred-Languages` field. Any other languages listed as `Preferred-Languages` are considered supplementary. A value of "No Language" on the right-hand side indicates a file with no supplementary languages. Note that English dominates both the primary and supplementary language choices in our dataset.

information using the `Preferred-Languages` field, an optional, comma-separated list of IANA language tags per RFC 5646 [32]. In this section, we examine the use of the `Preferred-Languages` field in Security.txt files, with a particular emphasis on how many files provide a preferred language and how frequently specific languages are chosen.

To identify a given language tag, SecMap uses the `language-tags`[7] Rust library, which provides a parser and validator for IANA language tags. We then use metadata from the IANA registry to convert each language tag into its named counterpart (e.g. `en-US` would resolve to "English"), deduplicating as necessary. Almost half of all Security.txt files (47%) did not provide any contact language, while English (32%) was the most popular contact language provided. Other popular languages included German (4.5%), Czech (3.0%), French (2.5%), Flemish (2.2%), Arabic (1.5%), and Spanish (1.1%). All other languages appeared in under 1% of files.

According to the Security.txt standard, stakeholders may provide multiple languages in the `Preferred-Languages` field, in order of preference. Figure 6 depicts the relationship between primary and secondary languages in Security.txt files. We define a "primary language choice" as any language that is the *first* element of a file's `Preferred-Languages` list.

[7]Available: https://crates.io/crates/language-tags

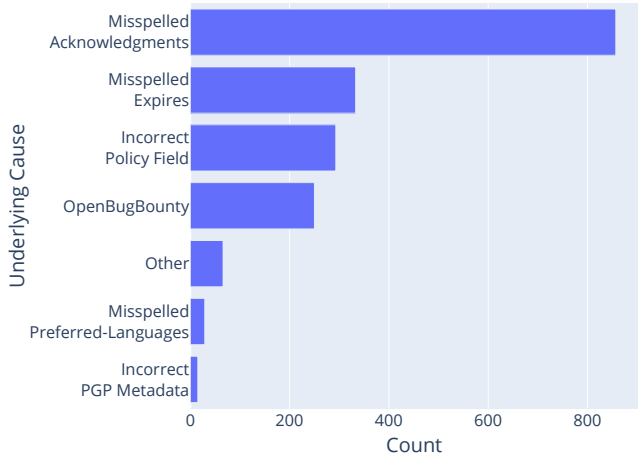

Fig. 7: The most common causes of extension fields in Security.txt files. These were found using a simple string-matching heuristic and manual examination of the data. The "Other" category comprises all extension fields that did not map cleanly onto this heuristic (e.g. unique extension fields).

A "secondary language choice" is any language present in the list that is *not* the first element (i.e. all other elements in the list). English is the most popular choice as both a primary and a secondary language. In total, only 21 Security.txt files (0.4%) did not list English as a primary or secondary contact language.

F. Extension Fields

To provide as much forward compatibility as possible [16], the Security.txt standard provisions for *extension fields*, which it defines as any field name not included in the official standard. For instance, a line like `Foo: Bar` would be considered an extension field. Characterizing the most common extension fields used in Security.txt files is important for a few reasons. First, it may highlight widely-used extensions that should be considered for inclusion in the official standard. Moreover, extension fields may be considered as hard errors by an ignorant parser[8] — SecMap initially considered extension fields to be hard errors before it was modified to support them. A subtler yet equally important factor is that extension fields may mask any unintentional errors (e.g., spelling mistakes) that invalidate an official field. While the Security.txt file would still be considered valid, this information would likely go unnoticed by automated software.

In total, there were 1843 extension fields in 1362 files. This number was small enough that we could manually apply a heuristic to characterize them by reason. We derive the heuristic iteratively by examining a sum aggregation of all extension field names and applying a simple string matching algorithm to them. In the end, we derive a list of the six common reasons for extension fields (shown in Figure 7). By far, the most common error was a misspelled `Acknowledgments` field (e.g. using the British English variant with an extra "e"). In total, this error accounted for 857 extension fields (46.5%). The second most common error was a misspelled `Expires` field (most commonly spelled `Expiration`), which accounted for 333 extension fields (18%). The third most common reason for extension fields was what we describe as an "incorrect policy field". These are fields that ordinarily should have been assigned to the official `Policy` field but were named something else in the Security.txt file. In total, incorrect policy fields accounted for 293 extension fields (15.9%). Other errors were less common but still occurred multiple times. A total of 29 extension fields (1.6%) were caused by a misspelled `Preferred-Languages` field. Another 15 (0.8%) were caused by invalid PGP metadata not picked up by SecMap's cleartext PGP parser.

`OpenBugBounty` fields were the fourth most common (and arguably the only legitimate) extension field encountered by SecMap. OpenBugBounty [31] is an open platform for collaboration between security researchers and stakeholders impacted by vulnerabilities. The Security.txt files with `OpenBugBounty` fields each contained a link to the organization's OpenBugBounty program. Due to its prevalence, this field could be a reasonable candidate for inclusion in a future version of the Security.txt standard.

V. Discussion

A. The Viability of Security.txt for Vulnerability Notifications

The primary factor limiting the utility of Security.txt for vulnerability notification campaigns is its relatively low adoption rate. In particular, we find that only 1.6% of the top 100K domains or 0.49% of the top 1M domains include at least one valid Security.txt file. Comparatively, in Li *et al.*'s original vulnerability notifications study, they found a total of 12K WHOIS abuse contacts in a population of 310K hosts, or a rate of about 3.8% [24]. WHOIS abuse records are designed to mitigate abuse by domain owners rather than as a destination for reporting security vulnerabilities. Although Security.txt files are admittedly more likely to provide the correct contact information for reporting a vulnerability, we still see a significantly lower adoption rate, even among the most popular domains.

Another critical factor is how conducive a given contact field is to automation. To conduct a large-scale vulnerability notification campaign, automated software needs to process the Security.txt file, extract contact information, and send the notification (for instance, via email). While the Security.txt file format is designed to be machine parsable [16], we find that many organizations either ignore or incorrectly implement the standard in such a way that could confound automated software in practice. Fortunately, the vast majority of Security.txt files discovered by SecMap (84.3% of all valid files) contain at least one `mailto` contact. This is an encouraging result since email is highly conducive to automated notification. While we note that 56 Security.txt files contained invalid example.com email addresses, this is a small minority of the total population.

[8]Extension fields are not well-documented in the official standard [16] and are absent in the official informational webpage [18].

Finally, language preferences merit consideration, as the effectiveness of a vulnerability notification campaign largely depends on its ability to convey the nature of security vulnerabilities to affected stakeholders [24]. Encouragingly, we find that only 0.4% of valid files (21 in total) did not list English as a primary contact language. This means that English is likely a viable choice for large-scale vulnerability notifications, and translation effort may not be required in most cases. Although this is an encouraging result, we note that nearly half (47%) of all Security.txt files do not list any preferred contact language.

In summary, while the Security.txt file format represents a promising step forward for vulnerability notifications, fundamental issues with the current standard, its misuse by downstream implementers, and a relatively low adoption rate are fundamental barriers to its practical utility.

B. Recommendations for Adopters

To maximize the utility of Security.txt for vulnerability notifications, we make the following recommendations to organizations seeking to adopt the standard:

1) **Read and Understand the Standard.** Organizations should read and understand the Security.txt standard before publishing a Security.txt file. When possible, guided tooling [18] should be favoured over hand-crafted files to ensure compliance with the Security.txt spec. In addition, organizations should keep up to date with the latest version of the Security.txt RFC to ensure that they are providing the most useful information possible to security researchers.
2) **Follow Security Best-Practices.** Organizations should follow recommended best practices when implementing their Security.txt file. This includes providing a `Canonical` field listing the correct URI where the Security.txt file can be found and cryptographically signing the file using a cleartext PGP signature. The file should also list at least one encryption key, including the one used to sign the file.
3) **Favour Explicitness.** Organizations should favour explicitness when possible. This includes defining all optional fields, even when the field value matches a default defined by the Security.txt standard. For instance, organizations should always provide at least one value to the `Preferred-Languages` field. Being explicit about preferences reduces the likelihood of misunderstandings on the part of security researchers, particularly when automated tooling is involved.
4) **Use Meaningful Expiry Dates.** Organizations should provide a meaningful expiry date when creating their Security.txt files. In particular, the expiry date should not be too far into the future. We recommend that organizations use an expiry date of at most six months in the short term to match pace with the lifecycle of Security.txt RFC versions. Files should be kept up to date and not left to expire.
5) **Be Aware of Automated Tooling.** Organizations should be aware of automated tooling when creating their Security.txt files. It is not feasible to expect a security researcher conducting a large-scale vulnerability notification campaign to manually inspect each Security.txt file for further information. Links to external resources should be avoided unless these resources are machine-readable (e.g. favour plaintext resources in a standard format over HTML pages or pages rendered with JavaScript). Extension fields should be avoided in favour of standardized fields where possible.

C. Recommendations for Improving the Standard

We make the following recommendations to improve the standard:

1) **Officially Adopt Popular Extension Fields.** Popular extension fields like `OpenBugBounty` should be considered for inclusion in a future version of the standard. Standardizing these fields will help to ensure that organizations follow the same blueprint rather than guessing at an appropriate field name. Moreover, automated software will be better equipped to recognize these fields when parsing Security.txt files.
2) **Make Important Fields Mandatory.** Essential fields like `Preferred-Languages` and `Canonical` should be made mandatory rather than optional. This helps to reduce potential sources of error when parsing Security.txt files and ensures that security researchers always have the information they need when attempting to notify stakeholders of a vulnerability.
3) **Remove Sources of Ambiguity.** Where possible, sources of ambiguity should be removed from the standard. Field names like `Acknowledgments` should be updated to include other popular regional spellings. The `Preferred-Languages` field should be modified to be more consistent with other fields — that is, use multiple fields rather than a single, comma-separated list. These changes will help to reduce potential friction between Security.txt files and parsers.
4) **Favour Machine Readability over Human Readability.** The Security.txt standard has the laudable goal of being simultaneously human-readable and machine-parsable [16, 18]. We argue that this may be a mistake since our experiences shows that the current RFC requires non-trivial engineering effort to parse correctly. More standardized file formats such as JSON may be more amenable to machine readability without completely sacrificing human readability. We also argue that the standard should expressly stipulate that all external resources linked within the file must be machine-readable.

VI. Related Work

The problem of vulnerability disclosure in computer security dates back to at least the mid-1980s with the advent of the earliest widespread computer exploits such as the Morris Worm [8]. Since the inception of such exploits, there has been widespread public debate [2, 3, 4, 8, 25, 26, 28, 29, 38] surrounding the best way to communicate security vulnerabilities to impacted stakeholders. One aspect of vulnerability disclosure that many can agree on is that we have an ethical responsibility as security researchers to promptly disclose

vulnerabilities to affected parties so that they may resolve the issue as soon as possible [25].

The research community has recently begun exploring the potential impact of widespread vulnerability notification campaigns. In such a campaign, researchers communicate directly with impacted stakeholders and work with them to achieve positive security outcomes (ideally total remediation). Li *et al.* [24, 25] conducted large-scale vulnerability notification campaigns using ZMap [7] to identify impacted stakeholders. They leveraged a variety of existing contact methods, including WHOIS abuse records, localized national CERTs and the US national CERT. Their findings were promising, albeit modest, with an 11% increase in remediation over the control when notifying WHOIS contacts. Among their findings, they cite a lack of availability of alternative contact methods as a primary limiting factor. Stock *et al.* [37] performed a similar study, measuring the impact of direct channels such as WHOIS abuse records and common email prefixes against indirect channels like hosting providers on vulnerability notifications. They expanded on their work in a subsequent study [36] to measure the impact of various factors such as spam filters and administrator apathy.

Poteat and Li [33] ran a longitudinal study exploring Security.txt adoption in Alexa's top 100K sites over 15 months. The authors find a comparatively larger number of domains with Security.txt files, although two differentiating factors in their methodology can explain this difference. First, their list of candidate domains changed throughout the study, whereas our results consider a fixed snapshot[9]. Second, their criteria for including files into their dataset appears to be less rigorous. We note several instances of files in their study which SecMap would have considered to be invalid. The authors also did not consider multiple Security.txt RFC versions in their analysis, instead focusing on RFC version 12. Aside from the density of valid Security.txt files and the selection criteria used to find them, many of the results from Poteat and Li [33] are consistent with our findings.

VII. Conclusion

Our results indicate that most top domains have not adopted Security.txt, and Among domains that have adopted the standard, Security.txt files are often out of date, expired, or invalid. These factors can significantly impede Security.txt's utility for vulnerability notifications, particularly when automated software may be required to scale up a notification campaign.

Based on our findings, we encourage standard authors to address several sources of confusion in the current standard, which contribute to a high rate of error among adopters. Similarly, we encourage adopters of the standard to follow security best practices, such as digitally signing their Security.txt files and providing a machine-readable link to a valid encryption key. We also stress the importance of ensuring that Security.txt files are as friendly as possible to automated software in order to maximize their utility for researchers conducting a large number of vulnerability notifications.

[9]While this is not necessarily a weakness of either study, we note that it is a differentiating factor.

References

[1] Alexa Internet, Inc., *The Top 500 Sites on the Web*, 2021. [Online]. Available: https://www.alexa.com/topsites (visited on 12/15/2021).

[2] Ashish Arora and Rahul Telang, "Economics of Software Vulnerability Disclosure," *IEEE Secur. Priv.*, vol. 3, no. 1, pp. 20–25, 2005. DOI: 10.1109/MSP.2005.12.

[3] Ashish Arora, Rahul Telang, and Hao Xu, "Optimal Policy for Software Vulnerability Disclosure," *Manag. Sci.*, vol. 54, no. 4, pp. 642–656, 2008. DOI: 10.1287/mnsc.1070.0771.

[4] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan, "Emerging Issues in Responsible Vulnerability Disclosure," in *Annual Workshop on the Economics of Information Security (WEIS)*, 2005. [Online]. Available: http://infosecon.net/workshop/pdf/cavusoglu.pdf.

[5] Louis F. DeKoven, Audrey Randall, Ariana Mirian, Gautam Akiwate, Ansel Blume, Lawrence K. Saul, Aaron Schulman, Geoffrey M. Voelker, and Stefan Savage, "Measuring Security Practices and How They Impact Security," in *Internet Measurement Conference (IMC)*, ACM, 2019, pp. 36–49. DOI: 10.1145/3355369.3355571.

[6] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicholas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson, "The Matter of Heartbleed," in *Internet Measurement Conference (IMC)*, ACM, 2014, pp. 475–488. DOI: 10.1145/2663716.2663755.

[7] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *USENIX Security Symposium*, USENIX Association, 2013, pp. 605–620. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric.

[8] Ted Eisenberg, David Gries, Juris Hartmanis, Don Holcomb, M. Stuart Lynn, and Thomas Santoro, "The Cornell Commission: On Morris and the Worm," *Commun. ACM*, vol. 32, no. 6, pp. 706–709, 1989. DOI: 10.1145/63526.63530.

[9] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," Internet Engineering Task Force, Internet Draft 7231, Jun. 2014. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7231 (visited on 12/10/2021).

[10] Pawel Foremski, Oliver Gasser, and Giovane C. M. Moura, "DNS Observatory: The Big Picture of the DNS," in *Internet Measurement Conference (IMC)*, ACM, 2019, pp. 87–100. DOI: 10.1145/3355369.3355566.

[11] Edwin Foudil and Yakov Shafranovich, "A Method for Web Security Policies," Internet Engineering Task Force, Internet Draft draft-foudil-securitytxt-02, Dec. 2017, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-foudil-securitytxt-02 (visited on 12/10/2021).

[12] Edwin Foudil and Yakov Shafranovich, "A Method for Web Security Policies," Internet Engineering Task Force, Internet Draft draft-foudil-securitytxt-00, Sep. 2017, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-foudil-securitytxt-00 (visited on 12/10/2021).

[13] Edwin Foudil and Yakov Shafranovich, "A Method for Web Security Policies," Internet Engineering Task Force, Internet Draft draft-foudil-securitytxt-04, Jul. 2018, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-foudil-securitytxt-04 (visited on 12/10/2021).

[14] Edwin Foudil and Yakov Shafranovich, "A Method for Web Security Policies," Internet Engineering Task Force, Internet Draft draft-foudil-securitytxt-05, Jan. 2019, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-foudil-securitytxt-05 (visited on 12/10/2021).

[15] Edwin Foudil and Yakov Shafranovich, "A File Format to Aid in Security Vulnerability Disclosure," Internet Engineering Task Force, Internet Draft draft-foudil-securitytxt-09, Feb. 2020, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-foudil-securitytxt-09 (visited on 12/10/2021).

[16] Edwin Foudil and Yakov Shafranovich, "A File Format to Aid in Security Vulnerability Disclosure," Internet Engineering Task Force, Internet Draft draft-foudil-securitytxt-12, May 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-foudil-securitytxt-12 (visited on 12/10/2021).

[17] Edwin Foudil and Yakov Shafranovich, "A File Format to Aid in Security Vulnerability Disclosure," Internet Engineering Task Force, Internet Draft draft-foudil-securitytxt-11, Mar. 2021, Work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-foudil-securitytxt-11 (visited on 12/10/2021).

[18] Edwin Foudil and Yakov Shafranovich, *security.txt*, Official informational website, 2021. [Online]. Available: https://securitytxt.org/ (visited on 12/10/2021).

[19] HackerOne, *HackerOne*, Official website, 2021. [Online]. Available: https://www.hackerone.com/company (visited on 12/12/2021).

[20] D Hubbard, *Cisco Umbrella 1 Million*, 2016. [Online]. Available: https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/ (visited on 12/15/2021).

[21] D Jones, *Majestic Million CSV Now Free For All, Daily*, 2012. [Online]. Available: https://blog.majestic.com/development/%20majestic-million-csv-daily/ (visited on 12/15/2021).

[22] Mattijs Jonker, Anna Sperotto, Roland van Rijswijk-Deij, Ramin Sadre, and Aiko Pras, "Measuring the Adoption of DDoS Protection Services," in *Internet Measurement Conference (IMC)*, ACM, 2016, pp. 279–285. [Online]. Available: http://dl.acm.org/citation.cfm?id=2987487.

[23] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," in *26th Annual Network and Distributed System Security Symposium*, ser. NDSS 2019, Feb. 2019. DOI: 10.14722/ndss.2019.23386.

[24] Frank Li, Zakir Durumeric, Jakub Czyz, Mohammad Karami, Michael Bailey, Damon McCoy, Stefan Savage, and Vern Paxson, "You've Got Vulnerability: Exploring Effective Vulnerability Notifications," in *USENIX Security Symposium*, USENIX Association, 2016, pp. 1033–1050. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/li (visited on 12/10/2021).

[25] Frank Li, Grant Ho, Eric Kuan, Yuan Niu, Lucas Ballard, Kurt Thomas, Elie Bursztein, and Vern Paxson, "Remedying Web Hijacking: Notification Effectiveness and Webmaster Comprehension," in *International Conference on World Wide Web*, ACM, 2016, pp. 1009–1019. DOI: 10.1145/2872427.2883039.

[26] George Mangalaraj and M. K. Raja, "Software Vulnerability Disclosure and its Impact on Exploitation: An Empirical Study," in *Americas Conference on Information Systems (AMCIS)*, Deepak Khazanchi and Ilze Zigurs, Eds., Association for Information Systems, 2005, p. 273. [Online]. Available: http://aisel.aisnet.org/amcis2005/273.

[27] Célestin Matte, Nataliia Bielova, and Cristiana Santos, "Do Cookie Banners Respect my Choice?: Measuring Legal Compliance of Banners from IAB Europe's Transparency and Consent Framework," in *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, IEEE, 2020, pp. 791–809. DOI: 10.1109/SP40000.2020.00076.

[28] David McKinney, "New Hurdles for Vulnerability Disclosure," *IEEE Secur. Priv.*, vol. 6, no. 2, pp. 76–78, 2008. DOI: 10.1109/MSP.2008.39.

[29] Miles McQueen, Jason L. Wright, and Lawrence Wellman, "Are Vulnerability Disclosure Deadlines Justified?" In *International Workshop on Security Measurements and Metrics (Metrisec)*, James Walden and Laurie A. Williams, Eds., IEEE, 2011, pp. 96–101. DOI: 10.1109/Metrisec.2011.9.

[30] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill, "ICLab: A Global, Longitudinal Internet Censorship Measurement Platform," in *IEEE Symposium on Security and Privacy*, IEEE, 2020, pp. 135–151. DOI: 10.1109/SP40000.2020.00014.

[31] OpenBugBounty Project, *OpenBugBounty*, Official website, 2021. [Online]. Available: https : / / www . openbugbounty . org / open - bug - bounty/ (visited on 12/15/2021).

[32] A. Phillips and M. Davis, "Tags for Identifying Languages," Internet Engineering Task Force, Internet Draft 5646, Sep. 2009. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5646 (visited on 12/14/2021).

[33] Tara Poteat and Frank Li, "Who You Gonna Call?: An Empirical Evaluation of Website Security.txt Deployment," in *Internet Measurement Conference (IMC)*, ACM, 2021, pp. 526–532. DOI: 10 . 1145 / 3487552 . 3487841.

[34] John P. Rula, Philipp Richter, Georgios Smaragdakis, and Arthur W. Berger, "Who's left behind?: Measuring Adoption of Application Updates at Scale," in *Internet Measurement Conference (IMC)*, ACM, 2020, pp. 710–723. DOI: 10.1145/3419394.3423656.

[35] Sudheesh Singanamalla, Esther Han Beol Jang, Richard J. Anderson, Tadayoshi Kohno, and Kurtis Heimerl, "Accept the Risk and Continue: Measuring the Long Tail of Government https Adoption," in *Internet Measurement Conference (IMC)*, ACM, 2020, pp. 577–597. DOI: 10.1145/3419394.3423645.

[36] Ben Stock, Giancarlo Pellegrino, Frank Li, Michael Backes, and Christian Rossow, "Didn't You Hear Me? - Towards More Successful Web Vulnerability Notifications," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, The Internet Society, 2018. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018%5C_01B-1%5C_Stock%5C_paper.pdf.

[37] Ben Stock, Giancarlo Pellegrino, Christian Rossow, Martin Johns, and Michael Backes, "Hey, You Have a Problem: On the Feasibility of Large-Scale Web Vulnerability Notification," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, USENIX Association, 2016, pp. 1015–1032. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/stock.

[38] Hemang Chamakuzhi Subramanian and Suresh Malladi, "Bug Bounty Marketplaces and Enabling Responsible Vulnerability Disclosure: An Empirical Analysis," *J. Database Manag.*, vol. 31, no. 1, pp. 38–63, 2020. DOI: 10.4018/JDM.2020010103.

APPENDIX

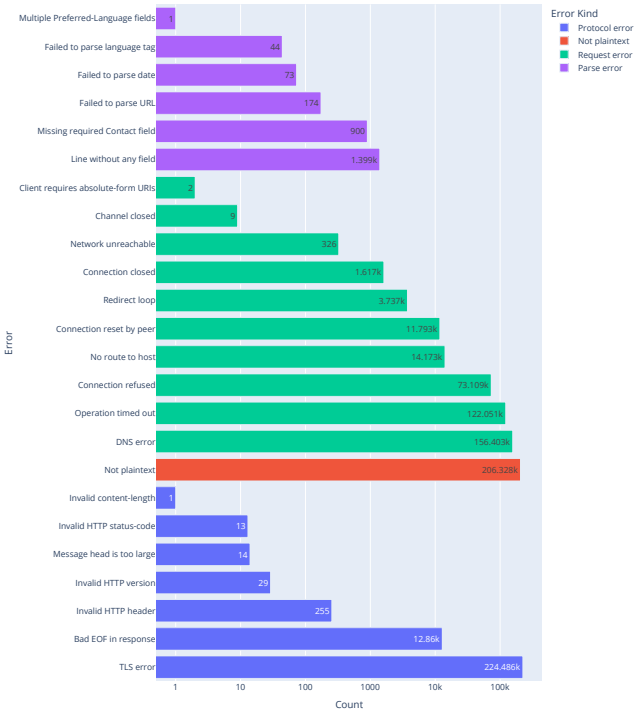

Fig. 8: Non-HTTP errors encountered while fetching and parsing Security.txt files. Protocol errors (blue) and request errors (green) collectively account for 31% of all requests. Note the logarithmic axis.

Listing 1: An example of a valid Security.txt file conforming to RFC 12 [16]. This example was generated using the guided Security.txt generator by Foudil and Shafranovich [18] and signed using the `gpg --clearsign` command in GNU/Linux. Mandatory fields are shown in blue, comments are shown in green, and the optional PGP cleartext signature is shown in purple. Note that this example follows several security best-practices; it contains a short expiry date some time in the future and includes a digital signature which can be verified using the provided encryption key. Since the file also provides the optional `Canonical` field, the signature can be used to verify the file's authenticity.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

# You may contact us at any of the following URLs:
Contact: mailto:example@example.com
Contact: https://example.com/security
Expires: 2023-01-01T04:59:00.000Z
Encryption: https://example.com/pub-key.txt
Acknowledgments: https://example.com/acknowledgments
# Use English or French to conact us please!
Preferred-Languages: en, fr
Canonical: https://example.com/.well-known/security.txt
Canonical: https://example.com/security.txt
Policy: https://example.com/policy
Hiring: https://example.com/careers
-----BEGIN PGP SIGNATURE-----

iQEzBAEBCAAdFiEE1N7b1zOZ1NHHUv2Fi/M/rc9rXxAFAmHXQn0ACgkQi/M/rc9r
XxCpNAgAjdtth/RFOSw0rXi8GzX/5vYAFGPstd8yKFq8d8TlmnmRByrngWjLO7ze
C1RcqIBID6wOibB4fOUbXAzTLEf5+HlT6scU7Z+/SZZFX9JTv4XgbC+yqXCCEyeB
uqzpQFsttdpcOC0SnQeX9dVRm5P8F+/El/nQSyUt66v1cGUW1AKfMYLEr01CvfUK
HS8cbKWE/Kc30ZglWtcDfNzP2bAQSR4YbV9vhS+XckLNUkt+Fw3UAQATOvm1KGgr
7hXGAbMw5qcavXXxo8XKYvPt6tCMCvdJXOCwXaNU5K1i2lSN3wz23HMNFo2bauFe
kBJ51fuR4xll9I9YBNhe26gn3BzPuw==
=jiaY
-----END PGP SIGNATURE-----
```