Measurement and Analysis of Automated Certificate Reissuance

Olamide Omolola[1], Richard Roberts[2], Ishtiaq Ashiq[3]
, Taejoong Chung[3], Dave Levin[2], and Alan Mislove[4]

[1] University of Vienna
[2] University of Maryland
[3] Virginia Tech
[4] Northeastern University

Abstract. The Transport Layer Security (TLS) Public Key Infrastructure (PKI) is essential to the security and privacy of users on the internet. Despite its importance, prior work from the mid-2010s has shown that mismanagement of the TLS PKI often led to weakened security guarantees, such as compromised certificates going unrevoked and many internet devices generating self-signed certificates. Many of these problems can be traced to manual processes that were the only option at the time. However, in the intervening years, the TLS PKI has undergone several changes: once-expensive TLS certificates are now freely available, and they can be obtained and reissued via automated programs.
In this paper, we examine whether these changes to the TLS PKI have led to improvements in the PKI's management. We collect data on *all* certificates issued by Let's Encrypt (now the largest certificate authority by far) over the past four years. Our analysis focuses on two key questions: First, *are administrators making proper use of the automation that modern CAs provide for certificate reissuance?* We find that a surprising fraction (40%) of domains do not reissue their certificate on a predictable schedule, suggesting a lack of use of automated tools for doing so. Second, *do administrators that use automated CAs react to large-scale compromises more responsibly?* To answer this, we use a recent Let's Encrypt mis-issuance bug as a natural experiment, and find that a significantly larger fraction of administrators reissued their certificates in a timely fashion compared to previous bugs. Overall, our results demonstrate that when given the right tools, administrators can perform their duties and improve security for all internet users.

1 Introduction

The Transport Layer Security (TLS) public key infrastructure (PKI) is an essential component of the modern internet: it allows users to communicate over the Internet in a trusted and confidential manner. However, previous work [8,21,13,2,3] has demonstrated that despite its importance, the *management* of the TLS PKI is often not compliant with recommended security practices. For example, systems administrators often fail to revoke or even reissue certificates when private

keys are compromised [20], many internet-of-things devices generate self-signed certificates (sometimes even with identical keys across devices) [13], and domains sometimes share private keys with content distribution networks due to limitations in the PKI itself [2].

Many of these management issues can be traced to inadequate tools for system administrators. For example, in the wake of the Heartbleed [11] bug in 2014, a significant fraction of web servers potentially had their private keys exposed; as a result, administrators should have revoked their old certificates and reissue new ones. At the time, doing so was a largely manual process: because certificates were typically valid for up to 5 years, many administrators presumably eschewed automating the infrequent process of obtaining and installing new certificates. As a result, it took over a *week* before even 10% of the vulnerable web servers had reissued their certificates [21]. Similarly, in the DNSSEC PKI, it has been observed that inadequate tools—in the case of DNSSEC, a manual process of uploading DS records—has lead to poor adoption of secure protocols [4].

However, the TLS PKI has changed dramatically since 2014. While previously expensive, TLS certificates are now free with the advent of certificate authorities such as Let's Encrypt [14] (which is now, by far, the most popular CA [16]). More importantly, these free CAs often have much shorter certificate lifetimes (90 days in the case of Let's Encrypt), which encourages the automation of the process of certificate reissuance and installation (as it happens every three months, rather than every five years). Open-source protocols (e.g., ACME) and tools (e.g., `certbot`, `acme.sh`, cPanel) now allow administrators to automate the entire process easily.

In this paper, we examine whether the presence of these tools and services has led to better TLS certificate reissuance. To understand the effects of automated tools in certificate reissuance, we focus on certificates issued by Let's Encrypt. We chose Let's Encrypt as it is by far the largest ACME-based CA [16], and it has the longest history of operation (and hence, the highest likelihood of having domain sets that have a long history of reissues). We use Certificate Transparency (CT) [12] logs to obtain a list of *all* 1.03B certificates Let's Encrypt issued over the past four years. We group certificates in this list by the set of domains they contain (similar to prior work [21], we refer to this as a *domain set*), enabling us to measure how often certificates are reissued.

We also use a recent bug discovered by Let's Encrypt as a natural experiment. In brief, in early 2020, Let's Encrypt discovered that over 3M certificates had been issued improperly, as they had failed to check for Certificate Authority Authorization (CAA) [19] records properly before issuance [5]. Because they were improperly issued, Let's Encrypt announced that they planned to revoke the certificates one week later, informing all system administrators that they needed to reissue their certificates. This serves as a natural experiment, as we can examine whether administrators took the necessarily manual action of reissuing their certificates, rather than simply relying on their automated reissuance.

Our paper makes two contributions: *First*, we examine how the behavior of system administrators reissuing TLS certificates have changed with the advent of

free CAs such as Let's Encrypt. We find that approximately 60% of the domain sets show a median reissuance period of 60 days (the default recommended by Let's Encrypt [14] and used by many ACME tools [23,6] for automated certificate reissuance).

Second, we use the Let's Encrypt bug mentioned above to explore whether system administrators now respond more quickly and completely when manual intervention *is* required. We focus on the subset of the 2M domain sets with a mis-issued certificate, and identify 98,652 domain sets that show a regular period of reissuance with at least one new certificate issued after the bug was discovered on February 29, 2020.[5] We demonstrate that, of these domain sets, at least 28% appear to have taken the manual steps necessary to reissue their certificates within a week, suggesting that, indeed, system administrators are better able to reissue certificates securely today.

2 Background

We begin with a brief overview of the TLS certificate ecosystem, and then detail related work.

2.1 Certificates

TLS is based on certificates, which are bindings between identities (typically domain names) and public keys. Certificates are signed by *certificate authorities* (CAs), who verify the identity of the requestor. Certificates have a well-defined validity period, which is expressed as `NotBefore` and `NotAfter` fields in the certificate; clients will refuse to accept certificates outside of their validity period. As a result, certificate owners have to periodically *reissue* their certificate by contacting their CA (or another CA) and obtaining a new certificate.

While certificates originally only contained a single identity (domain name), this often made the administration difficult for web servers that served multiple domains. Today, certificates can carry multiple identities (domain names) via a `Subject Alternate Names` list. In essence, the owner of the public key in the certificate has been verified by the CA to control *all* of the identities (domain names).

Finally, domain owners may wish to limit the set of CAs who are authorized to issue certificates for a given domain. They can now do so by publishing Certificate Authority Authorization (CAA) records, which are DNS records that specify a list of CAs that are/are not allowed to issue certificates (if no such record exist, all CAs are implicitly authorized). CAs today are required to check for the CAA records for domains before issuing certificates.

[5] Because of the way the bug manifested itself, the mis-issued certificates are *not* a random sample of all certificates. We explore this in Section 3.

2.2 Let's Encrypt

For a long time, TLS certificates were relatively expensive to obtain (typically $50 or more) and were valid for multiple years (typically 3–5) [13]. The cost and extended validity ended up having two effects: the overall adoption of HTTPS was relatively low (as administrators had to spend significant money to obtain the necessary certificates), and the system administrators who did purchase certificates were not incentivized to automate the infrequent reissuance process. Additionally, the certificate issuance and renewal processes were manual, administratively burdensome, and technically cumbersome.

In 2015, Let's Encrypt disrupted the TLS certificate business model by offering *free* certificates that were valid for 90 days. Other free CAs have also been created such as ZeroSSL[6] and Buypass[7], and the TLS ecosystem has since changed dramatically: the fraction of web connections using HTTPS has increased from ∼27% in early 2014 to ∼85% in 2020 [16], and Let's Encrypt is now the largest CA, with over 1B certificates issued and over 35% of the Alexa top 1M sites using Let's Encrypt certificates [1]. Importantly, while prior CAs often required certificates to be requested/reissued via web forms, Let's Encrypt is entirely automated via the ACME protocol; several popular ACME clients exist, including `certbot`, `acme4j`, and `acme.sh`.

In February 2020, Let's Encrypt announced that they discovered a bug in the Boulder software they used to issue certificates [5]. Specifically, the software failed to properly check for CAA records in requested certificates if (a) a certificate was requested for multiple domains, and (b) Let's Encrypt had previously checked the domain control validations (DCV) for these domains in the preceeding 30 days. While Let's Encrypt was supposed to re-check the CAA record for all domain names included in the certificate within 8 hours of issuing the certificate, under these circumstances, it only picked one domain name among the multiple domains in the certificate and ran the CAA check n times (equivalent to the number of domains in the certificate). Let's Encrypt originally announced on February 29, 2020 that it planned to revoke all these certificates on March 5, 2020, and it emailed all affected domain administrators. On March 5, 2020, Let's Encrypt reversed their decision and decided to not revoke en-masse [15].

2.3 Related work

Improvements in the ability to scan the Internet [10] in 2013 have led to a better understanding of the entire TLS ecosystem [9]. Researchers have unfortunately found that TLS clients and servers are often incorrectly managed [13], leading to reduced security for internet users. In the aftermath of the Heartbleed bug, it became evident that manual revocation and reissuance of certificates is a major security problem: most administrators failed to revoke or even reissue, and those that did sometimes reissued using the same key pair [21,8]. Similar behavior had

[6] `https://zerossl.com/features/certificates/`
[7] `https://www.buypass.com/ssl/products/acme`

been observed years prior when a bug in Debian caused many domains the need to reissue certificates [20]. Some domains have chosen to outsource certificate management to third-parties such as content delivery networks (CDNs); while this improves certificate management, it often requires sharing private keys [2].

To the best of our knowledge, there has not been significant study of automated certificate reissuance in the TLS PKI. Previous work by Matsumoto et al. proposed a decentralized audit-based system: Instant Karma PKI (IPK) to promote automation among HTTPS domains [18]. The recent development of CAA records also provides a useful tool for automation as the domain name holders or DNS operators can use CAA records to control which CAs that they would like to get a certificate from [19].

3 Methodology

We now describe the datasets we collected and our methodology to determine a set of certificates that have been reissued.

3.1 Certificates

Our goal is to see how certificates have been (re)issued by the system administrators. We focus on Let's Encrypt as it is the largest free CA, and it has the longest history of operation. To this end, we obtain *all* certificates issued by Let's Encrypt by leveraging the Certificate Transparency (CT) logs; when issuing a certificate, Let's Encrypt publishes the certificate to one of the CT logs managed by Google.[8] Thus, to obtain a nearly complete view of the certificates issued by Let's Encrypt, we first fetch all certificates from all of the CT log servers managed by Google,[9] obtaining 5.3B certificates in total from September 9, 2014 to May 18, 2020. We then identify the certificates issued by Let's Encrypt according to their `Issuer` field, which leaves us with 1.03B certificates.[10]

3.2 Let's Encrypt CAA bug list

On February 29th, 2020, Let's Encrypt announced the CAA issuance bug in their certificate issuance process (see § 2.2). Let's Encrypt publicly released a list of the certificates impacted by this bug [5] containing serial numbers of 3,048,289 certificates, some of which were potentially mis-issued (i.e., the CAA records for some of domains in the certificate may have not permitted Let's Encrypt to issue a certificate, even though they did). We use this list to study how the impacted certificates have been *reissued* by administrators.

[8] In order for a certificate to be "CT qualified" in modern browsers such as Chrome, it has to be logged on multiple CT log servers and one of them has to be from a Google log [7].

[9] `aviator`, `icarus`, `argon2018`∼`2023`, `xenon2019`∼`2023`, `pilot`, `rocketeer`, `skydiver`

[10] We intentionally exclude pre-certificates from the analysis (which Let's Encrypt has published as well since 2018 [17]) as they do not guarantee the issuance of their actual (final) certificates.

3.3 Defining Certificate Reissuances

While it is easy to identify when certificates are issued, there is a bit of subtlety to determining when they are *reissued*. In particular, we face two challenges: *First*, CT logs do not contain any identifier of the client such as IP address that sent a Certificate Signing Request (CSR), thus making it hard to identify if the certificate has been reissued from the same client; thus, we first link the certificates that share the same `Subject Alternate Name` (SAN) list.[11] We refer to this set of domains in the SAN list as the *domain set*. *Second*, we do not know when the client has replaced the old certificate with the new one; thus, we use the logging timestamp on the CT log server as a proxy.

In summary, we group certificates by their domain set and order them based on their timestamp on the CT logs; we refer to any certificates other than the first as reissued certificates. Using this, methodology we obtain 188M unique domain sets and 1.03B corresponding certificates issued during our measurement period. Out of the 188M domain sets, we find that 67M (35.7%) domain sets had no reissued certificates. One limitation of relying on CT logs alone worth noting is that we are unable to quantify how domain sets change, as we would need a way to "link" domain sets which is unavailable to us [2]. In these cases, the modified domain set would be considered a separate domain set in our analysis.

4 Results

We analyze the reissuance behaviors of certificates issued by Let's Encrypt. We aim to understand reissuance behavior of two types: reissuance that is likely done *automatically* (e.g., via a `cron` job) and reissuance that is likely done *manually* (e.g., directly invoked by a system administrator). We begin by describing how we distinguish these two cases.

4.1 Automated Reissuance

One of Let's Encrypt's key principles is that it makes it possible to automate obtaining and reissuing certificates. A new user of Let's Encrypt need only set up the first certificate issuing process with any ACME client of choice, then they can create a `cron` job to continually check if the certificate is still valid and request a new certificate once the current certificate nears expiry.

We first need to identify when we believe a certificate has been reissued via an automated process. As discussed previously we are not privy to Let's Encrypt's internal logs, so we can only rely on publicly available data from the CT logs. To do so, we group all Let's Encrypt certificates by the domain set present in them, and then sort these lists by the time in the CT log timestamp. We then examine the amount of time that passes between each pair of successive reissues.

Explain how top 10 CAs chosen (Scott Helme daily measurements), LE not included in top 10, explain what graph means

[11] Thus, if the same client adds or removes one domain and obtains the new certificate, it will not be considered as a reissuance.

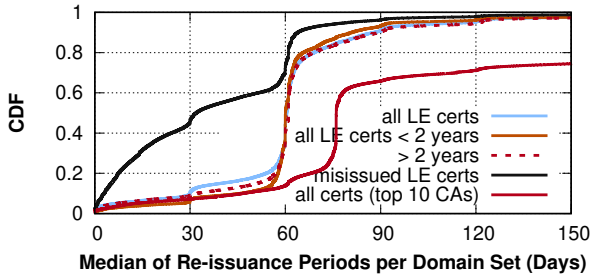

Fig. 1. Distribution of median reissuance per domain set for all Let's Encrypt certificates with or without lifetimes and mis-issued certificates

In Figure 1, we plot the cumulative distribution of the median of these reissue time lists in the line labeled "all LE certs." We immediately observe a large "spike" around 60 days, and observe that over 55% of domain sets have a median reissue time between 55 and 65 days. This lines up with the reissuance policy recommended by Let's Encrypt, which recommends reissuing certificates that are within 30 days of their expiry (i.e., are at least 60 days old) [14]. Moreover, this timing lines up well as the default policies of many ACME clients: `cerbot` [6] and `acme.sh` [23] both default to renewing within 30 days of expiry. We also observe that the "spike" does not happen *entirely* at the 60 day mark; this is likely because the renewal occurs the first time the `cron` job runs after reaching the mark. Finally, we observe a much smaller spike around 30 days, which is likely the behavior of a different ACME client or a system administrator who manually changed their client's behavior.

Next, we examine whether this median reissue period of 60 days is only present in domain sets that have a long history of being reissued (i.e., that have been around a long time) or if it is also present in newer domain sets. To do so, we divide the "all LE certs" line into those first issued greater than two years ago, and those first issued within the past two years; these are both plotted in Figure 1. We can observe the shapes of these curves are quite similar, suggesting that the behavior is relatively consistent between these two groups.

Finally, we discover that roughly 10% of domain sets in all categories had a median re-issuance period of *greater than* 90 days, meaning the certificates were more often than not renewed after expiry. This behavior could occur if the administrator did not set up a `cron job`, incorrectly set up a `cron` job to run very infrequently, or if the system was not always online. We leave a deeper exploration of these domain sets to future work.

Coefficient of Variation (CoV) While the median of the reissue time periods being so clearly at 60 days is suggestive that the administrators use automated software to reissue their certificates, it is not entirely definitive. Thus, we look for further evidence of automation by looking at how *similar* the reissuance periods of a given domain set are to each other. In other words, if a given domain set

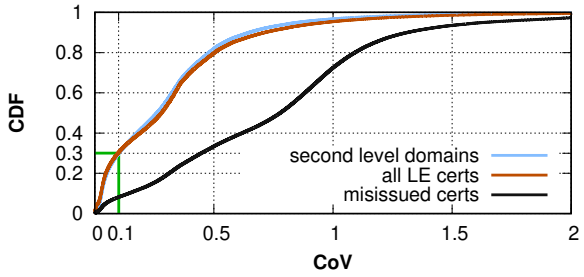

Fig. 2. Distribution CoV for second level domains and certificates

was using an automated process to reissue certificates, we would expect that the period between reissues would be highly consistent.

To do so, we calculate the *coefficient of variation* (CoV)—which is simply the standard deviation of a distribution over its mean—of the amounts of time between each successive reissuance. Automated reissuance would often lead to a consistent period between reissues, meaning that the CoV would be low i.e., 0.1 or smaller. We choose the CoV threshold of 0.1 as a cut-off as would allow, for example, a domain set with a mean reissuance time of 60 days to be classified as automated if the variance is less than 6 days (roughly one week). For this analysis, we only keep the domain sets where we have a sufficient reissue history of at least five reissues. Figure 2 plots the distribution of CoVs for the reissue time periods for each domain set under the "all LE certs" line. We can observe that many domain sets do show evidence of automation: 30.3% of domain sets have a CoV of less than 0.1.

We were concerned that particular domains with unusual patterns of reissuance may end up artificially shaping this curve, as our analysis is at the *domain set* level, rather than at the *system administrator* level.Thus, we additionally perform an aggregation to the second-level domain to see whether particular domains are skewing the results.

We aggregate domain sets into second-level domain through a weighted average: for each second-level domain S, we compute the average CoV for all domain sets that have at least one domain name from S. For domain sets that include domains from multiple second-level domains, we simply weigh the domain set's CoV by the fraction of domains that belong to S. The resulting cumulative distribution is also shown in Figure 2, and we can observe that the distribution is quite similar to the analysis at the domain set level. Thus, we have some confidence that the (potentially odd) behavior of a small number of second-level domain sets is not dramatically altering the results.

Noticing that many domain sets tend to have a high CoVs, we next examine how well the CoV methodology identifies domain sets with regular reissuance patterns. We do so by dividing up domain sets by their CoV, and plotting the cumulative distribution of their median reissuance time in Figure 3. We can immediately observed that the median reissuance time of certificates varies dra-

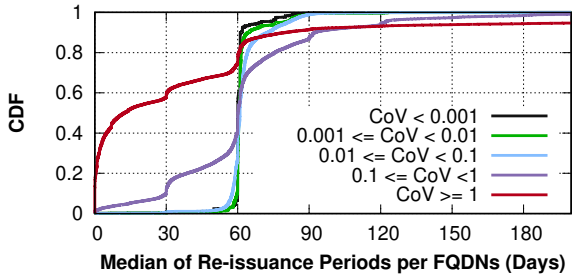

Fig. 3. Distribution of all Let's Encrypt certificate reissuances within CoV groups

matically by CoV: we find that the median reissuance period of domain sets with a very low CoV (0.1 or smaller) is 60 days, while domain sets with a CoV greater than 1 are much less predictable. Further, Figure 3 reveals that over 88% of domain sets with highly automated reissuance (CoV < 0.001) have a median reissue period of between 59 and 61 days (consistent with the reissue occurring during the first `cron` job to run after the 60 day period).

Initial renewal setup Moving on, we hypothesize that the initial setup and use of ACME clients may result in multiple, irregular requests, which would affect our CoV calculation. To understand the effects, we focus on certificates that have at least five reissues, and make the assumption that most administrators would be comfortable with operating ACME clients after a year. Out of 188M unique domain sets, only 60M unique domain certificates have at least five reissues; these form the basis of the following analysis.

Roughly 48.2% of these domain sets have a CoV less than 0.1. However, if we also look at *subsequences* of reissues, ignoring the first set of reissues as long as at least five reissues remain, we can identify an *additional* 29.9% of the domain sets that have a subsequence of reissues with a CoV less than 0.1. In other words, 78% of domain sets we consider have a regular reissue cycle that begins at some point in their lifetimes. Thus, we have identified a limitation of the CoV metric, as it may be too conservative in cases where administrators have an irregular initial reissunce cycle before fully debugging their ACME client setup.

4.2 Manual Reissuance

Having a good understanding on domain sets with likely automated reissuance infrastructure, we now turn to examine what happens for these domains when manual intervention is required. To do so, we use the Let's Encrypt mis-issuance bug as a natural experiment: because all of these certificates need to be reissued, we have a collection of domain sets where we can study whether the system administrator did, in fact, reissue their certificate.

We first need to examine the set of certificates affected by the bug, which was announced on February 29, 2020. Let's Encrypt reported that over 3M certificates were affected; we collected all of these certificates and plotted their issue

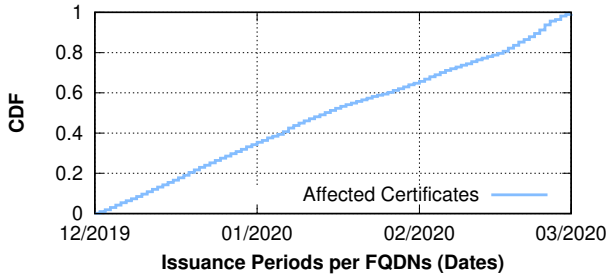

Fig. 4. Distribution of all mis-issued certificate logtime

time in Figure 4. We can see that these certificates went as far back as December 2, 2019, which would be expected given Let's Encrypt 90-day certificate lifetime. Importantly, the certificates appear to have been issued uniformly throughout the prior 90 days.

However, there are multiple reasons why these mis-issued certificates are *not* a random sample of all Let's Encrypt certificates. *First*, the bug only affected certificates with multiple domains in them, meaning any certificates with a single domain were not mis-issued. *Second*, and more importantly, it only affected domains *where the CAA record had been verified within the past 30 days*. As we observed previously, most certificates are reissued after *60* days, this means that the only certificates that were affected were ones that were either (a) not on a regular schedule to begin with, or (b) were on a regular schedule, but happened to be reissued in late 2019/early 2020 for another reason. This observation explains why the mis-issued certificates behave quite differently from all Let's Encrypt certificates in Figures 1 and 2: due to the nature of the bug, domain sets that had regular, 60-day reissue periods were much less likely affected. In fact, such domain sets would only have been affected if one of the domains in the domain set happened to be in *another* domain set whose certificate was reissued in the previous 30-day time period, or where the administrator had manually reissued that domain set during that period.

Nevertheless, we need to identify when we believe a certificate was manually reissued from among the mis-issued certificates. Recall that we do not have access to Let's Encrypt's logs, so we can only rely on the timestamps public CT logs. We want to see how certificates affected by the bug were automatically reissued before the bug, but manually issued a new certificate in response to the bug. We therefore focus on those domain sets that (a) were affected by the Let's Encrypt bug, (b) were on a regular cycle prior to February 29, 2020, and (c) had at least one new certificate issued after February 29, 2020 (to see if the regular cycle continued). To see if a domain set was on a regular reissue cycle prior to February 29, 2020, we see if the five certificate reissues prior to the bug date had a CoV less than 0.1. In total, 98,652 domain sets satisfy these three criteria.

Next, we calculate the CoV of the five reissues before the bug date *and* the first reissue after the bug date. If the CoV including the new certificate is high,

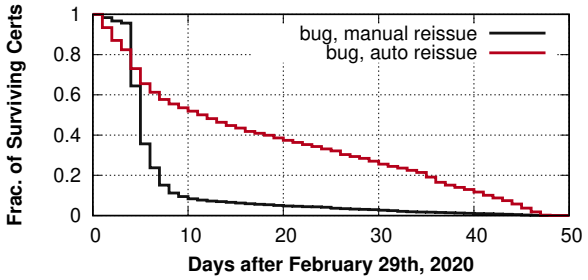

Fig. 5. Survival graph comparing reissues after LE Bug

then the first certificate after the Let's Encrypt bug could not have been automatically reissued; some form of manual intervention disrupted the issue cycle and caused the previously low CoV to increase. If the CoV including the new certificate remains low (<0.1), then the new certificate was likely issued on its expected regular schedule. It is also possible, though unlikely, that a new certificate was manually issued at the same time we would expect the next automatically reissued certificate. Of the 98,652 domain sets, 33,099 saw a significant CoV increase (i.e., likely had manual intervention) in the first reissue after the Let's Encrypt bug, and 65,553 likely did not.

We now examine *how quickly* these 33,099 domain sets were manually reissued after Let's Encrypt announced the bug, and emailed all administrators to tell them to reissue their certificates manually. Figure 5 plots the number of these certificates that survive in the line labeled "bug, manual reissue". We can observe that most certificates that are manually reissued are reissued quite quickly: within a week, over 84% of all certificates that we believe are manually reissued have been reissued. For comparison, we plot the same graph for the 66,553 certificates that were reissued close to their next reissue in the line labeled "bug, auto reissue". This group shows less-prompt reissuing than the manual reissues, as only 42% of likely-automatic reissues occurred in the 7 days following the bug announcement.

Finally, we plot the same data as in Figure 5, but do so as a fraction of *all* mis-issued domain sets with a CoV less than 0.1 before the bug date. This graph is presented in Figure 6, and it shows that among all the domain sets with a CoV less than 0.1 (those on a regular schedule before Feb. 29, 2020), at least 28% had reissued their certificate manually within a week of the bug announcement. This result is a significant improvement over prior incidents; with the Heartbleed bug, after a week, barely 10% of affected certificates had been reissued (and even fewer revoked) [22]. Even though circumstances between the two bugs differ significantly (such as notification of revocation), they both provide opportunities for natural experiments to see how the PKI is evolving over time, and the comparison suggests that system administrators may now be better managing the PKI.

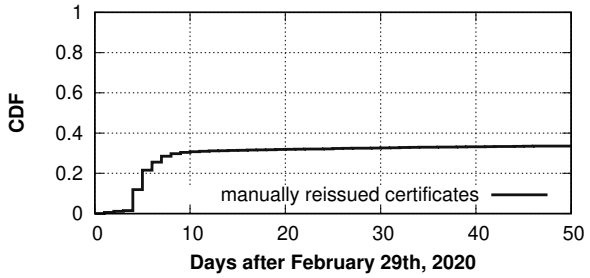

Fig. 6. CDF graph for manual reissues after LE Bug

5 Concluding discussion

Over the past five years, the TLS PKI ecosystem has changed dramatically: largely due to new CAs such as Let's Encrypt, we have moved from primarily expensive, long-lived certificates to primarily free, short-lived certificates. In this paper, we examined whether this change in the nature of the certificate ecosystem has also improved the *management* of the TLS PKI, as it has been previously been observed that system administrators often fail to properly manage their certificates. Using over four years of CT logs, we find significant evidence that most clients of Let's Encrypt have indeed set up automated processed for reissuing and installing their certificates. Moreover, we find evidence that even when manual intervention is required, system administrators are more prompt in doing so when compared to studies from the 2014 Heartbleed bug and the 2009 Debian PRNG bug. Taken together, our results underscore the need to improve the management of the TLS PKI, and how changes in the ecosystem can lead to significant management improvements, even as the ecosystem of TLS certificates has grown dramatically.

Acknowledgements

We thank the anonymous reviewers and our shepherd, Cecilia Testart, for their helpful comments. This research was supported in part by NSF grant CNS-1900879.

References

1. J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla, S. Schoen, and a. B. Warren. Let's Encrypt: An Automated Certificate: Authority to Encrypt the Entire Web. *CCS*, 2019.
2. F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem. *CCS*, 2016.

3. T. Chung, Y. Liu, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Measuring and Applying Invalid SSL Certificates: The Silent Majority. *IMC*, 2016.
4. T. Chung, R. van Rijswijk-Deij, D. Choffnes, A. Mislove, C. Wilson, D. Levin, and B. M. Maggs. Understanding the Role of Registrars in DNSSEC Deployment. *IMC*, 2017.
5. CAA Rechecking Bug. `https://community.letsencrypt.org/t/2020-02-29-caa-rechecking-bug/114591`.
6. Certbot User Guide. `https://certbot.eff.org/docs/using.html`.
7. Certificate Transparency in Chrome. 2019. `https://github.com/chromium/ct-policy/blob/master/ct_policy.md`.
8. Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The Matter of Heartbleed. *IMC*, 2014.
9. Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS Certificate Ecosystem. *IMC*, 2013.
10. Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-Wide Scanning and its Security Applications. *USENIX Security*, 2013.
11. Heartbleed Bug. `http://heartbleed.com`.
12. B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, IETF, 2013. `http://www.ietf.org/rfc/rfc6962.txt`.
13. Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, A. Schulman, and C. Wilson. An End-to-End Measurement of Certificate Revocation in the Web's PKI. *IMC*, 2015.
14. Let's Encrypt. `https://letsencrypt.org`.
15. Let's Encrypt Community Support: 2020.02.29 CAA Rechecking Bug. `https://community.letsencrypt.org/t/2020-02-29-caa-rechecking-bug/114591/3`.
16. Let's Encrypt Stats. `https://letsencrypt.org/stats/`.
17. LetsEncrypt: Submit final certs to CT logs (#3640). `https://github.com/letsencrypt/boulder/commit/1271a15be79b9717ee5b98e707b76e7ac86a9a0e`.
18. S. Matsumoto and R. M. Reischuk. IKP: Turning a PKI Around with Decentralized Automated Incentives. *IEEE S&P*, 2017.
19. Q. Scheitle, T. Chung, J. Hiller, O. Gasser, J. Naab, R. van Rijswijk-Deij, O. Hohlfeld, R. Holz, D. Choffnes, A. Mislove, and G. Carle. A First Look at Certification Authority Authorization (CAA). *CCR*, 48(2), 2018.
20. S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When Private Keys Are Public: Results from the 2008 Debian OpenSSL Vulnerability. *IMC*, 2009.
21. L. Zhang, D. Choffnes, T. Dumitraş, D. Levin, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. *IMC*, 2014.
22. L. Zhang, D. Choffnes, T. Dumitraş, D. Levin, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL Certificate Reissues and Revocations in the Wake of Heartbleed. *CACM*, 61(3), `https://cacm.acm.org/magazines/2018/3/225489-analysis-of-ssl-certificate-reissues-and-revocations-in-the-wake-of-heartbleed/fulltext`, 2018.
23. acme.sh. `https://github.com/acmesh-official/acme.sh`.