Private Key Leaks in the Wild

from PTS to RWC, and back to PTS

Gaetan FERRY - Guillaume VALADON

$ whoami

Guillaume

Cybersecurity Researcher

editor-in-chief of the **MISC magazine**

Scapy maintainer

previously at **Quarkslab,** ANSSI…

Gaetan

Cybersecurity Researcher

former researcher **@Sonar**

Synacktiv red teamer for 7 years

01

Private Key Leaks & Certificate Transparency

What 945,000 Leaked Keys Reveal

Leaks Everywhere You Look

Public secrets leaks an **underestimated problem**

29 millions

secrets leaked on GitHub in 2025

⬆️ 34% increase from 2024

430,000

private keys included

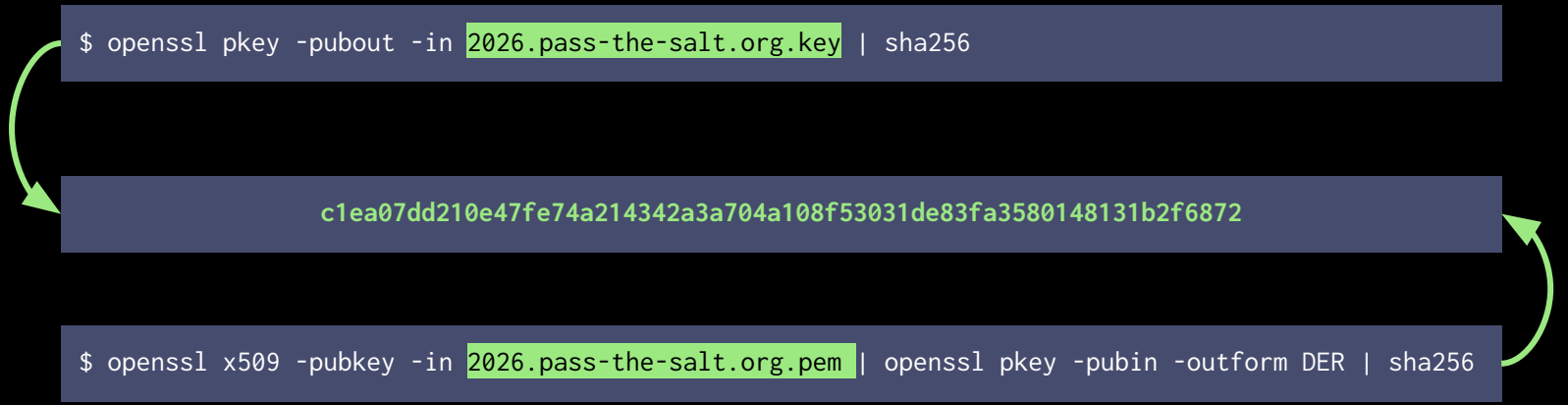
The Main Concept

A private key and its certificate share the same public part

- Hash the *SubjectPublicKeyInfo* from each side

Matching hashes link a private key to its certificate

- The secret part of the key is never needed

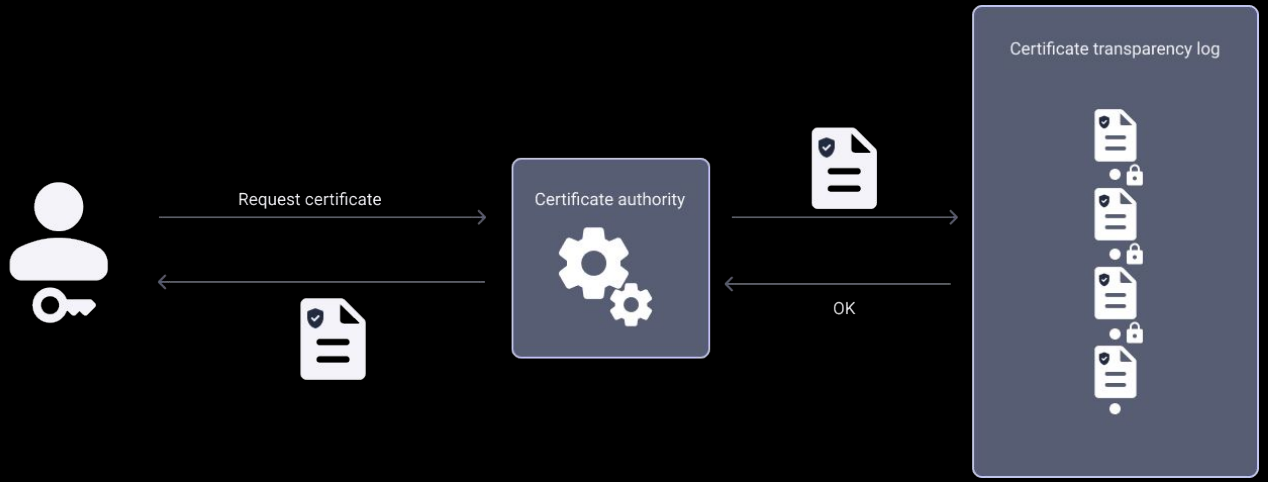


Certificate Transparency

The good

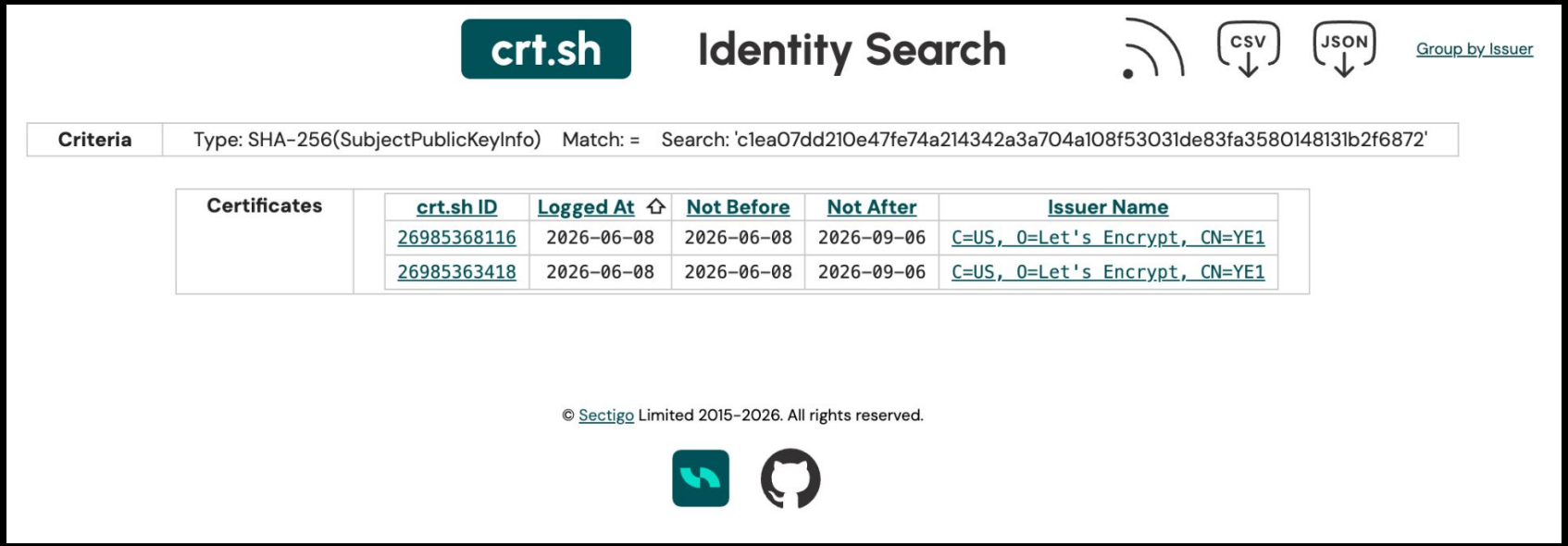
Core mechanism of the X.509 infrastructure since 2015

- CA sends certificates to a CT Log
- Logs are public, append only & tamper-proof
- Browsers check a proof of inclusion



Certificate Transparency

https://crt.sh



https://crt.sh/?spkisha256=c1ea07dd210e47fe74a214342a3a704a108f53031de83fa3580148131b2f6872

Certificate Transparency

Manual queries

```
$ curl 'https://ct.googleapis.com/logs/us1/argon2026h1/ct/v1/get-entries?start=17605&end=17605' | \
jq -r '.entries[0].extra_data' | base64 -d > extra_data.bin
```

```
$ binwalk -e -C extracted/ --dd=.*:crt extra_data.bin
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------------------------------
3             0x3             Certificate in DER format (x509 v3), header length: 4, sequence length: 1265
1278          0x4FE           Certificate in DER format (x509 v3), header length: 4, sequence length: 1200
2485          0x9B5           Certificate in DER format (x509 v3), header length: 4, sequence length: 863
```

```
$ openssl x509 -pubkey -in extracted/9B5.crt | openssl pkey -pubin -outform DER | sha256
706bb1017c855c59169bad5c1781cf597f12d2cad2f63d1a4aa37493800ffb80
```

Certificate Transparency

The collaboration

Perfect public lookup table but...

- Too big: ~38 TB of certificates for 2025 alone
- Too slow: rate limiting by logs' operators
- Too late: logs are put offline when certificates expire

At Pass The Salt 2025, we discussed about **our collaboration**:

- GitGuardian provides private key hashes from Public Monitoring
- Google performs the certificates lookup using their internal BigQuery

What We Found

42,690
keys attributed

› 4.5% of the total
› 77% from GitHub

139,767
unique certificates

› 36,978 distinct subjects

2,622
valid certificates

› in September 2025
› 35% publicly reachable

ℹ️ **Revocation is barely used**

› CRL: 24 invalid certificates; 1 with key_compromise
› OCSP: 56 invalid certificates; 2 with keyCompromise

Who Is Exposed and For How Long?

20% of keys
exposed for 2+ years

17% of certificates
valid when key is leaked

snap.cso.att.com
asapdoss.ciostage.accenture.com
***.amlarc.microsoft.com**
ensc-lille.fr
***.rireetchansons.fr**
zen.pole-emploi.fr

Examples from old certificates

ℹ️ governments, Fortune 500 companies, even a **Certificate Authority**

Responsible Disclosures

600
organizations

4,300
emails sent

9
bug bounties

~9%
overall response rate

› 36% companies
› 10% governments

9
CAs

2,193
certificates revoked

› almost immediately
› inline with CP/CPS

02
Do This At Home

Because, yes we can!

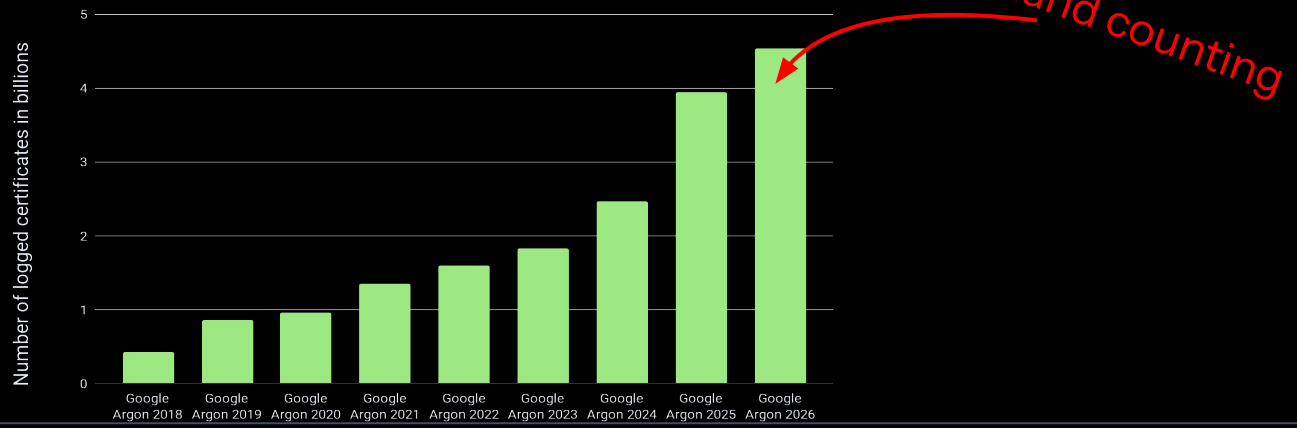
One Year Ago: Challenges Everywhere

RFC 6962 logs are costly to operate. This brings some challenges:

- Operators rate-limit clients request to save availability
- Old logs are shutdown as soon as possible to cut on costs
- Archival is not easily done

› Google operated historical logs mirrors, but that is costly too → shutdown planned

› All this is amplified by the huge amount of certificates



Now: Things changed, fast

Over the course of the year, CT ecosystem evolved to face its challenges

- RFC 6962 logs are too costly? → Static-CT
- Archival is an issue? → ct-archive

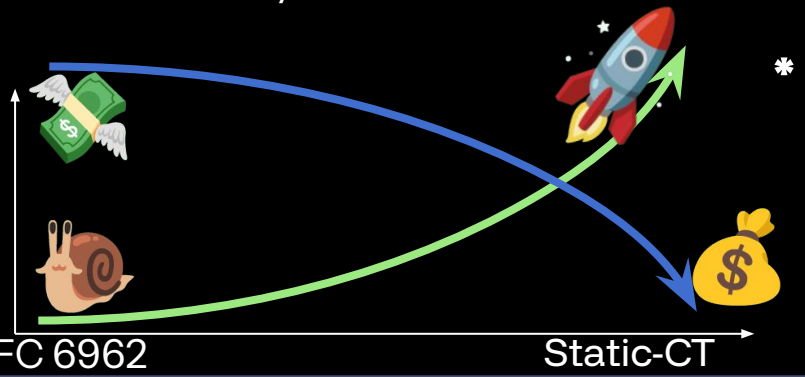
Static-CT: Making CT manageable

Static CT is a complete rewrite of the RFC 6962 API

- Development started in early 2024
- First used by Let's Encrypt for 2024h1 (testing)
- Let's Encrypt dropped RFC 6962 on November 30, 2025

Main ideas

- CT Merkle Tree data stored in plain files, data (certificates) sharded
- No need for a DBMS or application processing
- Monitoring offloaded to a file system (S3 ?)



* From laboratory measurements

Static-CT: Use It As a Client

Working with Certificate Transparency log, for our use case, requires:

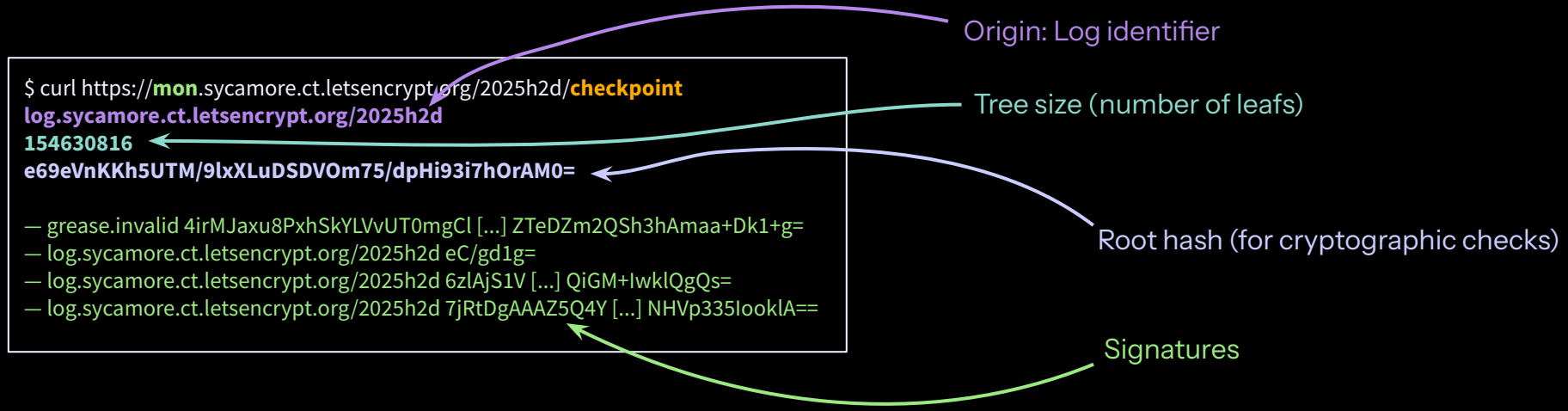
- Querying the MT root → Get the number of certs to download
- Enumerating the certificates

› We don't **need** the rest of the structure, we are not interested in the data authenticity

RFC 6962	Static-CT
`https://my.log/some/path/mylog2026h1/ct/v1/get-sth`	`https://mon.my.log/mylog2026h1/checkpoint`
`https://my.log/some/path/mylog2026h1/ct/v1/get-entries?start=13&end=37`	`https://mon.my.log/mylog2026h1/tile/data/x001/337`

Static-CT: Use It As a Client

The *checkpoint* endpoint is pretty straightforward



› We only really need the tree size. Other information are for **real** auditors.

Static-CT: Use It As a Client

The *tile/data* endpoints are where we collect the actual certificates

```
$ curl https://mon.sycamore.ct.letsencrypt.org/2025h2d/tile/data/x001/337
```

```
00000000: 0000 0199 2c27 2fa0 0000 0003 df30 8203  ....,'/......0..
00000010: db30 8203 80a0 0302 0102 0211 00bb b486  .0..............
00000020: 4894 ff19 5a0e 1bb1 311c d10b d430 0a06  H...Z...1....0..
00000030: 082a 8648 ce3d 0403 0230 3b31 0b30 0906  .*.H.=...0;1.0..
00000040: 0355 0406 1302 5553 311e 301c 0603 5504  .U....US1.0...U.
00000050: 0a13 1547 6f6f 676c 6520 5472 7573 7420  ...Google Trust
...
000003e0: 3af7 d27e 2527 8de1 ba5a 3742 0008 0000  :..~%'...Z7B....
000003f0: 0500 0004 7100 0060 1dfc 1605 fbad 358d  ....q..`......5.
00000400: 8bc8 44f7 6d15 203f ac9c a5c1 a79f d485  ..D.m. ?........
00000410: 7ffa f286 4fbe bf96 76b2 7b80 a580 27dc  ....O...v.{...'.
...
00000450: f3c1 df6c d433 1c99 0000 0199 2c27 2fa0  ...l.3......,'/.
00000460: 0000 0003 fb30 8203 f730 8203 9ca0 0302  .....0...0......
...
```

This retrieves data tiles, shards of 256 entries

Timestamp

Binary blob

Type (x509 / precertificate)

Length

Certificate (x509 here)

Extensions

Fingerprints

Encoding: index of the certificate (decimal, split in parts of 3 chars, x prefix after the first part)

Binwalk anyone ?

Static-CT: One Doesn't Just curl

Sunlight is developed in Golang (https://pkg.go.dev/filippo.io/sunlight)

› The package includes client components

```
package sunlight

// Client is a Certificate Transparency log client that fetches and
// authenticates tiles according to c2sp.org/static-ct-api, and exposes log
// entries as a Go iterator.
type Client struct { [...] }

func (c *Client) Entries(ctx context.Context, tree tlog.Tree, start int64) iter.Seq2[int64, *LogEntry] {

func (c *Client) UnauthenticatedTrimmedEntries(ctx context.Context, start, end int64) iter.Seq2[int64, *TrimmedEntry] {
```

Enumerate through all entries and returns them as an iterator.

Actual client to instantiate

Requires a verified Tree

Does not require a verified Tree but only returns trimmed entries: timestamp, subject, SAN

Useful if you are only interested in names.

Static-CT: One Doesn't Just curl

Simple example (`err` handling skipped)

```
func main() {
   block, _ := pem.Decode([]byte(`-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE4i7AmqGoGHsorn/eyclTMjrAnM0J
UUbyGJUxXqq1AjQ4qBC77wXkWt7s/HA8An2vrEBKIGQzqTjV8QIHrmpd4w==
-----END PUBLIC KEY-----))
   key, err := x509.ParsePKIXPublicKey(block.Bytes)

   client, err := sunlight.NewClient(&sunlight.ClientConfig{
       MonitoringPrefix: "https://mon.sycamore.ct.letsencrypt.org/2027h1/",
       PublicKey:        key,
       UserAgent:        "ExampleClient (changeme@example.com, +https://example.com)"
   })

   var start int64
   for {
       checkpoint, _, err := client.Checkpoint(context.TODO())
       for i, entry := range client.Entries(context.TODO(), checkpoint.Tree, start) {
           fmt.Printf("%d: %d %d %x\n", i, entry.LeafIndex, entry.Timestamp, entry.IssuerKeyHash)
           start = i + 1
       }
   }
}
```

🚨

Performs unnecessary crypto checks

- Slows down download

UnauthenticatedTrimmedEntries could help, but does not retrieve certificate data.

https://pkg.go.dev/filippo.io/sunlight#example-Client.Entries

Static-CT: Base Functions

Sunlight does not directly implement tile downloading, parsing, etc
Instead relies on Torchwood (https://pkg.go.dev/filippo.io/torchwood)
Torchwood exposes lower level functions to interact with Static-CT logs

```
package torchwood

func (c *Client) Entries(ctx context.Context, tree tlog.Tree, start int64) iter.Seq2[int64, []byte] {

func (f *TileFetcher) ReadTiles(ctx context.Context, tiles []tlog.Tile) (data [][]byte, err error) {

func (f *TileFetcher) ReadEndpoint(ctx context.Context, path string) (data []byte, err error) {

func ReadTileEntry(tile []byte) (entry, rest []byte, err error) {
```

Most functions available through a Sunlight client via: `func (c *Client) TileReader() torchwood.TileReader`

Static-CT: Base Functions

Simple example (`err` handling skipped)

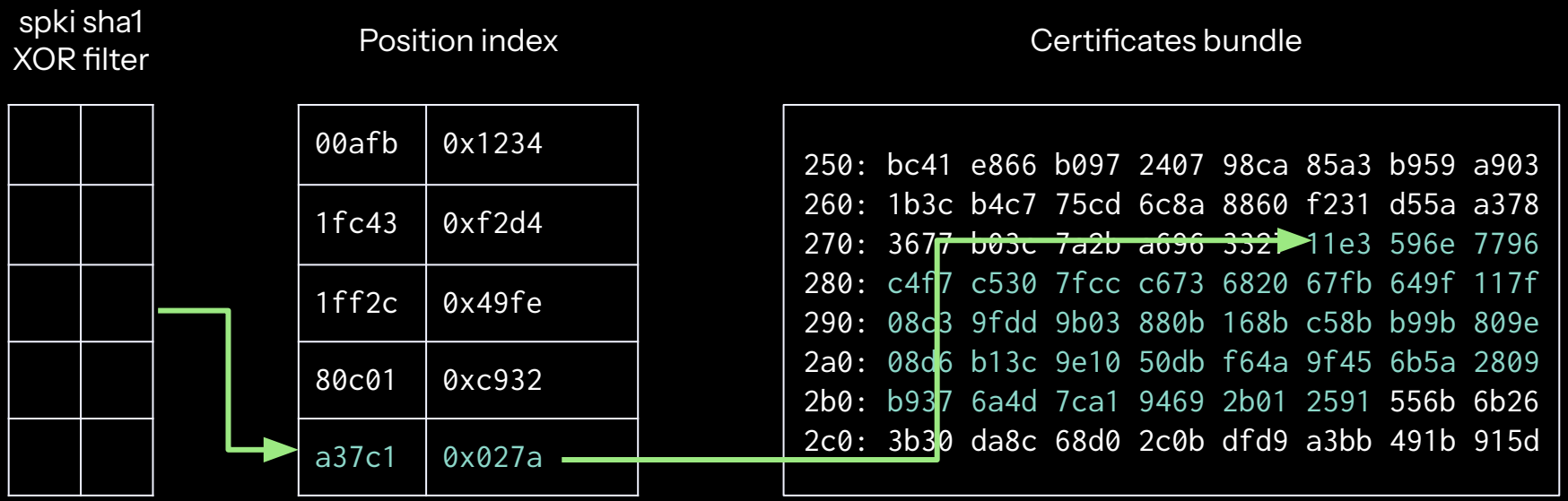
```
url := "https://mon.sycamore.ct.letsencrypt.org/2027h2/"

fetcher := torchwood.NewTileFetcher(*url, torchwood.WithTilePath(sunlight.TilePath))
ctx := context.Background()

cpData, err := fetcher.ReadEndpoint(ctx, "checkpoint")
cp, err := parseCheckpoint(cpData)
fmt.Printf("tree size: %d\n", cp.N)

tiles := fetcher.ReadTiles(ctx, []tlog.Tile{{H: torchwood.TileHeight, L: -1, N: 0, W: torchwood.TileWidth}})
tile := tiles[0]
for i := 0; i < 10; i++ {
   var e *sunlight.LogEntry
   e, tile, err = sunlight.ReadTileLeaf(tile)
   printEntry(int64(i), e)
}
```

Static-CT: Demo



https://github.com/gaetan-gg/cert-hunter

Static CT: The Limits

Static CT greatly improve the download speed of certificates from logs.
But it still has some limitations:

- The volume to download is still very high
- Working with such a volume is challenging
- Static-CT is not yet used by every operators

A list of all current CT logs is available at: https://www.gstatic.com/ct/log_list/v3/log_list.json
It lists RFC 6962 and static-ct logs.

› Currently 5 operators out of 8 have a tiled log available. Cloudflare, Digicert & Sectigo missing.

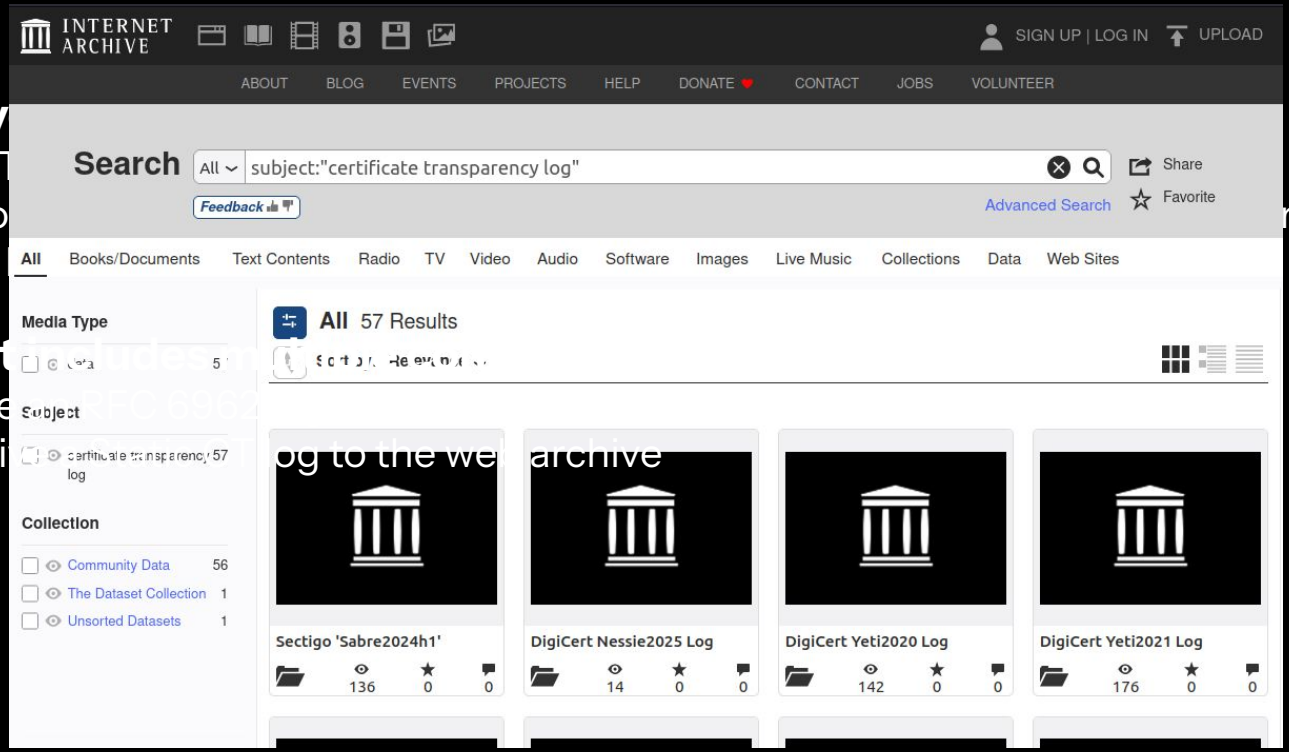
CT Archive: Save Historical Data

Logs Archiv

- Clone CT
- Open So
- Years of

The project

- To clone
- To archi



CT Archive: Archival format

Collection of zip archives, each of which stores a subtree of the log's MT

CT Archive: Using CT Archives

Good news is the format is exactly the same as the Static CT API (that's the point)
The same code can parse the archives data and the API responses

```
package sunlight

type ClientConfig struct {
[...]
   // If it has schema "archive+file://", Client will read tiles from a set of
   // [archival zip files] in the specified directory.
[...]
   MonitoringPrefix string

func ReadTileLeafMaybeArchival(tile []byte) (e *LogEntry, rest []byte, err error) {
```

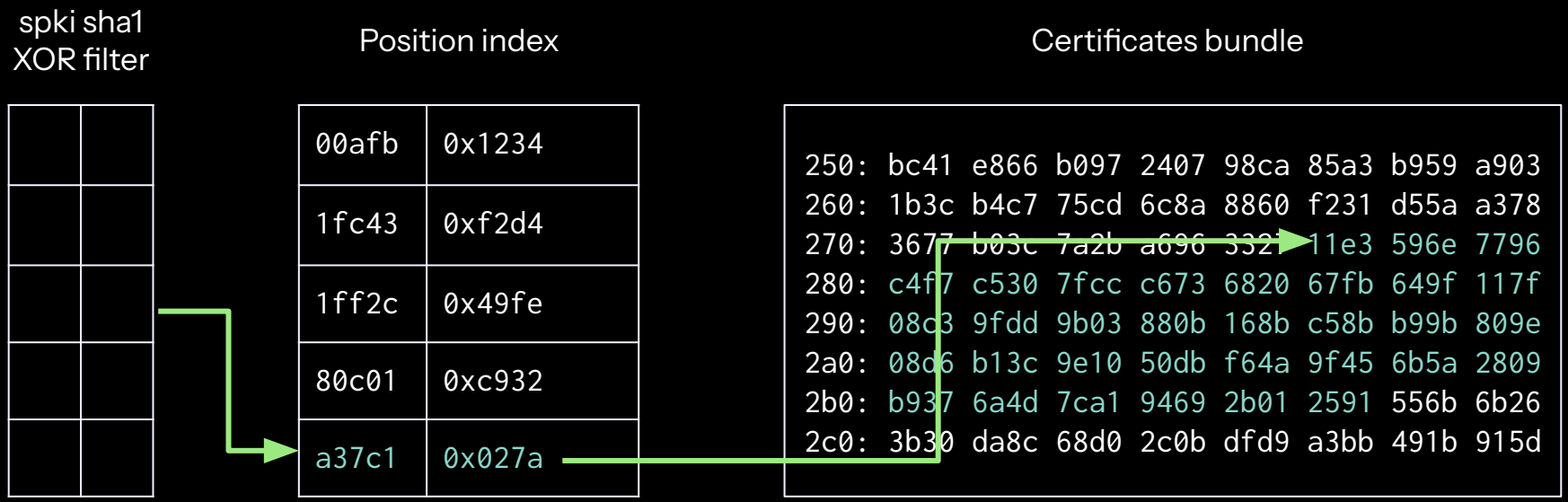
CT Archive: Using CT Archives

Minimal example

```
tileData, err := os.ReadFile(tileFile)
remaining := tileData
entryCount := 0
for len(remaining) > 0 {
    // Read the next log entry from the tile
    logEntry, rest, err := sunlight.ReadTileLeafMaybeArchival(remaining)
    cert, err = x509.ParseCertificate(logEntry.PreCertificate)
    fmt.Printf("%s\n", cert.Subject.String())

    remaining = rest
    entryCount++
}
```

CT Archive: Demo



https://github.com/gaetan-gg/cert-hunter

And Now, What's For The Future

Using Static CT and CT archives make it easier to ingest / analyze certificates at scale

But that won't be forever

- Certificate's lifetime is decreasing (200 days now, 47 days as a target)
- Number of logged certificates is going to explode
- So will the download time / storage volume / computational requirements

Ecosystem moving to Merkle-Tree certificates: will that help us? Likely yes:

- New *Issuance Logs* stores less information → No public keys, only hashes
- Entries are much smaller, reduced download time
- No need to compute key hashes, less computing

This is only speculation at the moment

https://github.com/gaetan-gg/cert-hunter

Thank you

Question time 🔥

Gaetan FERRY - Guillaume VALADON

GitGuardian