Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS

Antoine Delignat-Lavaud, Karthikeyan Bhargavan, Alfredo Pironti (Prosecco, Inria Paris)

Cédric Fournet (Microsoft Research)

Pierre-Yves Strub (IMDEA Software Institute)

User Authentication over TLS

- Applications rely on **weak authentication**
  - Web: **passwords**, session **cookies**, single sign-on **tokens**
  - Cookie confidentiality requires **secure flag**
  - Cookie integrity almost **never guaranteed**
  - **Bearer tokens are vulnerable to MITM attacks**

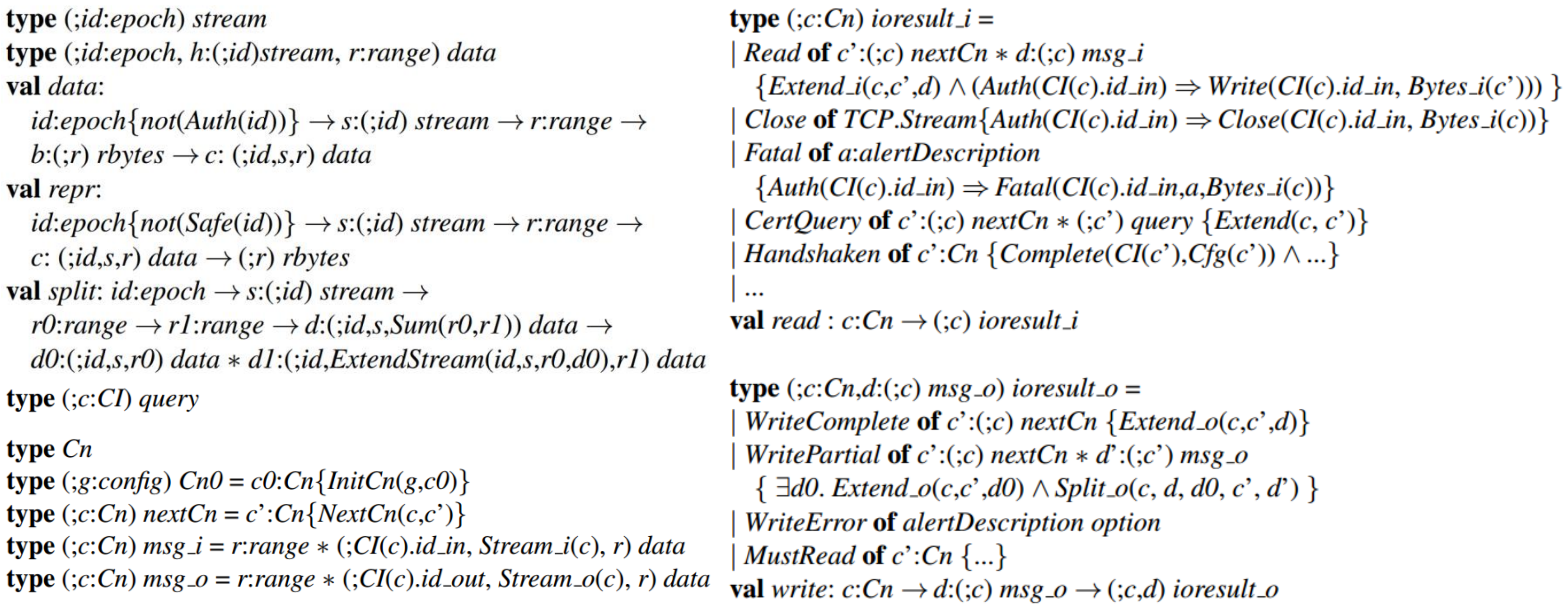
- Countermeasures bind tokens to the **TLS handshake**
  - TLS-OBC [Dietz et al., Usenix Security 2012], Channel ID
  - TLS client authentication after renegotiation
  - EAP-TTLS (wireless networks, VPN…)
  - SCRAM-PLUS (XMPP, mail servers…)
  - Extended Protection for Windows, SAML V2.0, …

Challenges

- Blurred line between application and transport layers
- Does TLS provide the right guarantees?
- Do applications use their TLS libraries correctly?

The API Problem

- What applications want: socket replacement
  - connect(), listen(), accept(), read(), write(), close()
- What we can prove: [miTLS project, S&P'13]



```
type (;id:epoch) stream
type (;id:epoch, h:(;id)stream, r:range) data
val data:
  id:epoch{not(Auth(id))} → s:(;id) stream → r:range →
  b:(;r) rbytes → c: (;id,s,r) data
val repr:
  id:epoch{not(Safe(id))} → s:(;id) stream → r:range →
  c: (;id,s,r) data → (;r) rbytes
val split: id:epoch → s:(;id) stream →
  r0:range → r1:range → d:(;id,s,Sum(r0,r1)) data →
  d0:(;id,s,r0) data ∗ d1:(;id,ExtendStream(id,s,r0,d0),r1) data
type (;c:CI) query

type Cn
type (;g:config) Cn0 = c0:Cn{InitCn(g,c0)}
type (;c:Cn) nextCn = c':Cn{NextCn(c,c')}
type (;c:Cn) msg_i = r:range ∗ (;CI(c).id_in, Stream_i(c), r) data
type (;c:Cn) msg_o = r:range ∗ (;CI(c).id_out, Stream_o(c), r) data
```

```
type (;c:Cn) ioresult_i =
| Read of c':(;c) nextCn ∗ d:(;c) msg_i
   {Extend_i(c,c',d) ∧ (Auth(CI(c).id_in) ⇒ Write(CI(c).id_in, Bytes_i(c'))) }
| Close of TCP.Stream{Auth(CI(c).id_in) ⇒ Close(CI(c).id_in, Bytes_i(c))}
| Fatal of a:alertDescription
   {Auth(CI(c).id_in) ⇒ Fatal(CI(c).id_in,a,Bytes_i(c))}
| CertQuery of c':(;c) nextCn ∗ (;c') query {Extend(c, c')}
| Handshaken of c':Cn {Complete(CI(c'),Cfg(c')) ∧ ...}
| ...
val read : c:Cn → (;c) ioresult_i

type (;c:Cn,d:(;c) msg_o) ioresult_o =
| WriteComplete of c':(;c) nextCn {Extend_o(c,c',d)}
| WritePartial of c':(;c) nextCn ∗ d':(;c') msg_o
   { ∃d0. Extend_o(c,c',d0) ∧ Split_o(c, d, d0, c', d') }
| WriteError of alertDescription option
| MustRead of c':Cn {...}
val write: c:Cn → d:(;c) msg_o → (;c,d) ioresult_o
```

API Example: SSL_read

- Return value 0: Read operation was not successful. The reason may either be:
  - **a clean shutdown** due to a *close_notify* alert sent by the peer (in which case the SSL_RECEIVED_SHUTDOWN flag in the SSL shutdown state is set)
  - **or the peer simply shut down the underlying transport**

OpenSSL Manual

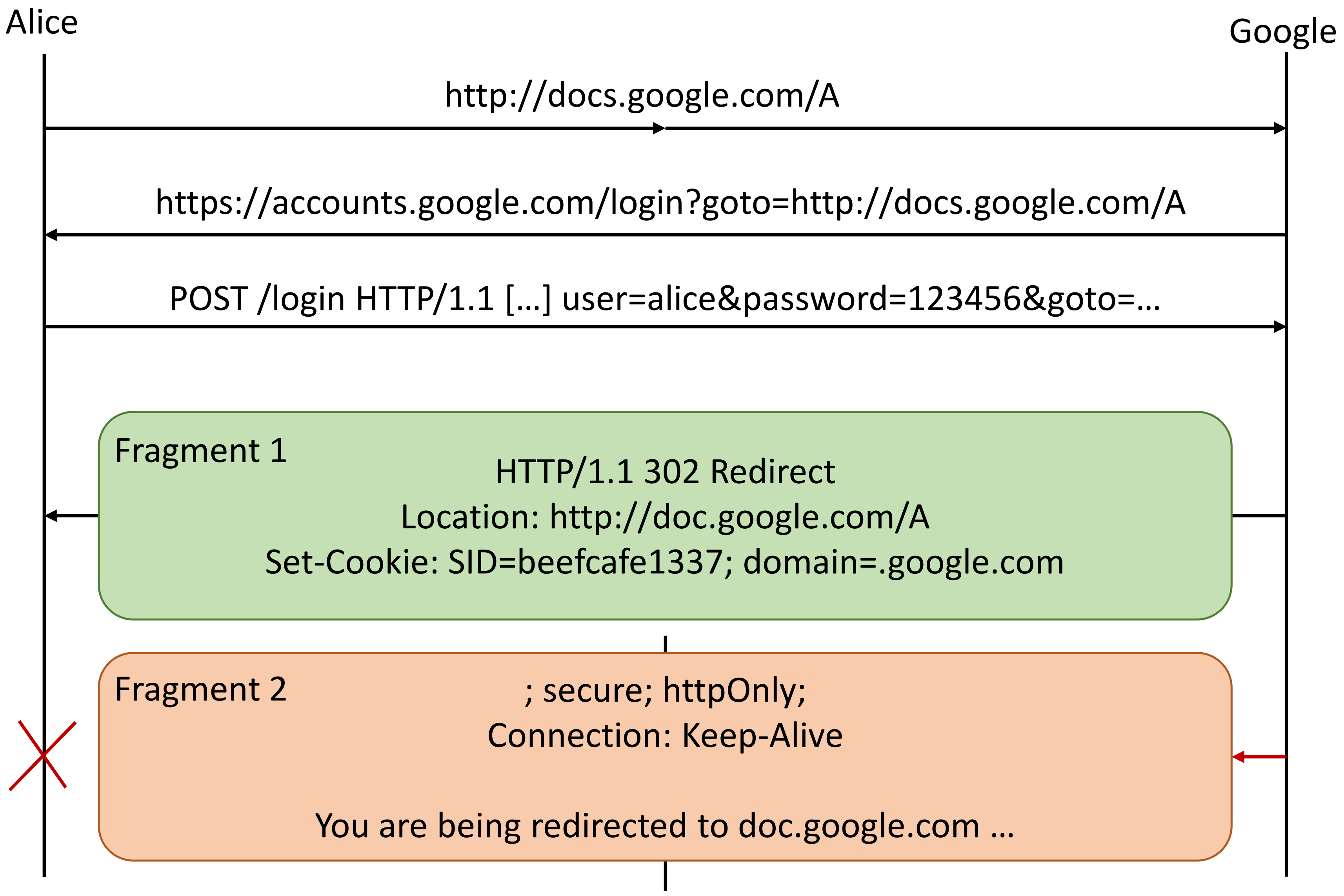
Attack: Cookie Cutter

- Network attacker can truncate HTTPS contents by closing underlying TCP connection

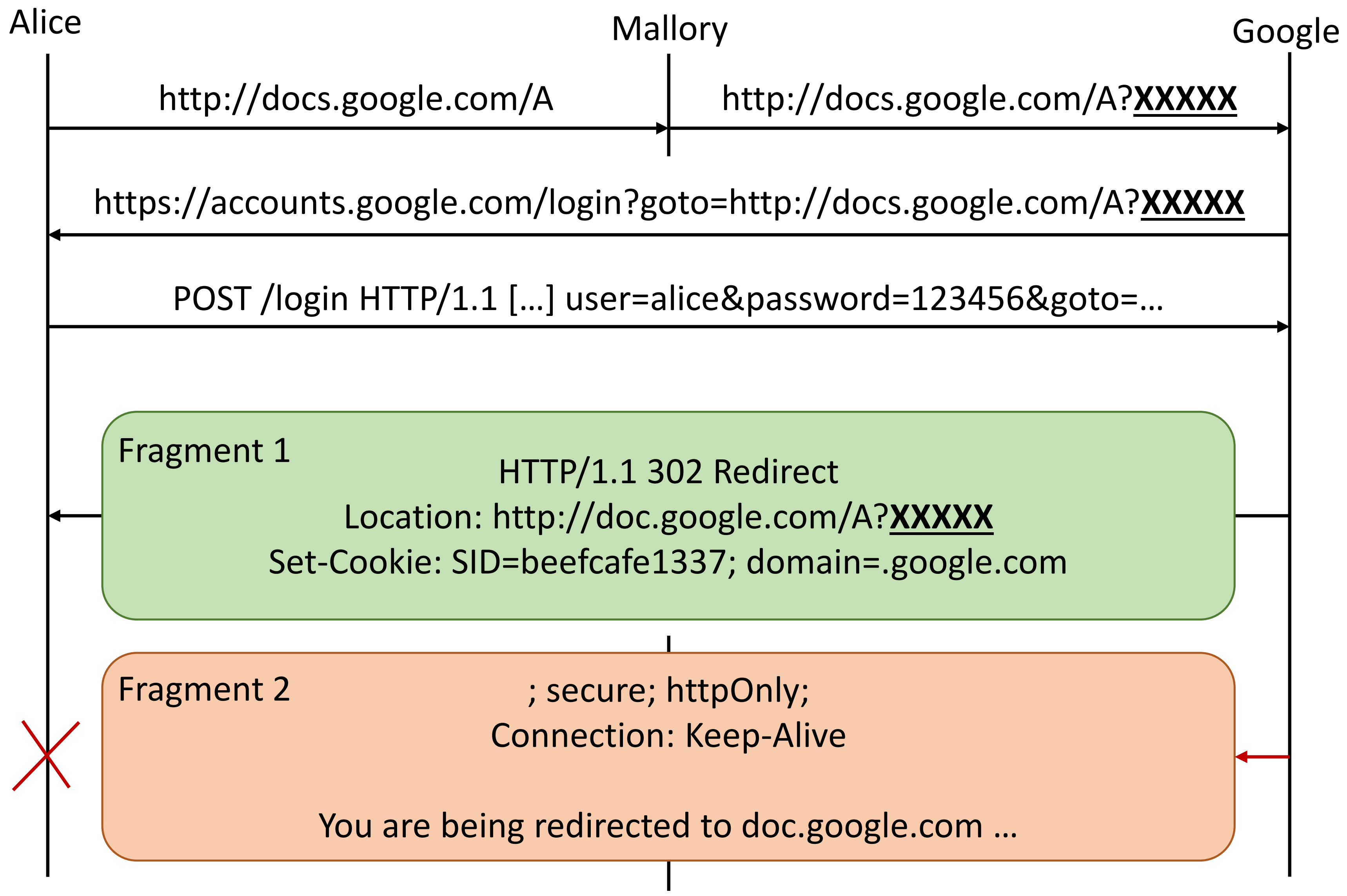
- **Security is an opt-in feature of cookies**
    - Set-Cookie: SID=BEEFCAFE; domain=a.com; secure

- What if we truncated the secure flag?
    - Header becomes syntactically invalid
    - “Conservative in what you send, liberal in what you accept”
    - Exploit fragmentation + plaintext injection for precise truncation point control

Attack: Cookie Cutter


Alice
Google
http://docs.google.com/A
https://accounts.google.com/login?goto=http://docs.google.com/A
POST /login HTTP/1.1 […] user=alice&password=123456&goto=…
Fragment 1
HTTP/1.1 302 Redirect
Location: http://doc.google.com/A
Set-Cookie: SID=beefcafe1337; domain=.google.com
Fragment 2
; secure; httpOnly;
Connection: Keep-Alive
You are being redirected to doc.google.com …

Attack: Cookie Cutter


Alice
Mallory
Google
http://docs.google.com/A
http://docs.google.com/A?**<u>XXXXX</u>**
https://accounts.google.com/login?goto=http://docs.google.com/A?**<u>XXXXX</u>**
POST /login HTTP/1.1 […] user=alice&password=123456&goto=…
Fragment 1
HTTP/1.1 302 Redirect
Location: http://doc.google.com/A?**<u>XXXXX</u>**
Set-Cookie: SID=beefcafe1337; domain=.google.com
Fragment 2
; secure; httpOnly;
Connection: Keep-Alive
You are being redirected to doc.google.com …

Cookie Cutter: Impact and Mitigation

- Network attacker can get victim’s browser to **process malicious truncated headers**
  - Steal secure cookies
  - Disable Strict-Transport-Security (SSL stripping)

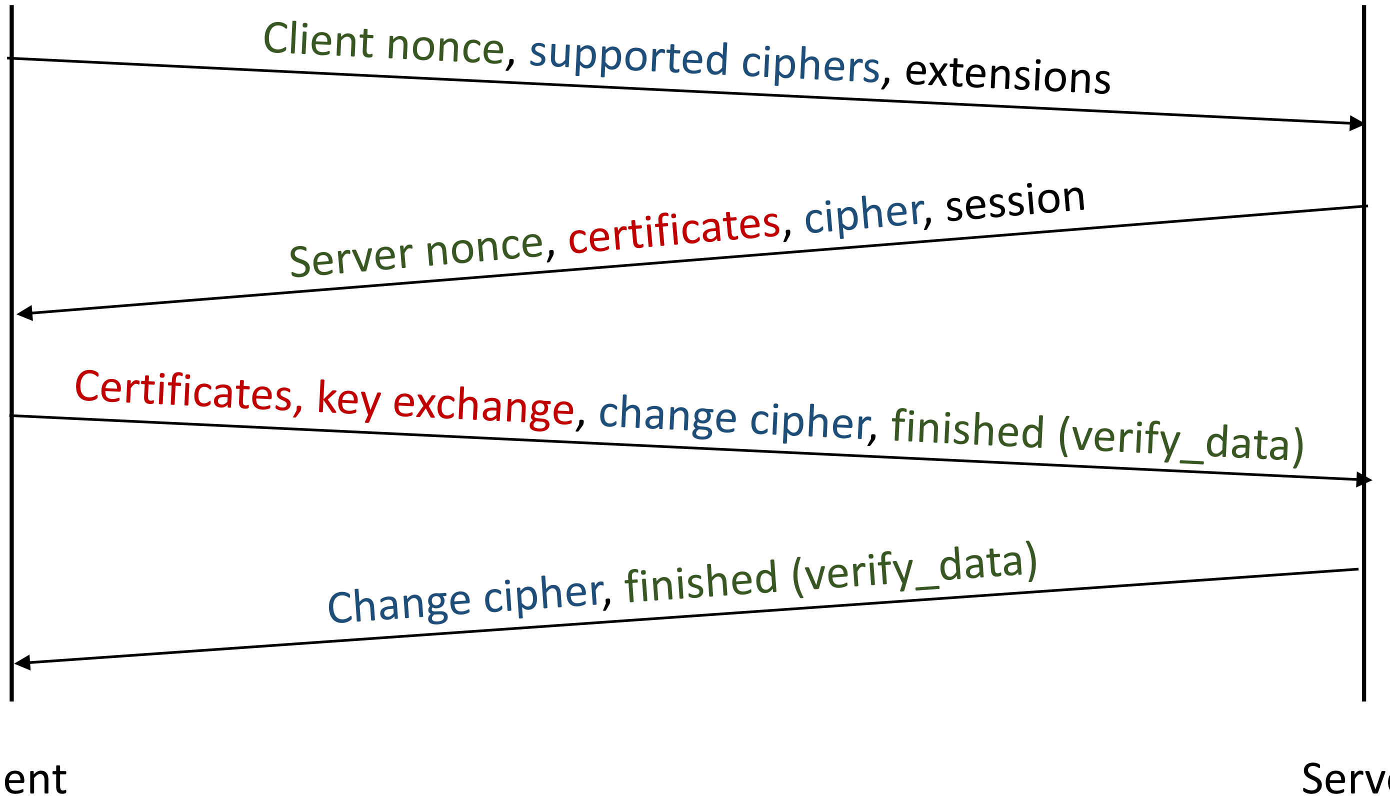
- Fixed in Chromium (NSS library, CVE-2013-2853), Android Browser (OpenSSL), and Safari (Secure Transport, APPLE-SA-2014-04-22-1)
  - Was the browser or the TLS library to blame?

API Example: Renegotiation

- "If peer requests a renegotiation, **it will be performed transparently** during the SSL_read() operation."
- "As at any time a re-negotiation is possible, a call to SSL_write() **can also cause read operations**!"

OpenSSL Manual

Background: TLS Handshake


Client nonce, supported ciphers, extensions
Server nonce, certificates, cipher, session
Certificates, key exchange, change cipher, finished (verify_data)
Change cipher, finished (verify_data)
Client
Server

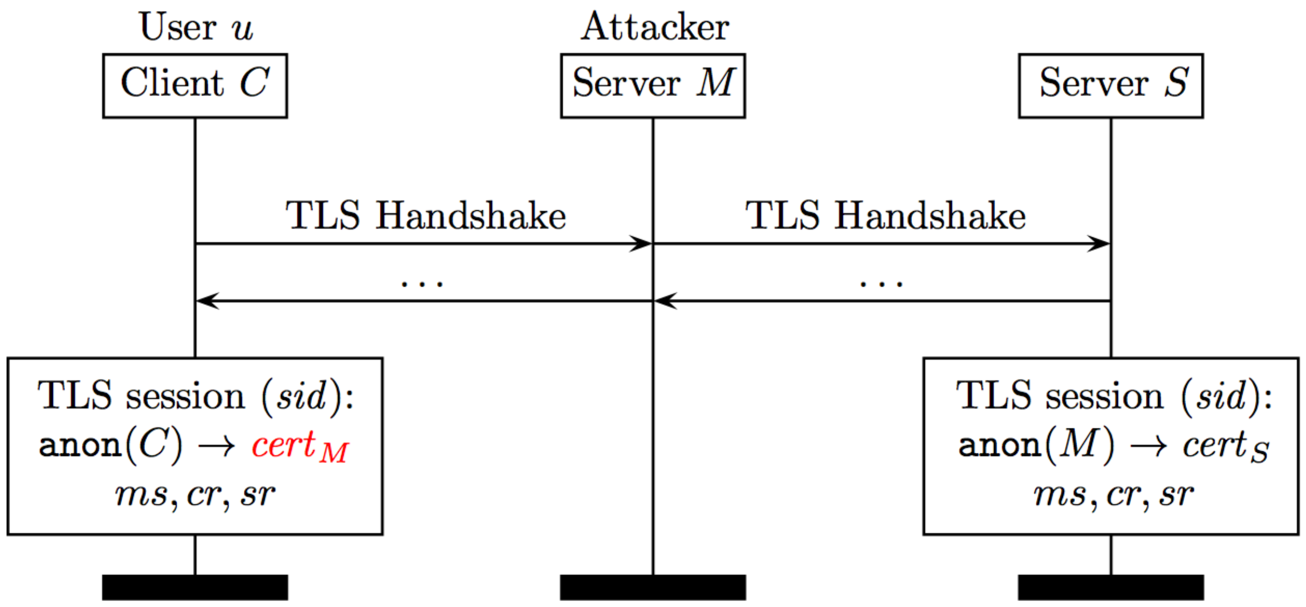
- Key exchange produces pre-master secret (PMS)
- MS = MS-PRF(PMS, Client Nonce, Server Nonce)

Background: 2009 Renegotiation Attack

- Renegotiation
  - A handshake is tunneled within an established TLS channel
  - The newly negotiated parameters are used thereafter
- Problem
  - New (inner) handshake not bound to outer tunnel
  - Is the peer starting a new session or renegotiating?
- Deployed solution
  - Renegotiation indication: mandatory extension
  - SRI = verify_data of the latest handshake on connection
  - New handshake authenticate the SRI of the previous one
  - Fresh connections, resumption start with empty SRI

Attack: 3Shake Step 1

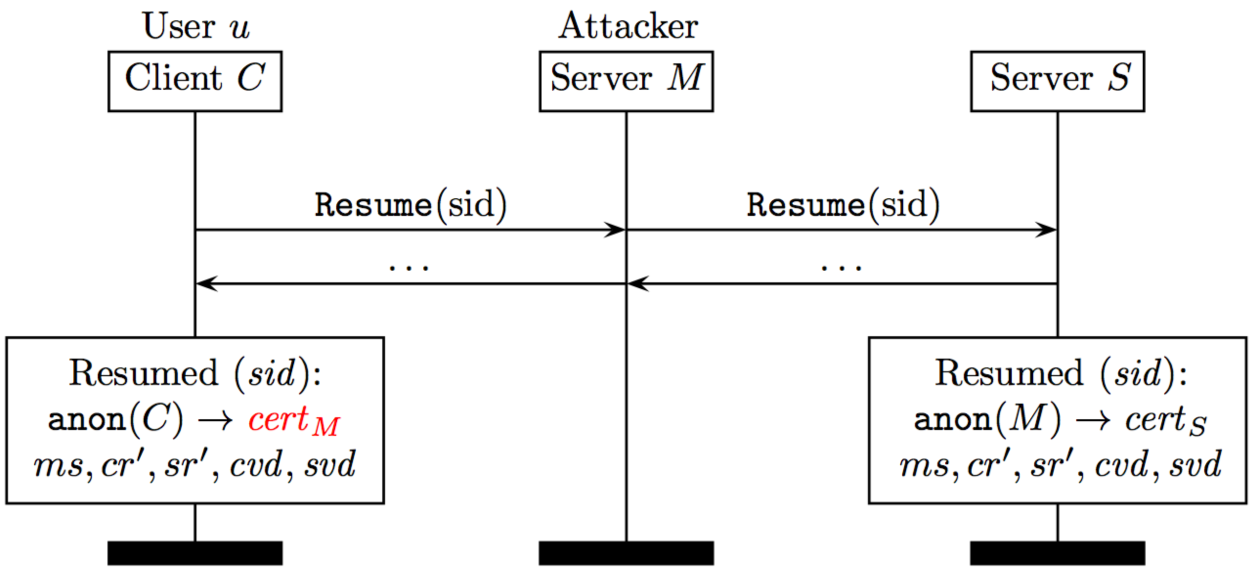
- A malicious server M can synchronize the key of a TLS session with a client C on another server S
  - RSA: M re-encrypts C’s PMS under S’ public key
  - DHE: M sends degenerate group parameters


User u
Client C
Attacker
Server M
Server S
TLS Handshake
TLS Handshake
...
...
TLS session (sid): anon(C) → $cert_M$ ms, cr, sr
TLS session (sid): anon(M) → $cert_S$ ms, cr, sr

- **Neither PMS nor MS is unique to a TLS session**

Attack: 3Shake Step 2

- **C can resume session with M on S** without any tampering. Hash of message log (*verify_data*) is equal on both sides

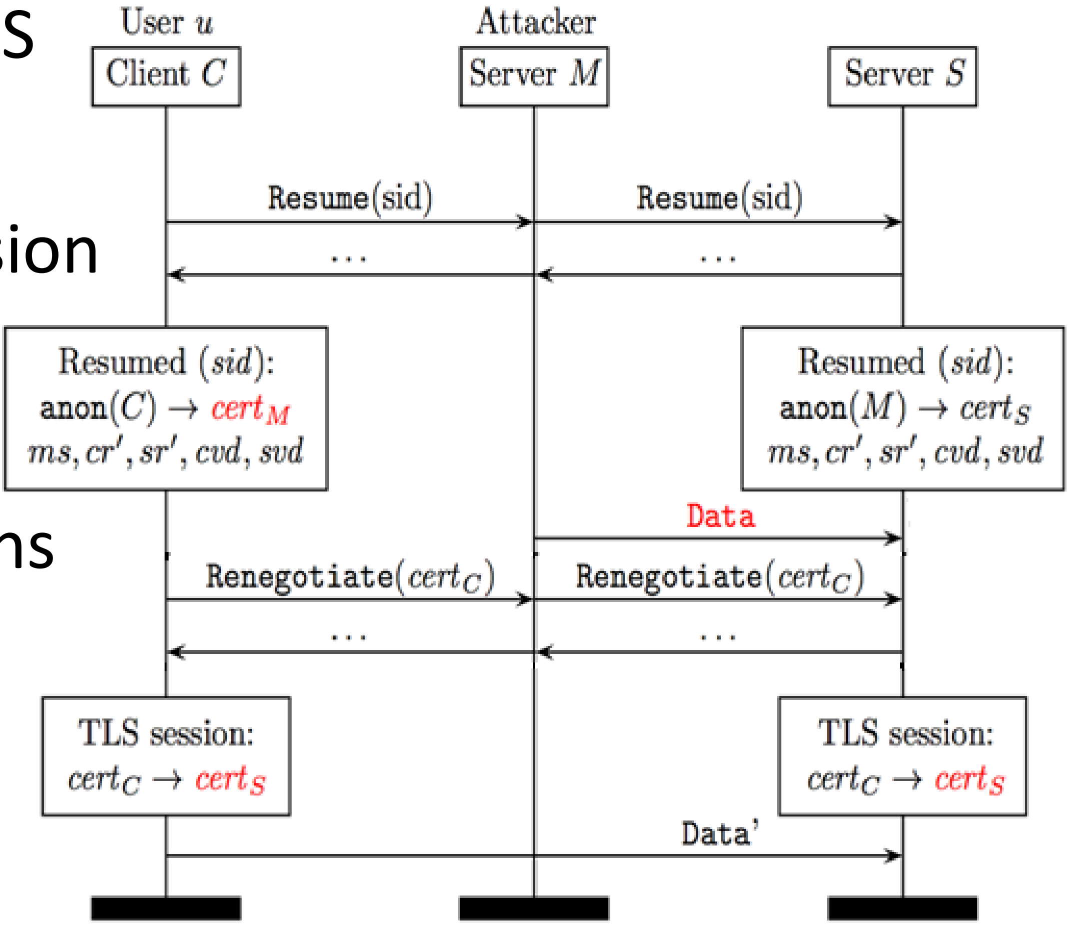


User u
Client C
Attacker
Server M
Server S
$\texttt{Resume}(\text{sid})$
$\texttt{Resume}(\text{sid})$
$\ldots$
$\ldots$
Resumed (sid): $\texttt{anon}(C) \rightarrow cert_M$ ms, cr', sr', cvd, svd
Resumed (sid): $\texttt{anon}(M) \rightarrow cert_S$ ms, cr', sr', cvd, svd

- The *tls-unique* binding (first *verify_data* of last handshake on the current conection) is **not unique after resumption**!

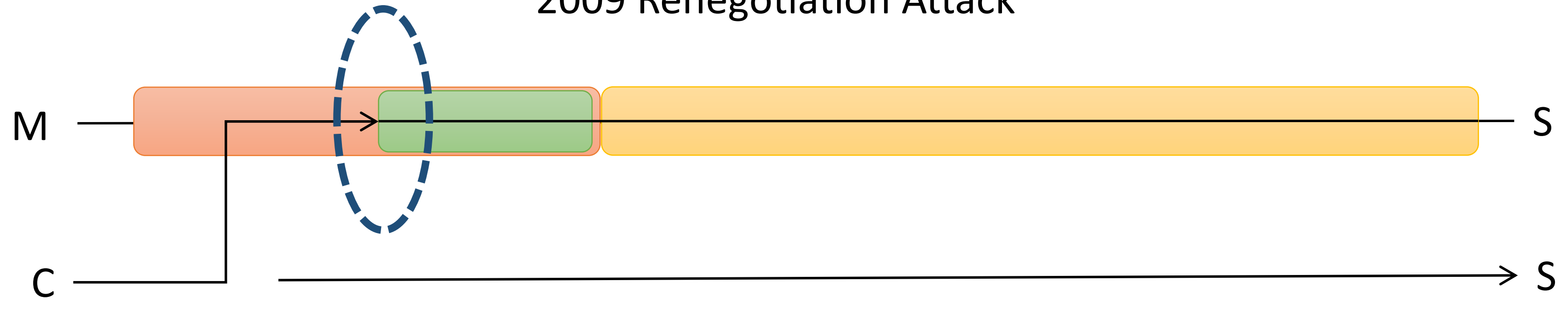
Attack: 3Shake Step 3

- M can forward authenticated renegotiation from C to S
- S associates the full session with C's certificate
- Implementation decisions
  - How does C handle the certificate change?
  - How does S handle data injected by M before renegotiation?

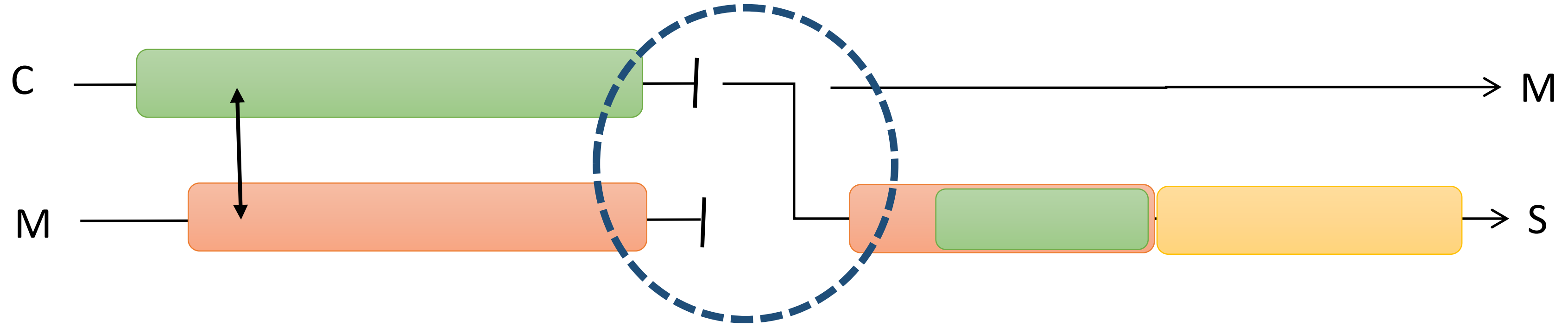

User u
Client C
Attacker
Server M
Server S
Resume(sid)
Resume(sid)
...
...
Resumed (sid): anon(C) → $cert_M$ ms, cr', sr', cvd, svd
Resumed (sid): anon(M) → $cert_S$ ms, cr', sr', cvd, svd
Data
Renegotiate($cert_C$)
Renegotiate($cert_C$)
...
...
TLS session: $cert_C$ → $cert_S$
TLS session: $cert_C$ → $cert_S$
Data'

TLS Session Headache

2009 Renegotiation Attack



Triple Handshake Attack



3Shake: Impact and Mitigations

- Conditions
  - C is willing to authenticate on M with his certificate
  - C ignores the server certificate change during renegotiation
  - S concatenates the data before and after renegotiation
- Impact
  - M can inject **malicious data authenticated as C**

3Shake: Mitigations

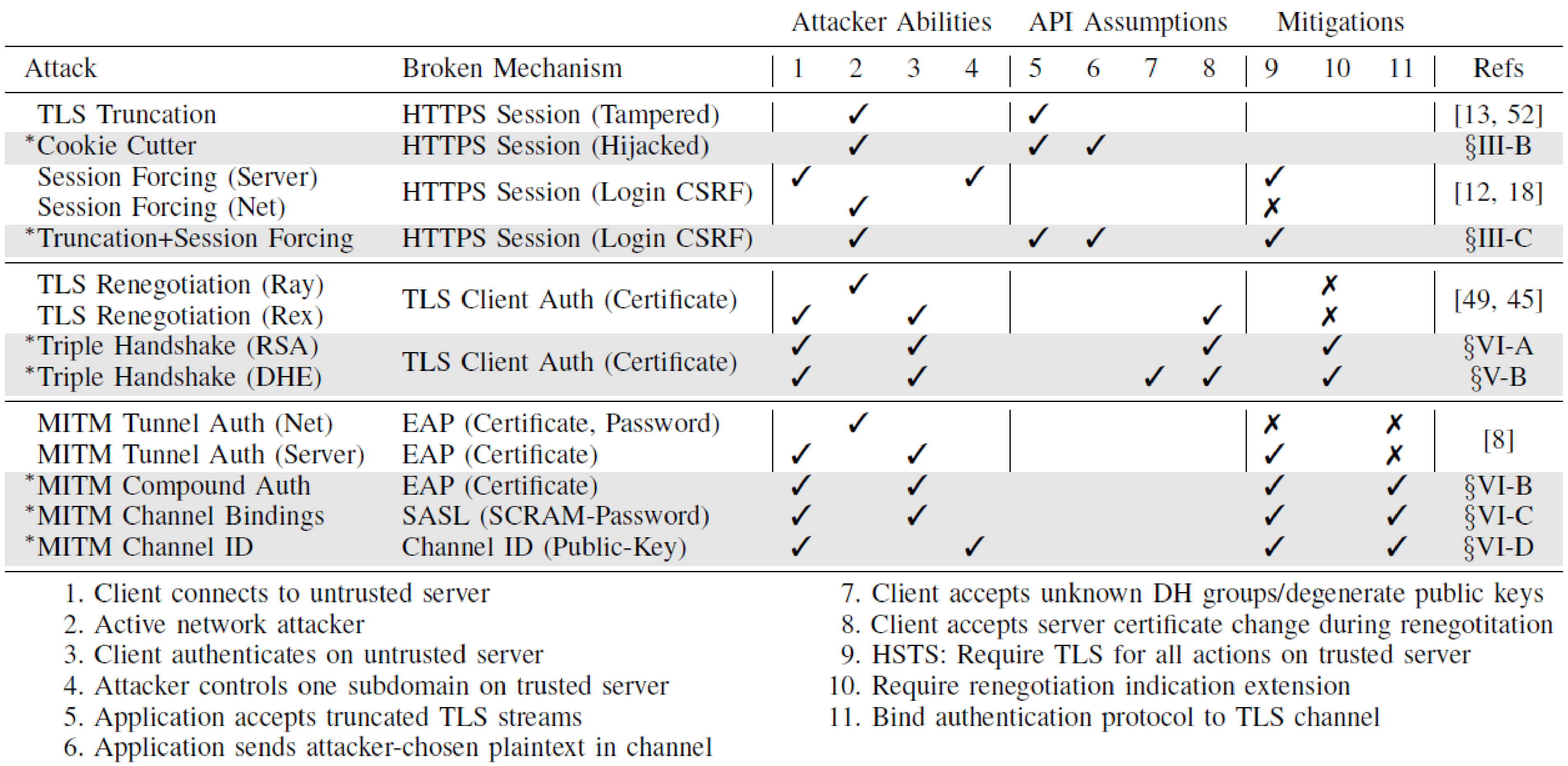
- Short-term Mitigations
  - C can block server certificate changes
    - Chomium (CVE-2013-6628)
    - Safari (APPLE-SA-2014-04-22-2)
    - Internet Explorer (pending)
  - S may refuse to accept data before client authentication

- Long-term: fixing the standards
  - We propose MS’ = MS-PRF’(PMS, tls-session-hash)
  - *tls-session-hash* = hash of the handshake messages that created the session up to client key exchange
  - Under consideration by the IETF (draft-bhargavan-tls-session-hash-01)

Why 3Shake Wasn't Discovered Earlier

- Bhargavan et al., IEEE S&P'13
  Implementing TLS with Verified Cryptographic Security
  - Attack falls outside the scope of their authentication guarantees for resumption
- Giesen *et al.,* CCS'13
  On the Security of TLS Renegotiation
  - Doesn't model resumption
- Krawczyk *et al*., CRYPTO'13. On the Security of the TLS Protocol: A Systematic Analysis
  - Doesn't model resumption or renegotiation

Variants and Related Attacks



		Attacker Abilities				API Assumptions				Mitigations			
Attack	Broken Mechanism	1	2	3	4	5	6	7	8	9	10	11	Refs
TLS Truncation	HTTPS Session (Tampered)		✓			✓							[13, 52]
*Cookie Cutter	HTTPS Session (Hijacked)		✓			✓	✓						§III-B
Session Forcing (Server)	HTTPS Session (Login CSRF)	✓			✓					✓			[12, 18]
Session Forcing (Net)			✓							✗			
*Truncation+Session Forcing	HTTPS Session (Login CSRF)		✓			✓	✓			✓			§III-C
TLS Renegotiation (Ray)	TLS Client Auth (Certificate)		✓								✗		[49, 45]
TLS Renegotiation (Rex)		✓		✓					✓		✗		
*Triple Handshake (RSA)	TLS Client Auth (Certificate)	✓		✓					✓		✓		§VI-A
*Triple Handshake (DHE)		✓		✓				✓	✓		✓		§V-B
MITM Tunnel Auth (Net)	EAP (Certificate, Password)		✓							✗		✗	[8]
MITM Tunnel Auth (Server)	EAP (Certificate)	✓		✓						✓		✗	
*MITM Compound Auth	EAP (Certificate)	✓		✓						✓		✓	§VI-B
*MITM Channel Bindings	SASL (SCRAM-Password)	✓		✓						✓		✓	§VI-C
*MITM Channel ID	Channel ID (Public-Key)	✓			✓					✓		✓	§VI-D

1. Client connects to untrusted server
2. Active network attacker
3. Client authenticates on untrusted server
4. Attacker controls one subdomain on trusted server
5. Application accepts truncated TLS streams
6. Application sends attacker-chosen plaintext in channel
7. Client accepts unknown DH groups/degenerate public keys
8. Client accepts server certificate change during renegotitation
9. HSTS: Require TLS for all actions on trusted server
10. Require renegotiation indication extension
11. Bind authentication protocol to TLS channel

See paper for details.

Towards Secure TLS Applications

- It is **too difficult to use current TLS APIs securely**
    - Certificate validation
    - Session and cache management
    - Identity and session transitions
    - Shutdown mode

- **We must verify applications under the precise guarantees offered by the TLS API**

- Critical for features outside the channel abstraction
    - SNI, ALPN, Channel ID, Channel Bindings, renegotiation, client authentication, Keying Material Exporters…

A Verified HTTPS Client

- We introduce miHTTPS, a **verified HTTPS client** built on top of the miTLS library

- miHTTPS supports cookies, TLS client authentication, resumption and renegotiation
  - Captures our attacks

- Using F7 along with Z3, we extend the refinements of the miTLS API into **HTTP-level security goals**:
  - Request integrity
  - Response integrity
  - Response tracking using fresh random cookies

Conclusions

- We found that applications fail to use the basic and advanced features of TLS implementations securely
- We found a **new logical flaw** in the resumption feature of the TLS protocol
- The **TLS library is not the right unit of verification** for today’s complex application protocols
  - We advocate verifying thin application protocol libraries similar to miHTTPS

Questions?

https://www.mitls.org