New issue

# crypto/tls: certificate chains aren't re-checked on resumption #77217

Open

Labels LibraryProposal NeedsDecision

rolandshoemaker opened on Jan 16 Member

As part of the report which led to #77113, **@rbqvq**, also noted that we don't reverify certificate chains during resumption. This has been the default behavior of Go since 1.21 (see #31641 for the discussion of this at the time), when we resolved our previous inconsistent behavior and landed on not reverifying certificates during resumption (and as such documented `VerifyPeerCertificates` was not called for resumed connections).

The issue as reported was that if the underlying trust configuration changed after an initial connection, this could cause session tickets to be used to resume a connection based on previous information that would otherwise be rejected if used during a fresh handshake. It is our belief that this does not represent a significant security issue, and that this behavior matches the behavior of a number of other implementations (namely BoringSSL and OpenSSL).

Firstly, it is documented to be a API violation to mutate a `tls.Config` once it has been passed to a TLS function ("After [a tls.Config] has been passed to a TLS function it must not be modified"), so mutating any of the `Config` fields which define trust (`Config.Roots`, `Config.ClientCAs`, `Config.ClientAuth`, `Config.VerifyConnection`, etc) while a `Config` is in-use is already a misuse of crypto/tls.

There was a related security issue with respect to using `Config.Clone` to clone a `Config` for mutation that retained the automatically generated session ticket keys for the cloned config (see #77113) that has since been resolved.

We also provide an advanced API, `Config.SetSessionTicketKeys`, which allows explicit user control over the session ticket keys, disabling our automatic generation and rotation of said keys. We believe making changes to the trust configuration of a `Config` but retaining the same keys is a misuse of this API (i.e. by not calling `Config.SetSessionTicketKeys` with a new set of keys after mutating the relevant trust configurations fields). Additionally, using the same set of session ticket keys across backends that are explicitly configured to have different trust configurations is a misuse of the API. I believe our documentation here is somewhat lacking and should be improved to detail that this is a rather sharp edged API, and should be used carefully with respect to these properties.

Skipping certificate chain verification is one of the main wins we get from session resumption, but it is worth considering if there is anything we can do to improve the security of resumption without reducing the performance. @**rbqvq** had some ideas here which are worth investigating.

cc @golang/security @**FiloSottile**

👍 1

**rolandshoemaker** added **NeedsDecision** on Jan 16

gabyhelp on Jan 17

**Related Issues**

- crypto/tls: TLS session resumption re-verifies the client's certificate chain #31641 (closed)
- crypto/tls: Config.Clone copies automatically generated session ticket keys, session resumption does not account for the expiration of full certificate chain (CVE-2025-68121) #77113 (closed)
- crypto/tls: OCSP and SCTs are dropped in resumed connections [freeze exception] #39075 (closed)
- crypto/tls: VerifyConnection is called twice by tls 1.3 servers if connection is resumed #39012 (closed)
- crypto/tls: GetCertificate called on resumed sessions #25352
- crypto/tls: avoid linkability across sessions by not reusing session tickets #60505

**Related Code Changes**

- crypto/tls: don't reverify but check certificate expiration on resumption
- [release-branch.go1.26] crypto/tls: don't copy auto-rotated session ticket keys in Config.Clone
- [release-branch.go1.24] crypto/tls: don't copy auto-rotated session ticket keys in Config.Clone

(Emoji vote if this was helpful or unhelpful; more detailed feedback welcome in this discussion.)

**gabyhelp** added **LibraryProposal** on Jan 17

rbqvq on Jan 17 · edited by rbqvq Edits •••

First, I need to point out the issues in #77113:

The vulnerability description doesn't explicitly state that sharing SessionTicketKeys can lead to session recovery between different trust configurations (whether it's config.Clone or SetSessionTicketKeys).

Most downstream applications are unaware that this could lead to authentication bypass across ClientCAs.

Furthermore, the related patch content and description are incorrect; `peerCertificates` does not include the RootCA, and it doesn't correctly check the RootCA's expiration.

The correct approach is to use `verifiedChains`.

Additionally, the related checks don't consider clock rollback; it lacks a NotBefore check.

Breaking config.Clone causes QUIC-Go's TLS Resumption (including 0-RTT) to fail, resulting in massive performance regressions.

---

In addition, there are additional risk factors:

Most people are unaware of the differences in session recovery (including Teleport) between TLS 1.0-1.2 and TLS 1.3:

- In TLS 1.0-1.2

No master key is renegotiated, therefore the ticket validity period is fixed at 7 days.

- In TLS 1.3

The master key is renegotiated, and a new ticket with a 7-day validity period is issued (forward security).

> That is, the window for authentication bypass can extend until the endpoint certificate expires (including intermediate CAs after #71773).

The ticket validity period is not linked to identity freshness, but rather to key freshness.

With public CAs no longer issuing new client certificates, private PKIs typically have validity periods of several years or even longer (they are not subject to CA/B restrictions).

> For example, Cloudflare's mTLS certificates can be valid for 1-15 years (10 years by default).

But Roland said:

> Additionally, we believe the ability to indefinitely refresh a session up to the point of the client certificate expiration is the expected behavior of TLS 1.3 servers, assuming a sequence of chained session keys is used. As a side note, connections can often remain open for extremely long periods, which likely allows a user to bypass a change in trusted roots, or cross a certificate expiration boundary (though probably not indefinitely).

Downstream applications, such as Teleport #62896, bypass the restrictions of config.Clone by using a dangerous workaround involving manually shared keys.

Teleport said:

> I was not aware of the difference in behavior between tickets in TLS 1.0-1.2 and TLS 1.3 (and QUIC, I assume?) but if that's all I think we are ok and I have made a note about tweaking our approach (i.e. resetting the session ticket keys if the set of trusted client certs changes in a way that can invalidate any previously valid certs) if the nature of that specific QUIC server changes.

---

I do not believe that TLS resumption and persistent connections are equivalent in nature.

The migration of TCP connections to QUIC in modern network environments demonstrates the fragility of TCP.

Furthermore, cluster maintenance and traffic redirection are common, which can interrupt existing connections and cause persistent connections to fail.

> See the Re-route status at https://www.cloudflarestatus.com/

---

Regarding #31641, normally Go teams need to go through a proposal process and leave a GODEBUG when making related changes.

I don't know why the team suddenly picked up this issue and merged it after it had been dormant for almost a year.

There are fewer than 30 discussions under #31641, and even fewer people know about the changes.

In a private email discussion, Roland said the following:

> Before detailing our rationale, note that the discussion of ClientAuth levels in #31641 referred to when we previously always decided to do reverification based on the ClientAuth level, not doing reverification if the ClientAuth level changed. Unfortunately, a lot of the additional context for our rationale in making the change we landed on was not captured in that issue, nor in the CL commit description, making that conversation slightly hard to follow to our final decision.

And

> When we changed the behavior of crypto/tls in 1.21 (https://go.dev/issue/31641), the change did not require the proposal process. It did not introduce a new API. The behavior change resolved undocumented internal inconsistencies in our implementation. Perhaps if we had done this today, we would have made a different decision, but this has now been documented behavior for almost 5 major Go versions.

In the Go 1.21 Release Note, the release team, as always, categorized the change as a "minor library change."

I believe most users are unaware of the potential security risks it may pose.

Most Gophers do not consider "minor library changes" to affect security.

This seems to violate the principle of transparency in open-source governance, ~~and is even somewhat similar to the XZ Utils Backdoor.~~

---

And, in the Go 16th Anniversary Blog

we discussed FIPS 140-3 compliance and the Secure-by-default principle:

> The Go standard library has continued to evolve to be safe by default and safe by design.

I'm not sure if this issue will affect FIPS 140-3 compliance, although the crypto/tls documentation suggests it's not fully compliant.

Roland email reply:

> We specifically considered implementing minimally invasive fixes that would maintain compatibility, but we concluded these changes could not reasonably be included in a minor security release. In particular, we take a relatively conservative approach to what is treated as a PRIVATE track security patch (per https://go.dev/security/policy), because we don't want users to be hesitant to pick up security releases because of possible regressions. Since imposing a lifetime limit on TLS 1.3 tickets, or encoding further trust configuration information in the ticket, would necessitate structural changes to tickets. This would immediately invalidate. all existing session tickets for upgraded users. This was considered too likely to cause performance regressions, making large users hesitant to upgrade.
> That said, as we've previously stated, we are happy to publicly discuss making these changes in a major Go version, which by design can make less conservative changes. I'm happy to open an issue for both points you've raised, or I'm happy for you to do so.

as well as

> Our policy for what to include in security releases is, as I've previously stated, quite conservative. We try to only include PRIVATE track patches which violate committed security policies, and the disclosure of which would create a security issue with no reasonable workaround, which is why we prioritized the Config.Clone fix. We are also committed to not making API changes, or would introduce unexpected behavior in violation of the documented properties of existing APIs when possible.
> Your patch, while I think a reasonable starting point for improving our implementation, introduces user-visible API changes. When we resolved https://go.dev/issue/31641 we explicitly documented in the The tls.Config API that VerifyPeerCertificates is not called on resumed connections. We cannot make a change to that behavior in a minor release. This, unlike https://go.dev/issue/31641, would need to go through the proposal process and could only be included in a major Go release (such as 1.27).

I submitted a patch to the Golang security team to restore trust chain verification. However, they did not adopt it.

If we have verified it before, it compares the validity of all certificates and verifies whether the root certificate is still in our RootCA.

This is just a SHA224 and O(1) performance in the session resumption process and solves the problem of cross-trust domain resumption.

It also solves the connection interruption caused by VerifyConnection failing on session resumption.

The new behavior is that if the resumption does not meet the requirements, we will degrade to a full handshake without causing a loss of availability.

I will subsequently submit a CL (I previous submit to them) that solves the compatibility issues before and after Go 1.21 while maintaining performance and security.

> I think calling VerifyPeerCertificate once more is acceptable. It won't cause any problems; if it fails, it will simply fall back to a full handshake.

I'm not satisfied with the fix.

The Golang security team is concerned about performance regression before.

But now, the current patch will cause large-scale QUIC-Go session resumption failures, and it's ineffective for advanced users manually setting session ticket keys, such as in clusters.

https://github.com/quic-go/quic-go/blob/master/internal/handshake/tls_config.go

---

Furthermore, I have concerns about a series of changes in Go 1.21 commits:

In the QUIC CL, `return` is missing after `sendAlert`

- `2cac7e8` #diff-59f954d41ee2bc65255172f3fb4c24d0a8d8f99f1d153a3078961ccf6484d382R396

> For example, `src/crypto/tls/handshake_server_tls13.go`, Line 575

In my previous report, a previous QUIC CL reviewer (Filippo) commented

> I don't actually get why this check is here, since we are changing write secrets, but we should be sure we don't need it / know why it was there.

And the Roland said

> I think it depends on how the higher-level QUIC implementation handled alerts. In the experimental x/net QUIC impl, seeing an alert is fatal, so the return was not strictly necessary because the handshake would immediately terminate anyway and the connection would be torn down.
> That said, I think we are all in agreement that this didn't really make sense to begin with, so removing it is fine.

My said:

> This is true for received alerts.
>
> But for the sent alert, after handshakeFn exits at handshakeContext, if the c.handshakeErr != nil will report a wrapped error (c.out.err) to the quic layer.
>
> That is, if the return is missing here, the c.handshakeErr is not set and no handshake error is thrown
>
> I think you can force an error to be thrown here without return to see if quic-go works.
>
> However, this code has been removed, and there seems to be no need to worry about it.
>
> My suggestion: For future work, all sendAlert should be followed by a return to avoid this "empty check" problem.

- https://go-review.googlesource.com/c/go/+/493655
- https://go-review.googlesource.com/c/go/+/724120/3/src/crypto/tls/handshake_client_tls13.go#b850

The reviewer doesn't understand the code they personally approved, and the maintainer hasn't checked the call chains of the relevant functions.

In #31641

- Missing Proposal (possibly unnecessary at the time)
- Missing GODEBUG recovery behavior
- Silent since the merge a year ago, with no new discussion
- Release Note categorized as "Minor library change"

And Roland said

> Unfortunately a lot of the additional context for our rationale in making the change we landed on was not captured in that issue, nor in the CL commit description, making that conversation slightly hard to follow to our final decision.

These two cases indicate that Go 1.21 exhibits a continuous, systematic review and transparency failure.

Our Peer Review, designed to ensure supply chain security, appears to have failed.

I need **@rsc** **@aclements** **@ianlancetaylor** to be aware of this matter.

I can provide full email PDF for this.

---

Furthermore, Go 1.24 has one last release, and Go 1.26 is coming soon.

As a native FIPS 140-3 milestone, I think we should reach a conclusion as soon as possible before the merge window closes to avoid leaving any potential problems.

I suggest partially reversing the erroneous decision in #31641 and reverting the changes in #71773. I will provide a new CL for this.

The behavior of `config.Clone` should be restored.

The call to `VerifyPeerCertificate` can be restored in Go 1.27 (but I still recommend immediately reversing the relevant changes).

👍 1

rbqvq on Jan 17

CC

- Go-OpenSSL-FIPS **@derekparker**
- QUIC-Go **@marten-seemann**

**rbqvq** mentioned this on Jan 17

update core binaries to go 1.25.6 for CVE fixes kubernetes/kubernetes#136258

**rbqvq** added 4 commits that reference this issue on Jan 17

`crypto/tls: add minimal certificate verification on resumption` ae4f38a

`crypto/tls: add minimal certificate verification on resumption` b68af5e

`crypto/tls: add minimal certificate verification on resumption` 5ff97af

`crypto/tls: add minimal certificate verification on resumption` a85d69f

**rbqvq** mentioned this on Jan 17

crypto/tls: add minimal certificate verification on resumption #77220

rbqvq on Jan 17

Technical details of the new CL

```
// If we have already verified the certificate chain.
    if len(verifiedChains) > 0 {
        // Check the validity period of the full-chain certificate (exclude the RootCA).
        for _, c := range certs {
            if opts.CurrentTime.Before(c.NotBefore) {
                return nil, x509.CertificateInvalidError{
                    Cert:   c,
                    Reason: x509.Expired,
                    Detail: fmt.Sprintf("current time %s is before %s", opts.CurrentTime.Form
                }
            }

            if opts.CurrentTime.After(c.NotAfter) {
                return nil, x509.CertificateInvalidError{
                    Cert:   c,
                    Reason: x509.Expired,
                    Detail: fmt.Sprintf("current time %s is after %s", opts.CurrentTime.Forma
                }
            }
        }

        reverifiedChains := make([][]*x509.Certificate, 0, len(verifiedChains))
        for _, verifiedChain := range verifiedChains {
            // If verifiedChain are empty, Skip it
            if len(verifiedChain) == 0 {
                continue
            }

            // Check the root certificate's validity period and whether it is still trusted b
            rootCA := verifiedChain[len(verifiedChain)-1]
            _, err := rootCA.Verify(opts)
            if err != nil {
                continue
            }

            reverifiedChains = append(reverifiedChains, verifiedChain)
        }

        // If we have the valid chains, use it directly.
        if len(reverifiedChains) > 0 {
            return fipsAllowedChains(reverifiedChains)
        }
    }
```

Regarding performance, `rootCA.Verify(opts)` is a SHA-224 hash lookup in a CertPool is an O(1) operation.

- https://github.com/golang/go/blob/master/src/crypto/x509/verify.go#L593

- https://github.com/golang/go/blob/master/src/crypto/x509/cert_pool.go#L176

---

I forked the official stack and added many extra features to demonstrate my technical prowess with it.

Welcome to use it :)

This stack, as a kTLS implementation of MinIO/warp, should have been extensively tested.

https://gitlab.com/go-extension/tls

gopherbot on Jan 17 Contributor •••

Change https://go.dev/cl/737200 mentions this issue: `crypto/tls: add minimal certificate verification on resumption`

rbqvq on Jan 17 •••

While the behavior of `VerifyPeerCertificates` during resumption has been documented since Go 1.21, documentation of a behavior does not mitigate the inherent security risk of a Trust Anchor Bypass.

Security regressions should not be grandfathered in simply because they have persisted through major versions.

The goal of the Go security team should be "Secure-by-Default," not "Inconsistent-by-Legacy."

I think consistency of behavior should be to become safer together, not to become less safe together.

In addition, the risk of the client and the server is not equal, the session is stateful for the client and stateless for the server.

marten-seemann on Jan 17 Contributor •••

I don’t think that certificate chains should be re-verified during resumption. Quite the opposite: resumption is designed such that it’s not necessary to retransmit the certificate chains in the first place.

It is a rather annoying quirk that crypto/tls saves the certificate chains in the session ticket, as it dramatically increases the size of the session ticket, which basically erases most of the benefits of session resumption (at least when looking at bytes transferred).

As far as I understand, this is done purely to fill out the field in the ConnectionState.

If we're willing to touch this logic, I'd strongly advocate for NOT storing the certificate chain at all. Again, this is how TLS session resumption is supposed to work: you trust your prior self to have verified the certificate chain. The correct way to handle changes in your trust configuration is to reject session resumption in that case.

👍 1

**44 remaining items**

Load more



## Metadata

**Assignees**

No one assigned

**Labels**

LibraryProposal NeedsDecision

**Type**

No type

**Projects**

No projects

**Milestone**

No milestone

**Relationships**

None yet

**Development**

**crypto/tls: add minimal certificate verification on resumption**
golang/go

**Participants**

 +4