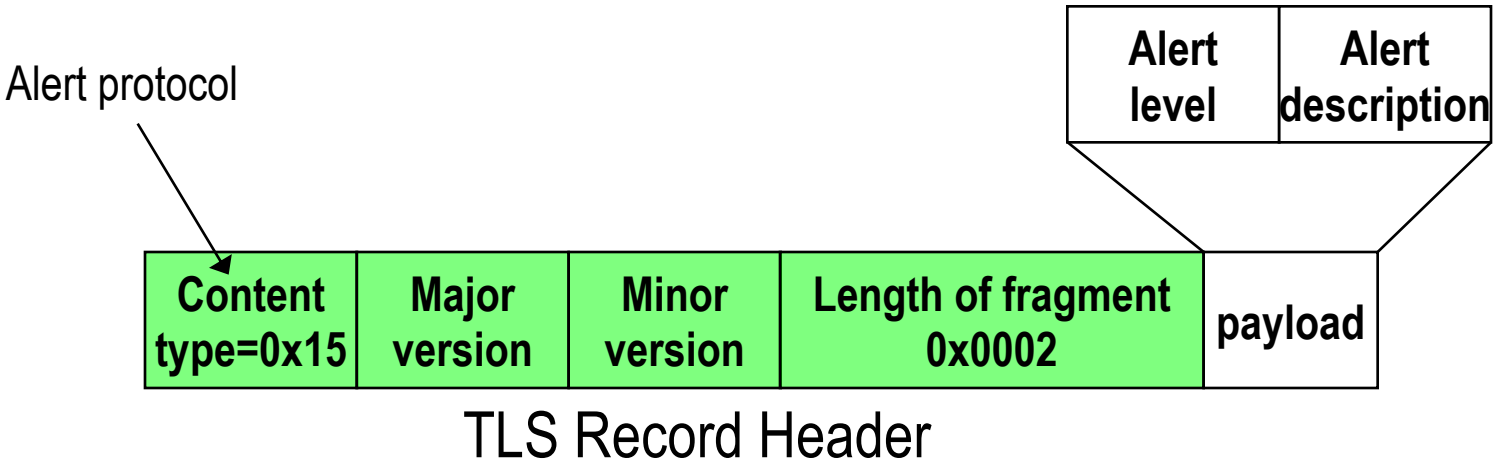
TLS connection management & application support

Alert Protocol

- **TLS defines special messages to convey "alert" information between the involved fields**
- **Alert Protocol messages encapsulated into TLS Records**
  - And accordingly encrypted/authenticated
- **Alert Protocol format (2 bytes):**
  - First byte: Alert Level
    - warning(1), fatal(02)
  - Second Byte: Alert Description
    - 23 possible alerts



TLS Record Header

Sample alerts

See RFC2246 for all alerts and detailed description

- **unexpected_message**
  - Inappropriate message received
    - → Fatal - should never be observed in communication between proper implementations
- **bad_record_mac**
  - Record is received with an incorrect MAC
    - → Fatal

FATAL: riparte dall'inizio, abort e restart
WARNING: left to implementation

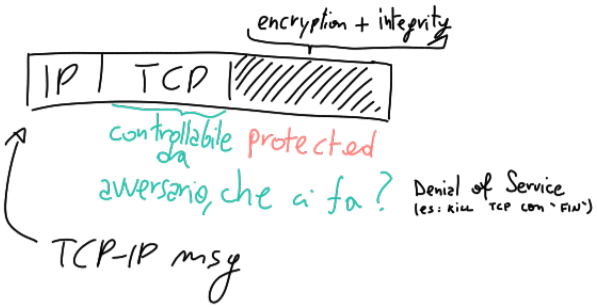
- **record_overflow**
  - Record length exceeds 2^{14}+2048 bytes
    - → Fatal
- **handshake_failure**
  - Unable to negotiate an acceptable set of security parameters given the options available
    - → Fatal
- **bad_certificate, unsupported_certificate, certificate_revoked, certificate_expired, certificate_unknown**
  - Various problems with a certificate (corrupted, signatures did not verify, unsupported, revoked, expired, other unspecified issues which render it unacceptable
    - → Warning or Fatal, depends on the implementation

If fatal, terminate and do not allow resumption with same security parameters (clear all!)

Remark: TLS do NOT protect TCP

→ Open to DoS attacks

- ⇨ Anyone can kill a TCP connection
  - → Just spoof a RST…



→ How to protect non SSL/TLS-aware applications?

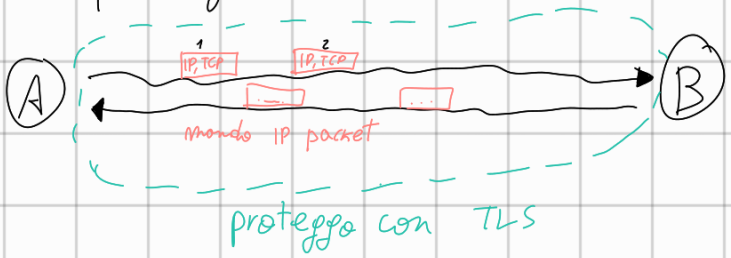
- ⇨ stunnel = TCP over TLS over TCP (!!!)
  - → www.stunnel.org
  - → Use as LAST resort…
- ⇨ TCP over TCP tunnels: performance impairment!!
  - → Conflicting congestion control loops
  - → Use DTLS tunnels, instead

Se volessi proteggere TLS (TCP) header, oltre a PAYLOAD?

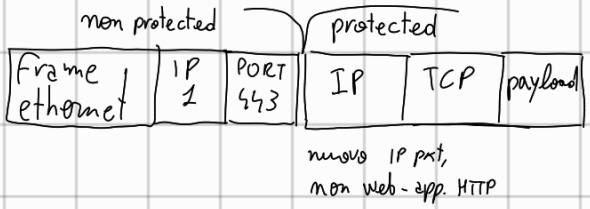
Posso usare IPSEC o "metodi alternativi", come:

Tunnel di TCP in TLS collection:

Nelle VPN creo encrypted tunnel da A a B e protocollo per farlo. IPSEC poco friendly. Creo TLS tunnel



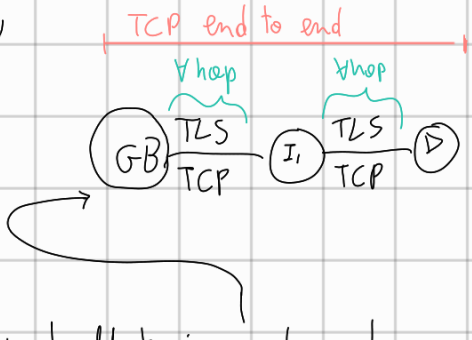
pkt:



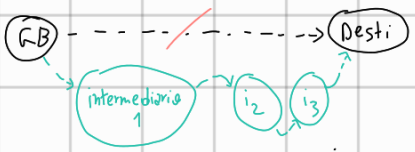
Perché non dovrei inviare pkt TCP su TLS tunnel?

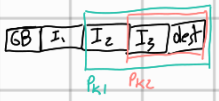
Motivi di performance, NON sicurezza!!

si è sostituito TLS con IPsec all'apparenza MOLTO veloce ma NON controllava la congestione, si passa a DATAGRAM TLS. Si scopre che il modello TLS/TCP controllava congestione sia in TCP end2end che ∀ host. (in conflitto, ho loop)



- Aprire TLS tunnel e trafficare TCP payload sopra non è buona idea! Usato in TOR browser!

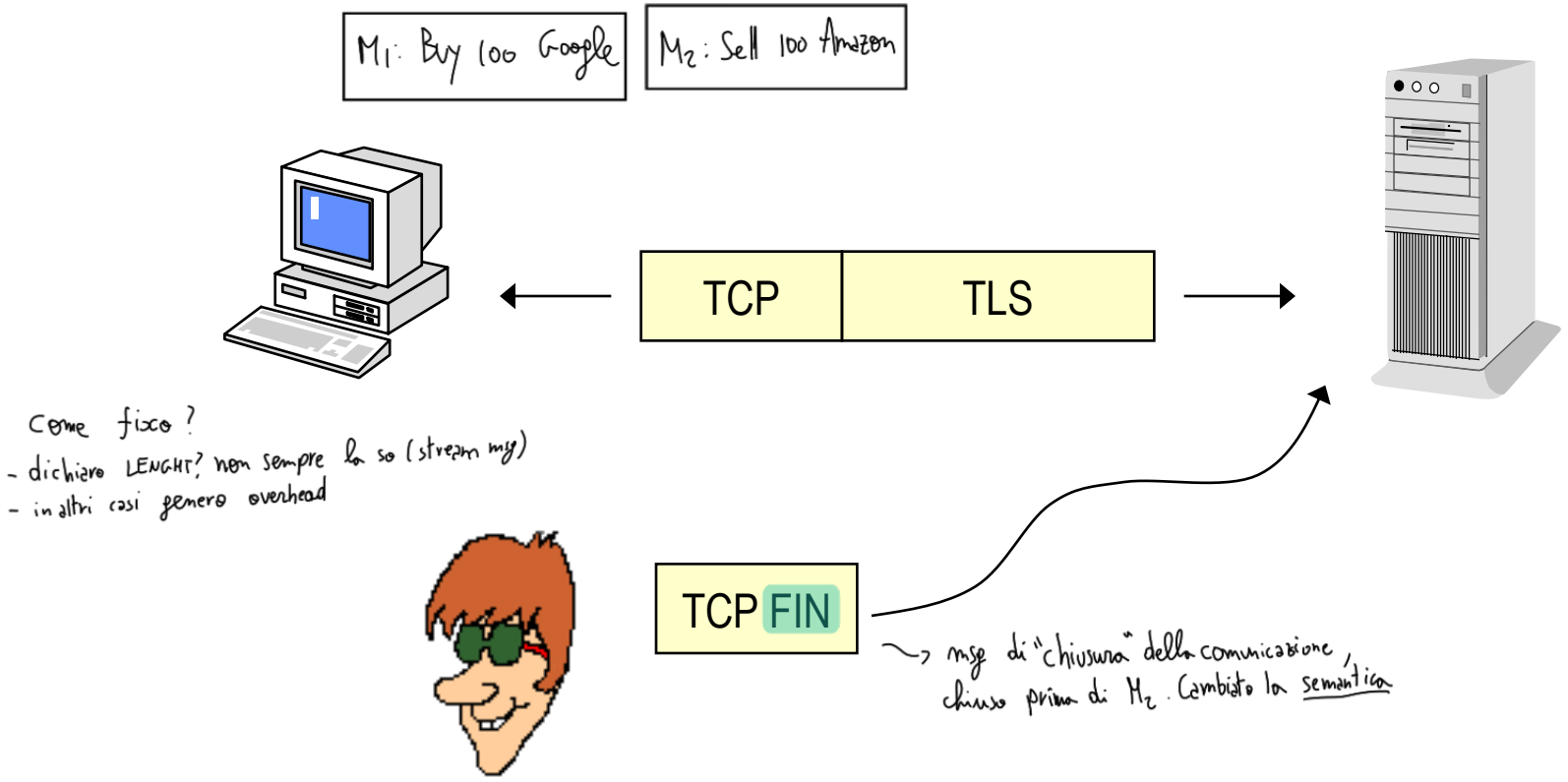
ONION ROUTING: 

, il pkt è 

$I_{i-1} < I_i < I_{i+1}$ vede solo quello. (il resto nascosto)

in TOR: non faccio Pk encryption ∀ msg, ma signaling pkt con info routing + rilascia "labels" per eseguirlo.

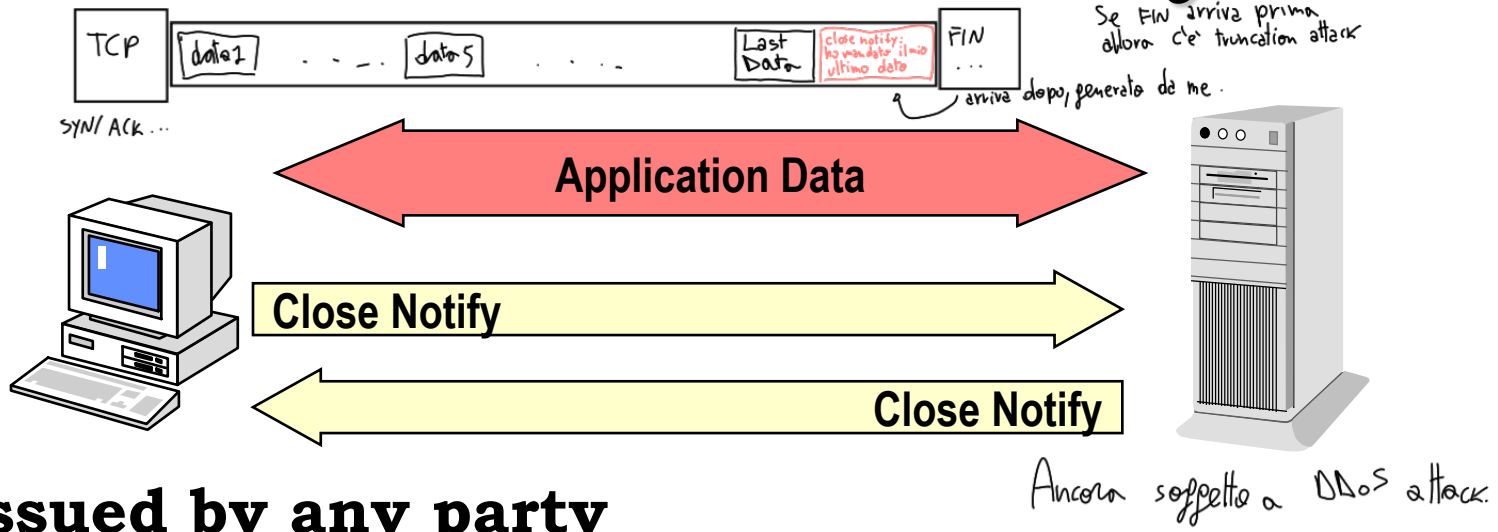
Truncation attack



An attacker may end connection at any time by sending a TCP FIN
Part of the intended information exchange is lost

How can Server and Client distinguish this from a transaction that "normally" completes?

Solution: Close Notify



➔ **Issued by any party**

- ⇨ Close Notify = Alert (warning level)
- ⇨ Half-close semantics (uni-direzionale, devo chiuderle in tutti e 2 i lati)

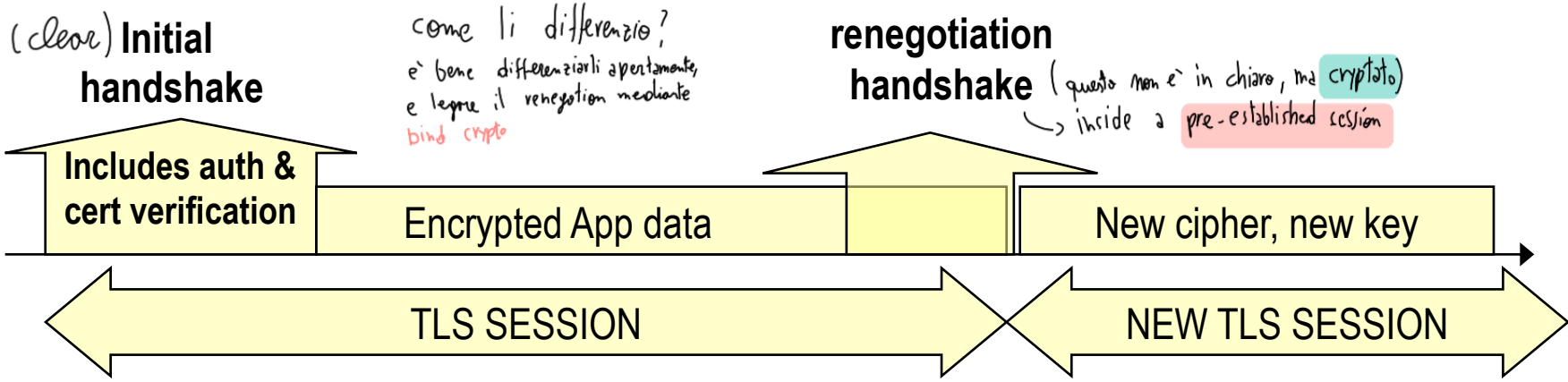
➔ **Informs that no more data will be transmitted**

➔ **A connection that ends abruptly without a Close Notify may be a truncation attack**

A remark: note the weakness of TLS against TCP DOS attacks in general!

Renegotiation

- **Renegotiation (re-handshake)**
  - Not "limited to" rekeying; more than session resumption
- **New full handshake (optionally) permitted.**
  - Encrypted via previously established ciphersuite
- **Purpose:**
  - Increase the security level (e.g. move to higher security cipher)
  - Authenticate Client
  - Anything else you may need
- **Renegotiation generates a NEW session ID**



Important: besides the fact that it is encrypted, renegotiation handshake otherwise NOT distinguishable from initial handshake (session = 0) – can this fact be (ab)used?

What about application data?

- **Renegotiation, when required, takes precedence over application data**
  - If applicable, server will buffer application data until renegotiation completed, then proceed on NEW TLS session
- **… renegotiation can be in the middle of an application layer transaction…**
  - Can we exploit this???

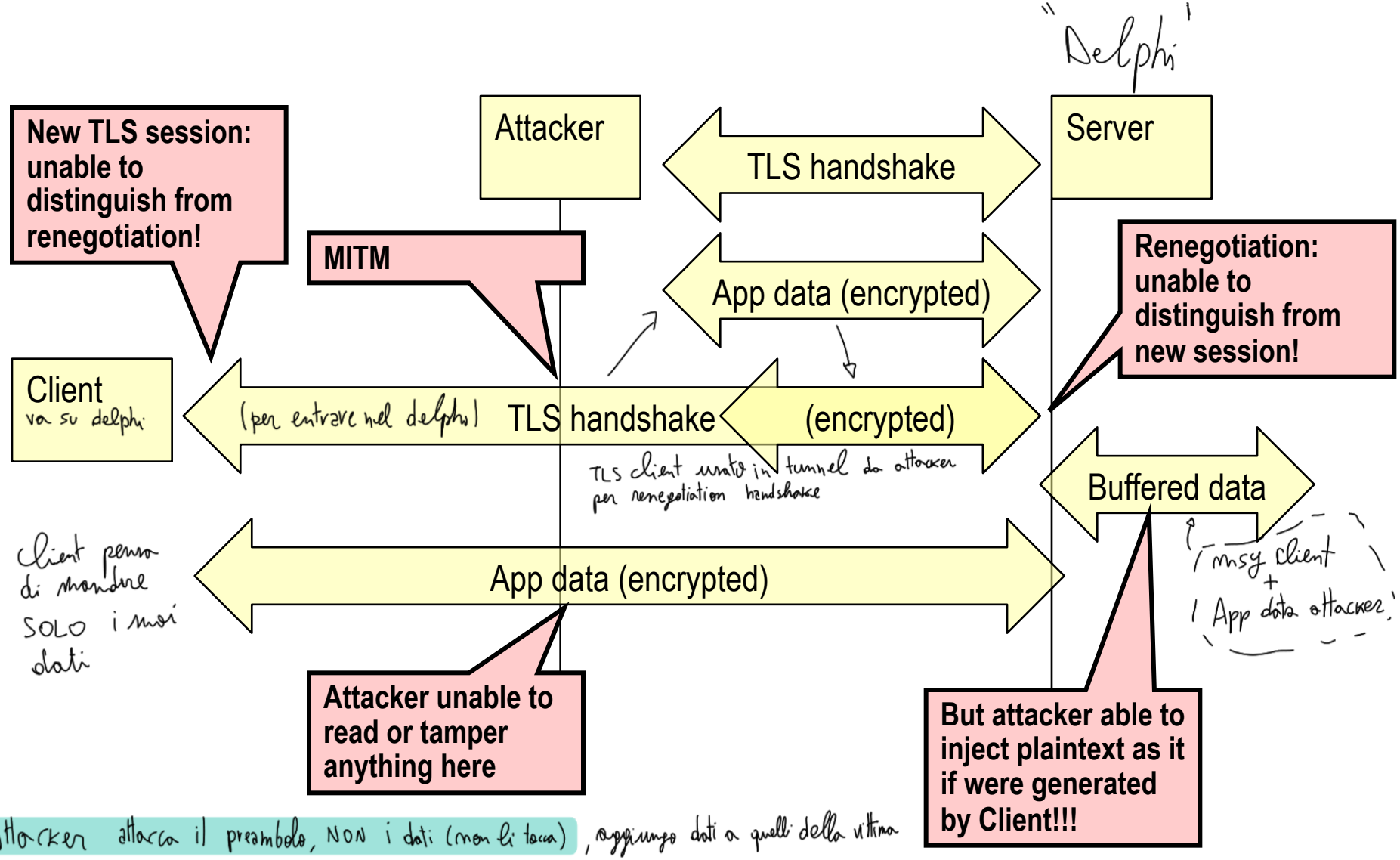
Renegotiation attack

(plaintext injection attack)

- **Discovered by Marsh Ray (PhoneFactor inc.)**
  - Aug-sep 2009; made public on begin of nov 2009
  - Limited effectiveness… but
- **Turned into practical credential stealing attack against twitter users by Anil Kurmus (ETH student)**
  - Mid november 2009
  - Twitter fixed it immediately (by disallowing renegotiation)
  - Ingenious application of such attack may apply to many other cases/scenarios
- **Came as a shock for TLS standard bodies**
  - Take-home golden rule: in security KEEP things SIMPLE!!

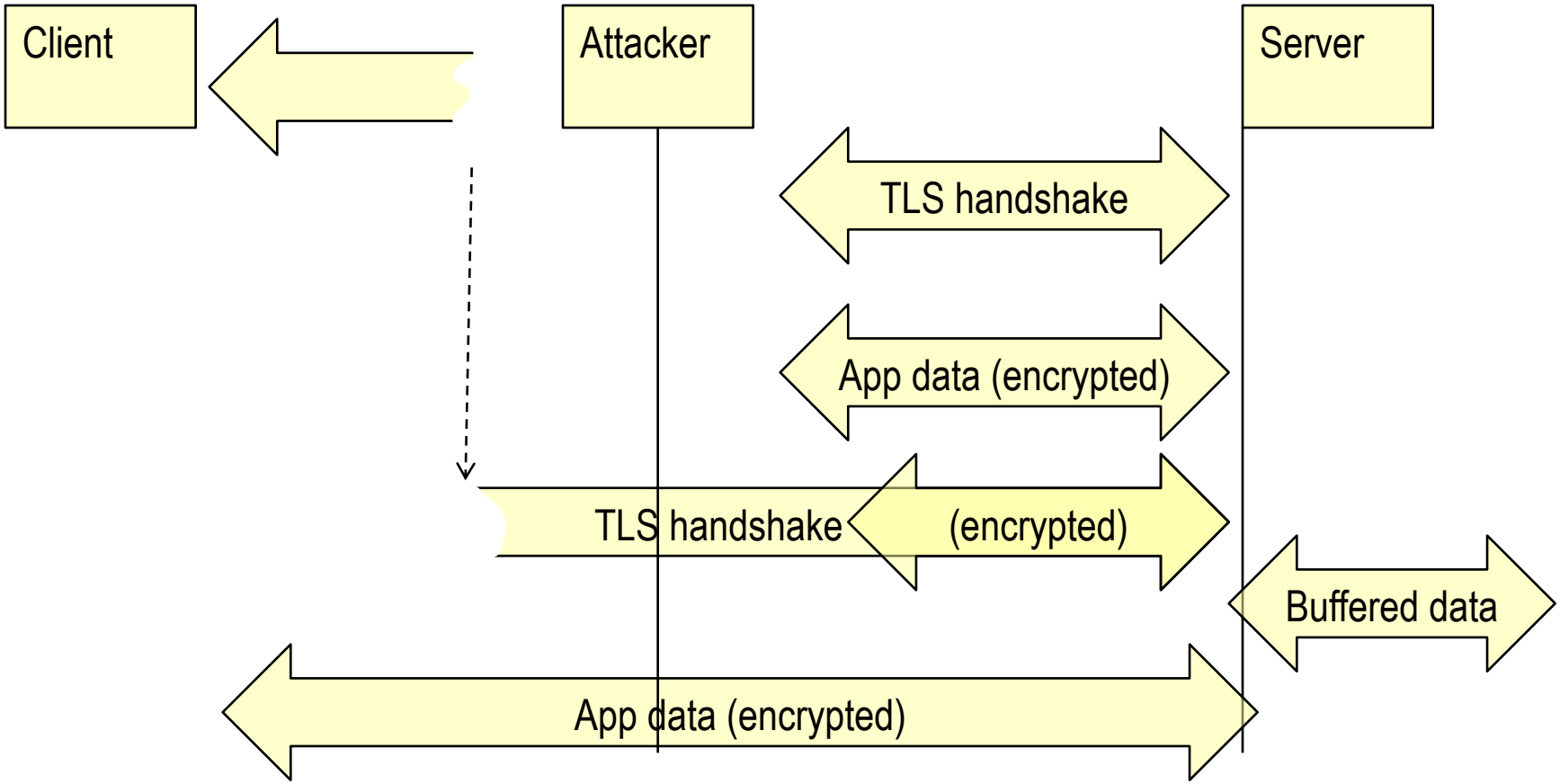
Funziona proprio perché i veri handshake non vengono differenziati

How it works



attacker attacca il preambolo, NON i dati (non li tocca), aggiungo dati a quelli della vittima

How it works / bis



Non trivial details (omitted here): i) how to properly buffer data? Ii) how to precisely trigger renegotiation? Both can be sorted out (indeed, attack was experimentally proved)

What can we do with prefix data injection?

➔ **Depends on application, of course! Example:**

➔ **Attacker:**

- ⇨ GET /pizza?toppings=pepperoni;address=attacker_address HTTP/1.1
- ⇨ X-Ignore-This: (no carriage return)

➔ **Victim:**

- ⇨ GET /pizza?toppings=sausage;address=victim_address HTTP/1.1
- ⇨ Cookie: victim_cookie

➔ **Result: glued requests:**

- ⇨ GET /pizza?toppings=pepperoni;address=attacker_address HTTP/1.1
- ⇨ X-Ignore-This: GET /pizza?toppings=sausage;address=victim_address HTTP/1.1
- ⇨ Cookie: victim_cookie

➔ **Server uses victim's account to send a pizza to attacker!**

Nicer example

(getting closer to actual Kurmus' twitter attack)

- **Attacker:**
  - POST /forum/send.php HTTP/1.0
  - Message = (no carriage return)

- **Victim:**
  - GET / HTTP/1.1 […]

- **Result: glued requests:**
  - POST /forum/send.php HTTP/1.0
  - Message = GET / HTTP/1.1 […]

- **Server discloses information generated by client upon request**

- **Anil Kurmus attack: conceptually similar**
  - Technicalities related to twitter API
  - Result (real world!): server posts a tweet (!) including the client credentials (base64 coded user/pass)!!!

Preventing TLS reneg attack

➔**Immediate solution: disable it** (renegotiation opzion)

➔**Standardization patch: renegotiation extension**

- ⇨(urgently) standardized in RFC 5746
  - ➔"TLS renegotiation Extension", February 2010
- ⇨First message: Extension empty
- ⇨Second message: Extension carries previous handshake verify (finished msg)
  - ➔Now it is trivial to detect attack

➔**TLSv1.3: forbids renegotiation!**

- ⇨«keep it simple» rule

And what about DTLS?

(presentano stessi difetti!)

DTLS vs TLS at a glance

- **RFC 4347 – Datagram TLS – April 2006**
  - TLS over UDP
- **DTLS design goal:**
  - Be as most as possible similar to TLS!
- **DTLS vs TLS**
  - TLS assumes orderly delivery
    - » DTLS: Sequence number explicitly added in record header
  - TLS assumes reliable delivery
    - » Timeouts added to manage datagram loss
  - TLS may generate large fragments up to 16384B
    - » DTLS includes fragmentation capabilities to fit into single UDP datagram, and recommends Path MTU discovery
  - TLS assumes connection oriented protocol
    - » DTLS connection = (TLS handshake – TLS closure Alert)