Session Resumption and TiiTLS tickets

1-RTT refers to a session resumption without early data. 0-RTT refers to a resumption with early data.

Lets Talk Tickets

Resumption tickets are a big part of TLS1.3. After the effort of making a full TLS1.3 connection to a website, it is quite common for a user to want to reconnect to the same site a few minutes later. Ideally without going through the same full process again. But at the same time a busy server does not want to be burdened with maintaining "state" for each client that may have connected to it over the last hour (and there could be millions of them).

So the trick is that the server issues a "ticket" to the client containing enough information for both the client and the server to resume the connection. Part of this ticket is encrypted by the server using an encryption key known only to the server.

So, in brief, sometime after a full handshake and during its initial connection, the server encrypts the current crypto state using its TEK (Ticket Encryption Key) and sends this blob of data to the client for safe keeping. Sometime later when the client wants to reconnect it sends this blob back to the server who can then recover and re-use that crypto state.

Client side

When the client collects a ticket it contains lots of useful unencrypted information, like the lifetime of the ticket and its time of birth, as well as the TEK encrypted blob. It also contains a random NONCE. During the original handshake after computing the traffic encryption keys the client also computes a Resumption Master Secret (RMS). The preshared key (PSK) to be used for a subsequent re-connection is calculated from this RMS+NONCE combination.

As part of the resumption clientHello the encrypted part of the ticket is returned to the server.

(Note that the client can also connect directly to a server using a PSK derived in some sort of out-of-band process completely independent of any initial handshake. It is an innovation of TLS1.3 to combine these mechanisms.)

Server side

The server can extract the same PSK from the encrypted part of the ticket that the client has returned to it. It is the same PSK that was encrypted with the TEK and sent to the client for safe keeping. Other stuff is bundled into the encrypted part of the ticket as well, for example ticket time of creation plus lifetime, and client identity (if the original handshake included client-side authentication).

The problem is...

The TLS1.3 RFC8446 does not say anything about the TEK, or indeed exactly what the internals of the encrypted part of a ticket look like. Its an opaque blob of encrypted data that only the server knows how to decrypt and make sense of. Each server can independently decide its own mechanism for creating it. As far as the client is concerned its just a meaningless blob passed from the server to the client, and then back again when resumption is requested.

There is another problem – replay attacks as apply in the 0-RTT setting. Early data is encrypted using a key derived from the PSK. However an attacker could simply record the initial clientHello, including the encrypted ticket and early data, and transmit it again a few minutes later, within the lifetime of the ticket. How such an attack might benefit the attacker is out-of-scope for this blog, but its worrying that such an attack should be possible. And combined with the fact that early data is not forward secure (we don't get forward security until we can introduce a Diffie-Hellman key exchange into the mix), then suddenly early data and session resumption start to look a bit iffy.

There are a few obvious mitigations. Don't allow 0-RTT. Ensure that tickets have a limited lifetime.

Or maintain a server-side database of outstanding tickets, so that a ticket can be used only once. And if the server is to maintain such a database, store the PSK key there as well, so that tickets just become database keys (and hence tickets become much smaller). But these solutions require the server to maintain server-side state, and we would much prefer not to have to do that.

So what to do

Since the TLS1.3 RFC offers little by way of guidance we need to come up with a concrete proposal ourselves. I think that we should try to support 0-RTT connections. However a canny server should be prepared to reject early data if it looks at all dodgy. The "freshness" check described in section 8.3 of the RFC seems like it might help.

At a minimum the encrypted part of a TiiTLS ticket should include the ticket time-of-birth plus lifetime, the PSK, and the client side certificate (if one were sent). The server needs only maintain one global AES TEK to encrypt and decrypt this information.

We might simply be guided by section 4 of RFC 5077*. But I am open to suggestions.

*https://datatracker.ietf.org/doc/html/rfc5077#page-10