Replay Attack in TLS 1.3 0-RTT Handshake: Countermeasure Techniques

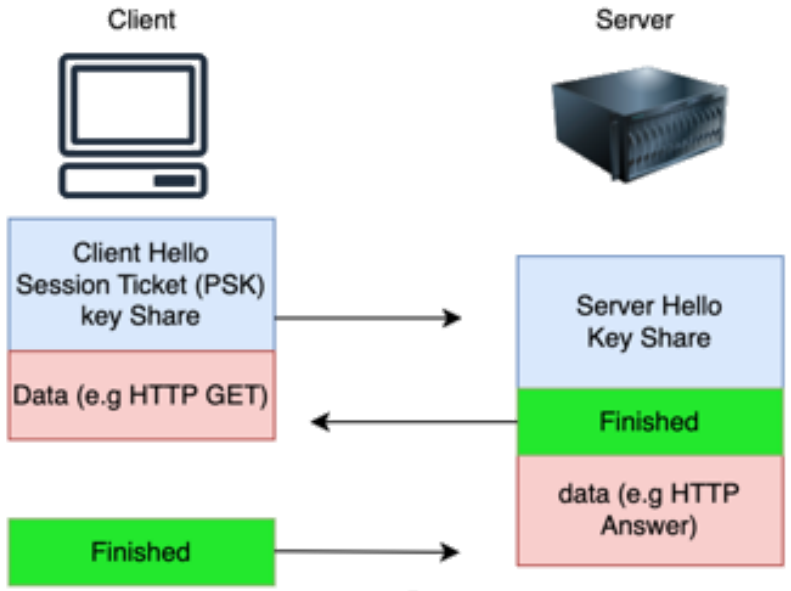
Network Security (933II)
M.Sc. Cybersecurity
Paolo Bernardi (660944)

The Paper

- **Conference:** 2023 IEEE 6th International Conference on Electrical, Electronics and System Engineering (ICEESE)
- **Authors:** M.E Abdelhafez (Malaysia), Sureswaran Ramadass (Malaysia), Mohammed S. M. Gismallab (Saudi Arabia)
- **Goal:** review anti-replay protection techniques
- **Keywords:** TLS 1.3, replay attack, 0-RTT, handshake

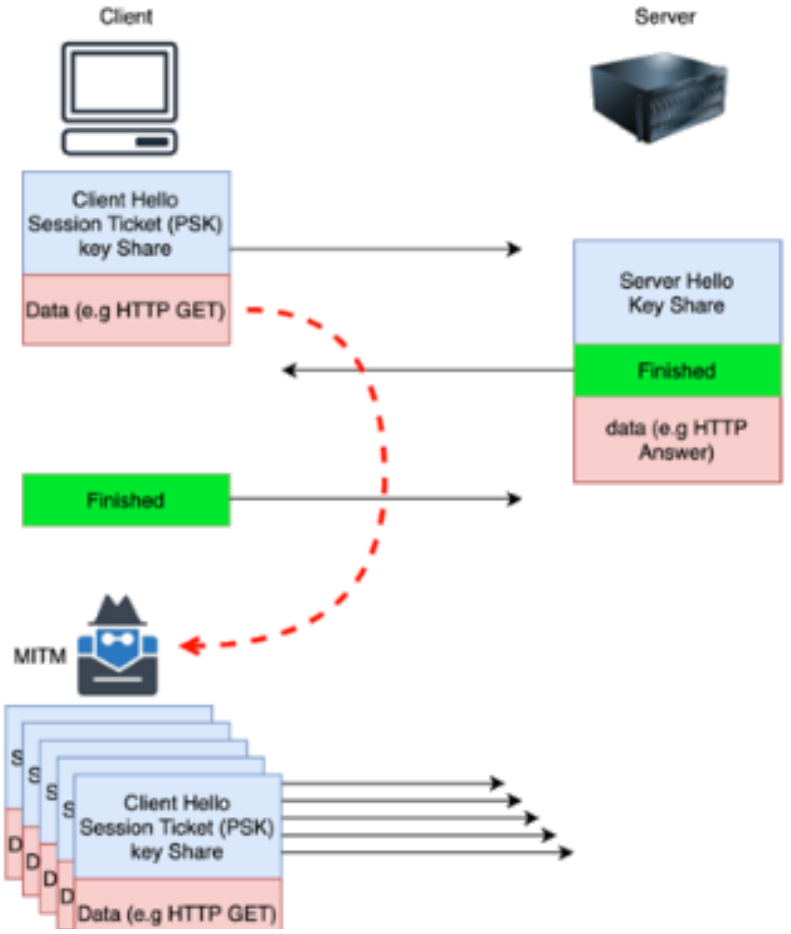
Context

- TLS resumable connections
- TLS 1.3 introduced **0-RTT** resume mode, based on a **Session-Ticket key** created during the initial full handshake
- 0-RTT obtained by sending a **single message** that contains both the **ClientHello** (with a known Session-Ticket key) and **Application Data** (also known as **Early Data**)



Attack Scenarios

- **Replay** attack
- Attacker intercepts and replays **ClientHello** messages with **Early Data**
- The replayed message is valid because the **ClientHello** contains a **Session-Ticket key** recognized by the server
- **ALTERNATIVE SCENARIO:** the attacker performs a **MITM** and makes the web client **to believe that the 0-RTT message wasn't received**, causing it to be sent multiple times



Project work

L'esame del corso comprende:

- Sviluppo di un'applicazione SOA
- Relazione sul lavoro svolto

Il lavoro potrà essere svolto nell'ultima parte del corso, dove sarò a disposizione per chiarimenti e supporto (a distanza).

Cosa sono le architetture SOA?

Le *architetture orientate ai servizi* (SOA) sono un approccio alla **progettazione** e all'**implementazione** dei sistemi software basato sulla creazione di *servizi* **indipendenti** e **interoperabili**.

Perché SOA?

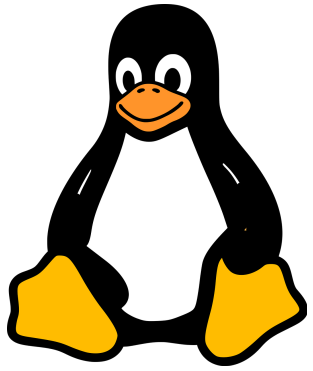
- Flessibilità
- Riusabilità
- Interoperabilità (e.g. software legacy)
- Scalabilità

Perché non SOA?

- Rete: overhead
- Rete: sicurezza (autenticazione, disponibilità ecc...)

💡 Suggerimento

> Usate architetture SOA solo quando è veramente necessario. Altrimenti un bel "monolite" basta e avanza (ma non ditelo a Tanenbaum).



I servizi

Come definire un *servizio*?

- Funzionalità indipendente
- Funzionalità autocontenuta
- ... qualunque funzionalità abbia senso separare dalle altre

💡 Suggerimento

> Non impelagatevi troppo in definizioni troppo stringenti, tanto dipende tutto dal contesto in cui **voi** vi trovate (ivi e soprattutto inclusi i vostri committenti)

Service contract

Alla fine la caratteristica fondamentale tra il servizio e chi ne usufruisce (client-server, anyone?) è il cosiddetto **contratto** o **interfaccia**.

- Protocollo di comunicazione
- Formato dati

Il contratto può essere esplicitato sotto forma di funzioni e relativi parametri.

Esempio di service contract

Servizio: Auth (authentication & authorization)

- Login(username: str, password: str) -> bool
- IsAuthenticated -> bool
- Logout
- Permissions(username: str) -> List[str]

> Il contratto è scritto in uno pseudolinguaggio inventato sul momento. In ogni caso sono importanti i nomi delle funzioni, i parametri (tipizzati) e i dati restituiti.

Interoperabilità: perché?

I sevizi sono spesso implementati con diverse tecnologie:

- Legacy (es. COBOL)
- Market standards (es. Java, C#)
- Moderne (es. Go, Rust)
- Esoteriche (es. Lisp)

Per questo motivo è fondamentale che i servizi siano interoperabili, ovvero che rispettino dei precisi standard di comunicazione e formato dati (es. SOAP, REST, Protobuf)

Interoperabilità e standardizzazione

Per avere sistemi SOA interoperabili servono degli **standard**, o quantomeno delle **best practice** o dei **pattern**. Eccone alcuni esempi:

Nome	Formato dati	Protocollo di rete	Standard?
RMI, Corba, RPC	Binario	TCP	In parte
SOAP	XML	HTTP, SMTP (wow!), TCP	W3C
REST	JSON, whatever	HTTP (in genere)	de facto
Protobuf	Binario	TCP	no

Ingegneria del software

- 1968/69: NATO conferences on software engineering (risposta alla *software crisis*)
- Ingegneria: applicazione sistematica di conoscenze scientifiche per risolvere problemi concreti
- Software: ingegneria o superstizione (o meglio **cargo cult**-based engineering)?

SOA e ingegneria

Quando parliamo di **architettura** (la "A" di SOA) parliamo di ingegneria:

- **Progettare** un sistema prima di implementarlo evita alcuni problemi spinosi (ne rimangono comunque molti altri)
- Progettare **troppo** un sistema, a seconda del **contesto**, può essere eccessivo (vedi il suggerimento SOA VS monoliti): in questo caso si potrebbe ricadere nella superstizione

Progettazione dei servizi

Principi da seguire:

- **Incapsulamento:** l'interfaccia pubblica deve essere ridotta al minimo
- **Riuso:** l'interfaccia pubblica dovrebbe essere abbastanza flessibile da consentire il riuso del servizio in altri contesti

Più in dettaglio (ma sempre molto arbitrario):

- **Alta coesione:** un singolo servizio dovrebbe contenere internamente codice ad alto grado di correlazione
- **Basso accoppiamento:** tra i diversi servizi, invece, il grado di correlazione dovrebbe essere quanto più basso possibile

Service boundary

Il problema, quindi, è determinare:

- Quali sono i servizi
- Quali sono le funzionalità di ciascun servizio

Questo è il cosiddetto problema del **service boundary**.

Nonostante i principi elencati precedentemente la soluzione deriva inevitabilmente dall'esperienza e dalla pratica empirica.

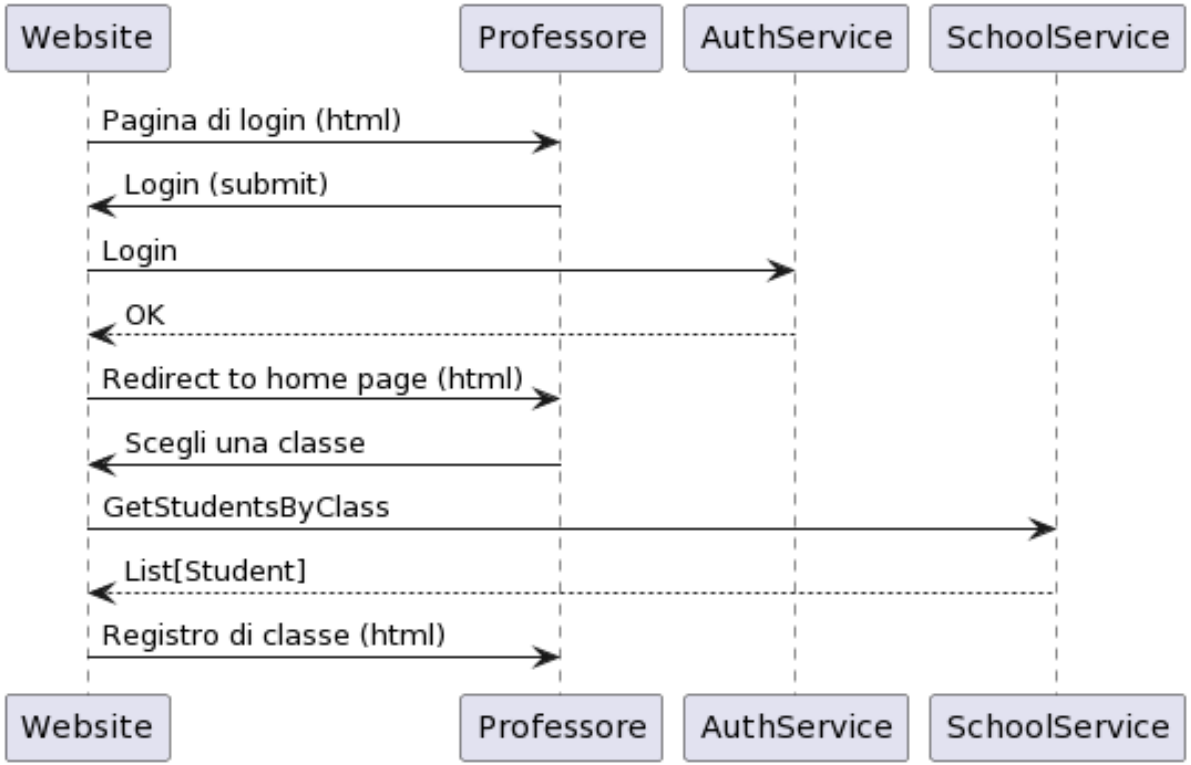
Interazione tra servizi

Non basta definire i servizi ed i loro contratti, bisogna anche considerare come andranno ad interagire tra loro.

Possiamo utilizzare diversi metodi per illustrare le interazioni, uno comune sono i **diagrammi di sequenza UML**.

Interazione tra servizi: il registro elettronico

Questo è un'esempio semplificato di **registro elettronico**.



PlantUML: sequence diagrams

https://plantuml.com

```
@startuml
Website -> Professore : Pagina di login (html)
Professore -> Website : Login (submit)
Website -> AuthService : Login
AuthService --> Website : OK
Website -> Professore : Redirect to home page (html)
Professore -> Website : Scegli una classe
Website -> SchoolService : GetStudentsByClass
SchoolService --> Website : List[Student]
Website -> Professore : Registro di classe (html)
@enduml
```

Esercizio 1

> Immagina di essere un architetto software incaricato di progettare un'architettura SOA per un negozio online. Il negozio vende prodotti e offre servizi ai clienti. La tua sfida è identificare i servizi e le funzionalità proposte per questa architettura SOA.

1. Identifica le funzionalità chiave
2. Identifica i servizi
3. Definisci il contratto dei servizi individuati
4. Mostra qualche esempio di interazione tra i servizi