Private Key Leaks in the Wild: Insights from Certificate Transparency

Gaëtan Ferry[1], Guillaume Valadon[1], David Tao[2], and Philippe Boneff[2]
gaetan.ferry@gitguardian.com
guillaume.valadon@gitguardian.com
dtao@google.com
phboneff@google.com

[1] GitGuardian
[2] Google

Abstract. Private key leaks represent a critical security vulnerability, with over 430,000 leaked keys on GitHub in 2025, yet their real-world impact remains largely unknown due to the challenge of linking these mathematical objects to their operational usage. We present the first systematic analysis mapping leaked private keys to active certificates, combining GitGuardian's dataset of 945,560 unique leaked private keys with Google's historical Certificate Transparency databases. Our methodology successfully mapped 42,690 private keys to 139,767 certificates, revealing the impact of private keys leaked on GitHub and DockerHub. Using custom online and offline validation, we identified 2,622 valid certificates, enabling website impersonation and MITM attacks. Our analysis reveals systematic failures in certificate revocation practices, with only 80 certificates revoked via CRL/OCSP and just 3 properly marked as key-compromised. Finally, we successfully attributed certificates to 600 organizations across critical industries, though many could not be mapped to identifiable owners. With 20% of valid certificates having been exposed for over two years, our large-scale responsible disclosure campaign sent thousands of emails and revealed significant challenges in reaching certificate owners.

1 Introduction

Public secret leaks are a prevalent problem, yet the corresponding risks are widely underestimated despite recurring breaches that exploit them, such as the recent Salesloft Drift breach [17]. On the GitHub platform alone, more than 29 million secrets were leaked in 2025, a 34% increase from 2024 [7]. This includes close to 430,000 non-encrypted private keys stored in structured formats like PEM.

To external observers, these leaked private keys appear as nothing more than abstract mathematical objects without any information about ownership or associated usages. Indeed, this makes it challenging to assess

their real-world impact. Unlike vendor-specific secrets that can be linked to a service provider, such as GCP, and potentially validated, private keys cannot be traced back to their intended systems or owners without further investigations. This complexity slows down both risk assessment and remediation efforts for private key leaks.

More commonly, these private keys are used in a specific context, such as protecting communications with the TLS protocol. As a result, these mathematical objects are associated with contextualized data structured in the form of X.509 certificates, which define, among other things, their usages, owners, and validity. In the TLS ecosystem, a private key leak poses a critical threat, as attackers on the appropriate network path can impersonate websites, intercept or manipulate data, and decrypt past communications, particularly if the same private key has been used for a long period of time before the widespread adoption of PFS.

Since a private key and a certificate are linked unambiguously – the latter contains the public key associated with the former – it is possible to verify their relationship by calculating and then comparing the public key hashes. When simultaneous leaks of keys and certificates enable immediate attribution and validity checks, this analysis remains static and time-bound. If the private key keeps being used after its exposure, it creates an ongoing security vulnerability where attackers can still compromise communications secured with newer certificates.

Initially designed to detect fraudulent or mistakenly issued certificates, Certificate Transparency (CT) provides an elegant solution to this ownership challenge by publishing public, tamper-proof, append-only logs of all the certificates issued by Certificate Authorities. With CT, the attribution logic is, in theory, straightforward: download the logs, verify their integrity, parse the certificates, compute, then compare public key hashes with the leaked private keys. In practice, implementing such a strategy is far more challenging.

2 Methodology

Working with Certificate Transparency logs at scale raises three distinct challenges: storage volume, processing time, and archive impermanence (see Section 5.2).

To address these scale and complexity challenges, we combined GitGuardian private keys and Google's historical CT databases to directly address the goal of retrieving all certificates associated with known leaked private keys. With those two datasets at hand, the research complexity

goes back to comparing Subject Public Key Information hashes computed from both the private keys and certificates. To our knowledge, this is the first time that so many leaked private keys have been combined with the CT logs to find their purpose and owners.

One of our goals was to discover leaked private keys and corresponding certificates that can be used by attackers to impersonate websites. Therefore, we checked the certificates online and offline to make sure they were valid and to see if the private keys were compromised. The online checks involved connecting to one of the subjects in the certificate using a standard TLS stack over 443/TCP, and verifying if the received public key hash is the expected one. The offline checks simulated the steps taken by a TLS stack to check things like signatures, lifetimes, and CRL and OCSP revocation status.

Assuming that private leaks are not handled properly and certificates are not revoked, we approximated TLS stack validation to the certificate lifetime only. While this could be considered a weak result, it does allow us to estimate the upper limit of the number of certificates exposed each year using commit metadata.

3 Results

3.1 Origin of Private Key Leaks

In July 2025, we extracted 945,560 unique private keys from GitGuardian's dataset, dating back to 2021, after CT was created, and successfully mapped 42,690 of them in Google's historical CT Logs dataset. Most of these private keys, 32,696 (76.58%), were discovered in public commits on GitHub, while the remaining 9,994 (23.42%) come from public Docker images published on DockerHub.

As few as 4.5% of the private keys were found in CT Logs. At first glance, this may seem insignificant, but it should be noted that there is a strong bias in the leaked keys dataset: not all of these private keys are used for TLS server authentication. Indeed, we managed to manually identify other cryptographic usages not related to TLS and the CT ecosystem, namely SSH, JWT, VPN, or even tests and documentation.

3.2 Certificate Analysis

These private key leaks correspond to 139,767 unique certificates stored in historical CT Logs; 77.12% were discovered on GitHub (see Table 1). That's three times more than the number of unique private keys, and is

explained by the fact that certificates are renewed over time, and that CT Logs also contain pre-certificates. As an example, a single specific private key has been found to be linked to 4,614 certificates over a period of 3 years. For certain private keys, this phenomenon is exacerbated by the short validity period of certificates, such as the 90-day period offered by Let's Encrypt.

All	**Unique Certificates**	**Unique Private Keys**
Total	139,767	42,690
GitHub	107,789 / 77.12%	32,696 / 76.58%
Docker Hub	39,978 / 22,88%	9,994 / 23,42%

Table 1. Origin of leaks

Since organizations often reuse the same private key across multiple certificate renewals (explaining why 42,690 unique keys map to 139,767 certificates), we simplify our analysis by approximating that a private key is equivalent to a certificate for counting purposes (see Table 2). Consequently, we report the impact based on unique compromised keys rather than inflating numbers by counting every renewed certificate. This is an acceptable trade-off as 96% of the certificates are renewed less than 10 times.

Valid Only	**Unique Certificates**	**Unique Private Keys**
Total	4,972	2,622
GitHub	3,715 / 74.71%	1,888 / 72.00%
Docker Hub	1,257 / 25.29%	734 / 28.00%

Table 2. Unique certificates vs unique private keys

Certificate Validity As of September 12, 2025, a total of 2,622 certificates were still valid, accounting for 6% of the 42,690 unique private keys. We validated 35% (921) of them using basic online checks, while we had to simulate TLS stack checks for the others 65% (2,568).

This simulation provides a rare insight into how revocation is used. However, as some might argue, only a few certificates are revoked as they should be. Regarding CRL, 24 certificates are invalid, including a single one with an explicit *key_compromise* status. For OCSP, 56 are invalid, with only two having a *keyCompromise* status.

By comparing the results of online and offline checks, we discovered an interesting phenomenon: 21.77% of certificates validated offline differ from those currently served online. This suggests one of two things: either the owners are unaware of the leak and renewed the certificate and keys for operational reasons, or they do not understand the need to revoke a certificate.

Hostname and Certificate Authority Analysis Among the valid certificates, the top stems are wildcards (1189 / 45.34%) and www (771 / 29.40%). Alone, both stems account for the vast majority of the certificates that we retrieved.

Regarding Certificate Authorities, the top 5 concentrate 87.56% of the compromised certificates. Let's Encrypt is second and issued 28.52% of them. Focusing on wildcard certificates, Let's Encrypt issued 132 / 11.10% of them, while the four other authorities from the top 5 issued 76.58% of them. As wildcard certificates are usually more expensive than regular ones, this is an interesting result that indicates that not only free and cheap certificates are leaked.

Duration of Exposures on GitHub When a private key leaks on GitHub, it is possible to retrieve metadata such as the date of the initial exposure. Over the 2,622 certificates, 1,888 come from GitHub and allow computing the duration of exposure as the difference between the value of the *Not Valid After* field of the most recent certificate, and the date of first public exposure.

Figure 1 shows that 57.36% leaked in 2025, and 20.44% leaked 2 years ago or longer.

Past Exposures on GitHub Since revocation mechanisms seem to be rarely used, it is tempting to simulate valid certificates at the time of the leak using data from GitHub, and the certificates' lifetime. Thus, a certificate may be considered valid if the private key leak occured before or during the certificate's validity period.

Figure 2 shows thousands of valid certificates compromised every year due to a private key leak. In total, that is 23,985 certificates; 17.16% of the 139,767 certificates discovered in CT.

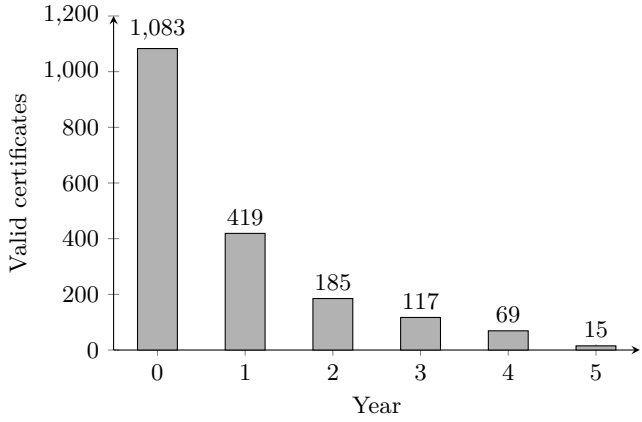

Fig. 1. Private key exposures duration (in years)

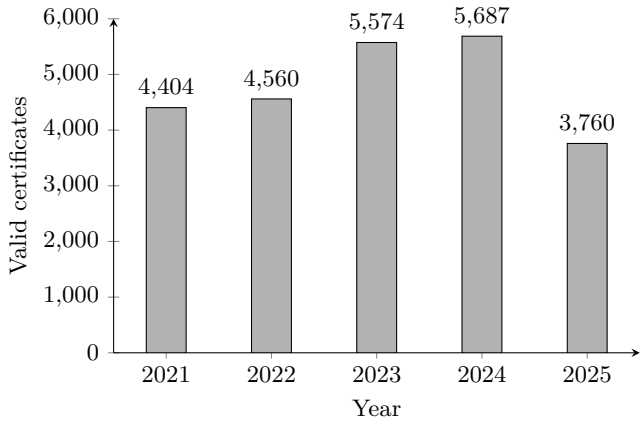

Fig. 2. Simulation of valid certificates at the time of first leak

4 Responsible Disclosures

Given the severity of these findings, identifying and notifying the affected organizations became a priority. Using various methods, including analyzing the Organization fields from certificates, we managed to map 25% of these valid certificates to 600 organizations across many industries, such as Information Technology (including a Certificate Authority), Automotive, Oil & Gas, Healthcare, and Public Administration. Given the broad impact of these findings, we attempted to notify all the certificate owners in a complex and large-scale responsible disclosure campaign. In total, this represented more than 4,300 emails sent to 1,706 distinct entities in this

disclosure campaign. It also included 9 Bug Bounty Program submissions when no better communication channel was available.

4.1 Target Identification

Leaked private keys pose a serious threat to the security of the trust infrastructure of their owner. Having identified the keys and found the valid certificates associated with them, responsibly reporting on the danger to the affected individuals and companies is a necessary next step.

Certificates are used to bind a public/private key pair to an identity thanks to the certificate subject's field. However, this identity is most often the hostname of a TLS-protected website that uses this certificate. Finding the real owner's identity or contact information can become a challenge.

We used a combination of methods to determine those disclosure targets. One of them was to rely on the *Organization* field in the subject's attribute of the certificates. However, over the nearly 5,000 valid certificates to report on, only 430 included this field.

Other methods involved exploiting Whois databases, Open Source Intelligence Techniques, and other Marketing tools.

All those methods combined allowed the identification of 1,300 certificate owners representing 600 distinct companies. Other issues were reported to a selection of email addresses deduced from the certificates' hostnames or the corresponding website content, on a best-effort approach. In that task, AI Agents proved useful to analyze and crawl public websites bound to compromised certificates' domains programmatically. Unfortunately, this approach failed to provide contact information for 1,300 certificates, either because the domains did not have public DNS information available or had no MX record or crawlable HTTP site.

Note that a specific methodology has been used for all government-related certificates, for which the corresponding information was directly reported to national CERTs.

An important point to note is that the list of entities we reported to is not limited to uneducated internet users or small companies without dedicated security teams. Contrary to our initial expectations, we had to report private key leak issues to 19 Governments and multiple Fortune 500 and other security companies, one of which operates a certification authority. This fact highlights how widespread the underlying private key-related misconceptions are.

4.2 Responses and Reception

In October 2025, the disclosure campaign received only 54 answers, for a total of 9% of contacted entities. This low response rate cannot be attributed solely to poor contact information quality. Indeed, reports sent to companies identified with close to 100% certainty received a response in only 36% of the cases. Among the 20 contacted national CERTs, only 2 answered a week after the disclosure email was received.

This poses a question regarding the understanding and consideration of the role of private keys in the security of the TLS protocol. While the importance of HTTPS in browsing the web is now well understood, even by the non-technical end-user population, it appears that private keys' security is still a challenge, even for the technical, security-aware population.

The discussion that followed the BugBounty submission messages strongly suggests a gap in securing the Internet's trust infrastructure. Over the 9 Bug Bounty submissions, 3 required a Proof of Concept of the impact of the leak, and 3 were closed as informative without action. Only 3 were properly processed by the triage and corporate security teams.

This lack of education is also reflected in some of the mail answers we received from certificate owners. While not widespread, we observed certificate owners confusing private keys and certificates or certificate validity and usage on exposed assets.

4.3 Revocation by Certificate Authorities

After the low response rate we observed on our disclosure campaign, we turned to certificate authorities directly to discuss a better solution to have this issue fixed. No Certificate Authority was able, or willing, to share contact information for the impacted certificates. This seems to make sense from a privacy standpoint.

Instead of contacting the certificate owners, most Certificate Authorities proposed to perform a direct revocation of the compromised certificates. This operation also triggers a blacklisting of the associated private key to prevent any future issuance.

Most major Certificate Authorities publish a Certification Policy and Certification Practice Statement (CP/CPS) that provides the necessary information regarding their certificate lifetime management process. This document generally follows the structure defined in the informational [3], which contains a *Certificate Revocation and Suspension* section explaining the intended revocation process.

Each authority is responsible for defining its revocation process. We have seen two main mechanisms to submit revocation requests:

— direct contact with proof of ownership;
— dedicated service (web portal or API) with a proof of ownership.

In both cases, the authorities require the submission of proof of ownership, which can be:

— the private key itself;
— a message signed using the private key, usually in the form of a CSR.

There is no standard procedure, which can make the whole process difficult when a lot of different authorities need to be contacted. We also observed that some authorities, nested under bigger operators, sometimes ask for the revocation to be submitted to the parent authority. Because certification practice documents are not easily discoverable from a certificate alone, identifying those corner cases is not a smooth process.

Overall, we submitted 2,193 certificates for revocation to 9 certificate authorities, see Table 3. Certificates not included in this list had already expired at the time of revocation.

Certificate Count	CA Name
1,264	Sectigo
366	GoDaddy
256	GlobalSign
153	Digicert
54	GoGetSSL
52	Internet2
22	SSL Corporation
22	Starfield Technologies
4	Hellenic Academic and Research Institutions CA
2,193	**Total**

Table 3. Certificates disclosed to CA

5 State of the Art

Our work aligns with several significant developments within the TLS ecosystem. These advances aim to strengthen the robustness, security, and privacy of Certificate Transparency Logs, as well as decrease their operational cost.

5.1 TLS Revocation Technology Evolution

In April 2025, the CA/Browser Forum [2] voted to approve shorter certificate lifetimes. The current goal is to issue certificates with a 47-day lifetime by March 2029. This prevents a certificate from outliving its usage and remaining valid for a domain that no longer exists. Combined with the systematic renewal of private keys each time a certificate is issued, this greatly reduces the conditions under which a leak can be exploited. From an operational standpoint, this aims to limit human intervention in the renewal and distribution of certificates, as the associated administrative tasks are frequent and repetitive. In order to allow organizations to adapt and anticipate changes, the duration will be gradually reduced to 200 days in March 2026, then to 100 days in March 2027, and finally to 47 days in March 2029. This 47-day duration is not to be mistaken with Let's Encrypt 45 days lifetime that will be enforced in March 2026 [13].

This shorter certificate lifetime also comes from a hard fact: current OCSP and CRL technologies are inadequate to the modern TLS ecosystem. As a consequence, OCSP support became optional in March 2024 [1], and certificate authorities can choose to only maintain CRL services. This follows several findings against OCSP. First, requests are made over plaintext HTTP. This is a concern for privacy as the conversation can be observed and trivially replayed until the answer expires. Indeed, for performance and operational reasons, successful OCSP responses are often cached for up to 7 days, allowing an attacker to replay them. With regard to the duration of TLS negotiation, OCSP also introduces a significant (from 100 to 300 ms) and unnecessary delay, as most certificates will be deemed valid after the check. Finally, according to Let's Encrypt [11], *operating OCSP services has taken up considerable resources*. As a consequence, it stopped its OCSP responders in August 2025.

With OCSP being slowly deprecated by CA, does this mean that browsers are now required to keep CRLs up to date to properly manage certificate revocation? Not quite. Browser providers manage their own revoked certificate caches and periodically send them to our browsers.

On the one hand, Chrome uses CRLSet. This is a controversial solution [6] that consists of a reduced list of curated high-risk certificates and can be pushed quickly and multiple times a day to users. This is not a drop-in replacement of OCSP, but it helps quickly mitigate dangerous issues.

On the other hand, Mozilla developed CRLite [16], based on a dedicated data structure called Clubcard that aims to provide fast and comprehensive certificate revocation checks within a compact database. Updated every

12 hours, measurements show that Firefox users download an average of 300 KB of revocation data per day. Behind the scenes, Mozilla retrieves all CRLs and corresponding revoked certificates using Certificate Transparency logs [15]. It then creates a compact representation of the complete set of revoked certificates ready to be consumed by Firefox. Compared to OCSP and CRL, CRLite offers faster, anonymous, and comprehensive checks without requiring large data downloads.

Listing 1 shows how to retrieve and test CRLite, assuming that the Rust cargo command is available. Most of the files in the `crlite_db/` directory are delta files, each less than 200 KB in size. They correspond to the data downloaded by Firefox every day.

Listing 1: Example CRLite usage

```
# Clone the crlite GitHub repository
git clone https://github.com/mozilla/crlite
cd crlite/rust-query-crlite

# Download the latest CRLite filters
cargo run -- --update prod https://gitguardian.com
INFO - Loaded 80 CRLite filter(s), most recent was downloaded: 0
↪  hours ago
INFO - gitguardian.com Good

# Get the size of the retrieved CRLite filters
du -hs crlite_db
 21M    crlite_db

# Get the status of a certificate retrieved over HTTPS
cargo run -- https sstic.org
INFO - Loaded 80 CRLite filter(s), most recent was downloaded: 0
↪  hours ago
INFO - sstic.org Good

# Get the status of a revoked certificate
cargo run -- x509 leaked.der
INFO - Loaded 80 CRLite filter(s), most recent was downloaded: 0
↪  hours ago
INFO - leaked.der Revoked
```

Table 4 contains the status of the 2,622 certificates that were valid in September 2025, retrieved from CRLite in January 2026.

Here, Expired is the status with the highest number, as CRLite checks certificates lifetime before applying any other filters. Unsurprisingly, most of these certificates were issued by Let's Encrypt. NotEnrolled means that the corresponding CAs are not taken into account when building CRLite

filters; most of these certificates are related to an Asian governmental CA that never responded to our disclosure messages. NotCovered indicates that the corresponding certificates are not included in the CRLite database, and that browsers must perform other checks.

As of January 2026, 18.5% of the certificates have been revoked, and according to the CAs that we talked to, the corresponding private keys have been banned. Unfortunately, despite our disclosure efforts, 84 certificates still seem valid today. Note that this is only a partial answer to the question of validity, as new certificates may have been reissued with the same private key. Therefore, the number of certificates still valid (i.e. Good) reported by CRLite is unfortunately underestimated. In fact, we checked the NotCovered certificates online and found that 78 of them are still valid at the time of writing.

Status	**Public Key Hashes**	**%**
Expired	1,534	58%
NotCovered	506	19%
Revoked	484	18.5%
Good	84	3%
NotEnrolled	14	0.5%

Table 4. CRLite applied to the 2,622 leaked certificates – January 2026

5.2 Certificate Transparency Querying and Evolution

At GitGuardian, we encountered three distinct challenges. First, the storage requirements: since January 1, 2025, more than 5 billion certificates have been submitted to CT Logs, accounting for 10TB of storage for an average size of 2.3KB per certificate. Second, processing time: this depends on the CT Logs operators; however, it took us up to 7 days to recover 1 billion certificates from a single log. Third, CT Logs persistence: log entries are only useful if the certificates are still valid, and as long as they are trusted by user agents. So, log operators are not committed to keeping archives available indefinitely. However, these archives are a really valuable source of information for OSINT.

Since we originally started this research, a lot has happened in the Certificate Transparency ecosystem. Some of those advances could have helped circumvent some of the difficulties we have faced.

RFC 6962 [9] is the original specification document of the Certificate Transparency mechanism. Among other things, the RFC defines the formats of the public API that log operators should expose to allow public auditing of the web certificates. By nature, this API requires the logged certificates to be stored in a relational database that an application server needs to query to fulfill the clients' requests.

A second version of the Certificate Transparency specification was published under RFC 9162 [10]. This specification does not fundamentally change the way logs are operated, but only redesigns the protocol to be more extensible and future-proof. The main differences are in message formats and data representation. This standard did not receive any adoption [14].

RFC 6962 logs are costly to operate. The high volume of certificates stored in each log makes the storage cost high. Additionally, requesting relational databases for such a high volume is computationally intensive. To maintain a high availability service, log operators applied strict rate limiting on log queries, limiting the number of certificates that can be retrieved in a given amount of time.

For example, at the time of data collection, CloudFlare applied a rate limit of 100 requests per 10 seconds, with a batch size of 1024 certificates per request, allowing for downloading 10,000 entries per second at most. Given that, at that time, the Nimbus 2025 log contained 2 billion entries, cloning it entirely required more than two days, in the best-case scenario.

Likewise, Let's Encrypt Oak logs only allow a batch size of 256 and an empirical rate limit of 30 requests per second. Those numbers limit the number of certificates that can be downloaded to about 8,000 entries per second. Given that the Oak 2025h2 log contained 1B entries at the time of the experiment, cloning required 38 hours.

The rate limit is dependent on the log operator and is not always explicitly stated, which can make it difficult to adapt the download speed to avoid temporary bans. There are no hard requirements for operators regarding the rate limit. However, there have already been frictions around that topic in the past, including in the normal operation of Certificate Authorities and log auditing [4].

Overall, cloning the whole historical Certificate Transparency data poses a time issue. With a rough estimate of 2 days required to clone a single log, a total count of 12 different logs per year, taking into account the multiple operators, and 10 years of historical data, we evaluate the time needed to enumerate the complete dataset to be between 20 and 240

days, depending on the network bandwidth. In a typical network setup, a clone time of 2 months sounds reasonable.

In addition to the download time, the storage volume can also be a challenge. Each log now receives about 2 billion certificates per year. From our experiment, a certificate is about 3KB on average. Therefore, storing a single log requires about 6TB per year at least.

For our experiment, we did not need to store the complete certificate information. To match the private keys and work with our results, we only needed:

— the certificate fingerprint;
— the Subject Public Key Info hash;
— the Not Valid Before and Not Valid After fields;
— the certificate Common Name.

This information, stored in a simple ASCII encoding in CSV format, requires about 200B per entry on average. This reduced the overall required storage to 400GB per year per log, or around 40TB for the complete historical CT dataset. While manageable on the storage side, searching through such a large amount of data requires dedicated computational power.

For all those reasons, reconstructing the historical CT data is challenging for an individual or small corporation. This explains why we chose to turn to Google, which already stores this dataset, to get access to the data and query it efficiently.

Static CT RFC 6962 is not only a challenge for CT consumers, but also for log operators [12]. The operational cost, driven by the increase in issued certificates and the reduction of certificate lifetime, made RFC 6962 logs unsustainable for the future.

To make operating CT logs future-proof, Filippo Valsorda designed a new API format with the Sunlight project [19], which was later renamed to the Static CT API (SCT API). The main difference between RFC 6962 logs and the new Static CT API resides in how the log's Merkle Tree is stored.

With SCT, the tree data, including the leaves and certificates, are stored in flat files that can easily be stored on any storage system, including S3-compatible storage. This makes querying the data much faster, as no processing is required on the operator side, and most of the work is offloaded to the storage provider. Such files can also be efficiently cached, increasing the overall performance. Therefore, logs operated through a Static CT API do not require rate limiting.

As an example, Let's Encrypt Sycamore log distributes its tiles directly from Amazon AWS S3 buckets. The download speed is therefore limited only by the network bandwidth of the client. From our experiment, it is possible to download 200 data tiles per second, leading to 53,200 entries, from a simple 1Gb connection. This is 8 times faster than the API under RFC 6962. A much faster rate can likely be achieved by downloading from AWS infrastructure directly.

It is worth noting that the Sunlight library, which implements the SCT API, provides a client component that can be used to query logs from a static API [18]. This client provides two main ways of requesting the data:

— `func (*Client) Entries`, which enumerates all entries, while performing the integrity computation on the Merkle Tree signatures;
— `func (c *Client) UnauthenticatedTrimmedEntries`, which enumerates all entries, without any cryptographic verification, but only yields trimmed entries that do not include the complete logged certificate.

Due to the cryptographic computations, the `Entries` method is slower. In our test environment, this method could retrieve 1.2 million entries in one minute from the Sycamore log using a simple code snippet. The `ConcurrencyLimit` parameter was left to 0 for no limit.

Listing 2: Example Golang snippet to enumerate a Static CT log using Sunlight

```
func fetchEntries(ctx context.Context, client *sunlight.Client,
↪   tree tlog.Tree, start int64,
    outputChan chan<- ProcessedEntry) error {
    for index, entry := range client.Entries(ctx, tree, start) {
        select {
        case <-ctx.Done():
            return ctx.Err()
        case outputChan <- ProcessedEntry{Index: index, Entry:
↪   entry}:
        }
    }
    if err := client.Err(); err != nil {
        return fmt.Errorf("iterator error: %w", err)
    }
    return nil
}
```

Downloading the tiles directly using a simple command-line web client with 256 parallel jobs, on the contrary, allows enumerating the entries at a much higher pace of 3 million entries per minute.

Listing 3: Downloading 1000 tiles using curl

```
$ time seq 0 999 | parallel -j 256 'curl -o {}.dat \
"https://mon.sycamore.ct.letsencrypt.org/2026h1/tile/data/"$(printf
↪  "%03i" {})'
real    0m4,897s
```

The downloaded tiles can then be processed using the Sunlight library's `ReadTileLeafMaybeArchival` function to enumerate the actual certificates.

Listing 4: Reading a local data tile using Golang's Sunlight client

```
tileFile := os.Args[1]
tileData, err := os.ReadFile(tileFile)
remaining := tileData
entryCount := 0

for len(remaining) > 0 {
    logEntry, remaining, err := sunlight.ReadTileLeaf(remaining)
    //Process the logEntry
}
```

Overall, Static CT represents a useful performance improvement for clients querying logged certificates.

Log Archives One last challenge, which actually pushed us to work on this topic, is historical logs availability. Most of the operators only expose the logs containing certificates that have not expired yet. For example, as of January 2026, CloudFlare only exposes its 2025, 2026, and 2027 logs. This could lead to the historical data being lost.

Until recently, the availability of historical logs was only possible thanks to mirrors run by Google [8]. However, that model is both unsustainable and not open enough.

For this reason, Filippo Valsorda started a CT Logs archival project [20]. In essence, this project's objective is to clone CT Logs while they are available, store them using the Static CT format, and archive the resulting data on the web archive. As of January 2026, Logs data dating back to 2018 [5] is available in the archive.

For example, Let's Encrypt Oak 2025H1 archive is 680GB, split into 60 archives of 12GB each. Each of those archives contains a subtree of the whole Oak MT, including the data tile for that subset. Each smaller archive can be processed independently of the others, which makes it more efficient to process the whole log data.

Once downloaded, the archives can be interacted with using the Sunlight Static CT Golang client, which supports an `archive+file://` file handler. Doing so will perform the whole verification of the subtree using the packaged information. If this is not required for the use case, the data tiles can also be parsed directly using `ReadTileLeafMaybeArchival`, like before.

This log archive initiative will make it possible to replicate this research, even in the event of Google deprecating its historical mirrors.

6 Conclusion

This joint research between GitGuardian and Google represents the first systematic analysis mapping leaked private keys to real-world certificate usage at Internet scale. Our disclosure campaign received only 54 responses from 9% of contacted entities. The low response rate reflects not just poor contact quality, but a fundamental misunderstanding of private key risks in TLS security. Bug Bounty submissions reinforced this gap: of 9 submissions, 3 required proof-of-concept demonstrations, 3 were dismissed as informational, and only 3 were properly processed by security teams.

In addition to its immediate and obvious findings, it highlights several gaps and opportunities for improvement in the ecosystem.

First, informing the owners seems much more complicated than it should be. One potential solution would be the ability to reach out to owners by going through Certificate Authorities or by defining a dedicated X.509 extension for security contacts to complement RFC9116's `security.txt` file.

Second, keeping the same private key after renewing a certificate exacerbates the threat of leaked keys. A systematic approach to renewing private keys would be beneficial. Combined with the future 47-day duration, this would help reduce the impact of key leaks. However, some now marginal uses, such as certificate pinning in mobile applications, could be impacted by this renewal, but it is probably a good thing to stop this practice, as we did with HPKP.

Third, operated independently and similar to Certificate Transparency (CT) logs, this research started discussions to support dedicated Compromised Private Keys logs as a solution to address the problem of private key reuse. These logs would enable certificate authorities to identify leaked private keys before certificate issuance, thereby safeguarding end-user security.

Finally, CT logs are a valuable resource for research, and mapping the private keys of a leak to a certificate is probably one of the many use cases for CT logs. However, live logs are difficult to retrieve, and old ones may soon disappear. The recent advances in the Certificate Transparency ecosystem make working with CT easier. Simpler retrieval with Static CT and longer-term archiving should make replicating similar research possible in the future.

References

1. CA/Browser Forum. Ballot SC063v4: Make OCSP Optional, Require CRLs, and Incentivize Automation, 2023. https://cabforum.org/2023/07/14/ballot-sc063v4-make-ocsp-optional-require-crls-and-incentivize-automation/
2. CA/Browser Forum. Ballot SC081v3: Introduce Schedule of Reducing Validity and Data Reuse Periods, 2025. https://cabforum.org/2025/04/11/ballot-sc081v3-introduce-schedule-of-reducing-validity-and-data-reuse-periods/
3. S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. RFC 3647, RFC Editor, 2003. https://datatracker.ietf.org/doc/html/rfc3647
4. Chromium CT Policy. YETI 2022 Rate Limiting and Log Auditing, 2022. https://groups.google.com/a/chromium.org/g/ct-policy/c/AJ7msx2aWac/m/oz9kh8HVAgAJ
5. Geomys. CT Archive: Archived Logs, 2025. https://github.com/geomys/ct-archive?tab=readme-ov-file#archived-logs
6. Gibson Research Corporation. An Evaluation of the Effectiveness of Chrome's CRLSets, 2023. https://www.grc.com/revocation/crlsets.htm
7. GitGuardian. State of Secrets Sprawl Report 2026, 2026. https://www.gitguardian.com/state-of-secrets-sprawl-report-2026
8. Google. Google Certificate Transparency Mirrors, 2024. https://groups.google.com/a/chromium.org/g/ct-policy/c/IZaXMbRKtNo/m/yq2LugQcAwAJ
9. B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, RFC Editor, 2013. https://datatracker.ietf.org/doc/rfc6962/
10. B. Laurie, E. Messeri, and R. Stradling. Certificate Transparency Version 2.0. RFC 9162, RFC Editor, 2021. https://datatracker.ietf.org/doc/rfc9162/
11. Let's Encrypt. Ending OCSP Support, 2024. https://letsencrypt.org/2024/12/05/ending-ocsp
12. Let's Encrypt. End of Life for RFC 6962 Logs, 2025. https://letsencrypt.org/
13. Let's Encrypt. From 90-Day to 45-Day Certificate Lifetimes, 2025. https://letsencrypt.org/2025/12/02/from-90-to-45
14. Let's Encrypt Community. Certificate Transparency Versions and Status of CTv2, 2024. https://community.letsencrypt.org/t/certificate-transparency-versions-and-status-of-ctv2/218492/3
15. Mozilla. CRLite Part 2: End-to-End Design, 2020. https://blog.mozilla.org/security/2020/01/09/crlite-part-2-end-to-end-design/

16. Mozilla. CRLite: Fast, Private and Comprehensive Certificate Revocation Checking in Firefox, 2025. https://hacks.mozilla.org/2025/08/crlite-fast-private-and-comprehensive-certificate-revocation-checking-in-firefox/
17. Salesloft. Drift/Salesforce Security Notification, 2025. https://trust.salesloft.com/?uid=Drift%2FSalesforce+Security+Notification
18. Filippo Valsorda. Go Sunlight Client, 2024. https://pkg.go.dev/filippo.io/sunlight#Client
19. Filippo Valsorda. Sunlight: A CT Log Implementation, 2024. https://sunlight.dev/
20. Filippo Valsorda. Archiving Certificate Transparency Logs, 2025. https://groups.google.com/a/chromium.org/g/ct-policy/c/Y25hCTrCjDo