# Appunti su Attacchi TLS: BEAST e CRIME

Appunti dalla Lezione

7 novembre 2025

## Indice


**1 L'attacco BEAST (Browser Exploit Against SSL/TLS) 2**

1.1 Contesto: Modalità CBC e Initialization Vector (IV) . . . . . 2

1.2 La Vulnerabilità in TLS 1.0 . . . . . 2

1.3 Meccanismo dell'Attacco (Chosen Plaintext) . . . . . 3

1.4 Rendere Pratico l'Attacco: Chosen Boundary Attack . . . . . 4

**2 L'attacco CRIME (Compression Ratio Info-leak Made Easy) 5**

2.1 Il Concetto: Fuga di Informazioni dalla Compressione . . . . . 5

2.2 Meccanismo dell'Attacco (DEFLATE e LZ77) . . . . . 5

2.3 Impatto e Contromisure . . . . . 6

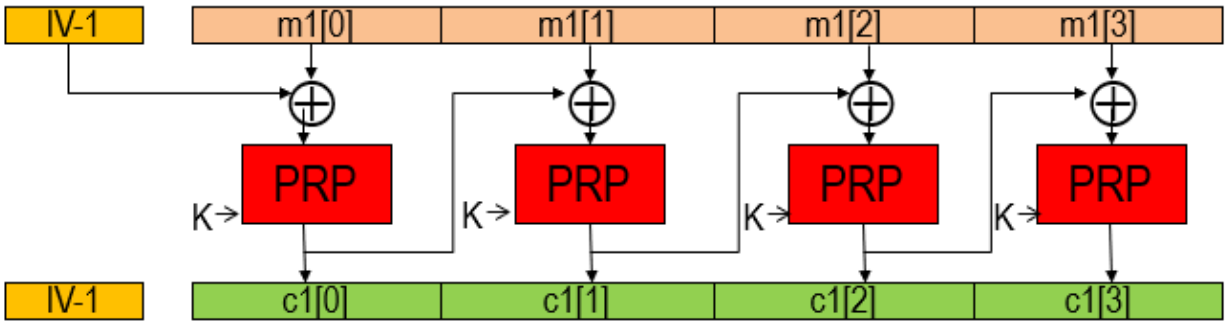
# 1 L'attacco BEAST (Browser Exploit Against SSL/TLS)

L'attacco BEAST, presentato da T. Duong e J. Rizzo nel 2011, è un attacco che sfrutta una vulnerabilità nel protocollo TLS v1.0. Nello specifico, colpisce l'uso di un Initialization Vector (IV) prevedibile nella modalità di cifratura Cipher Block Chaining (CBC).

## 1.1 Contesto: Modalità CBC e Initialization Vector (IV)

La modalità CBC è una modalità operativa per cifrari a blocchi. Per crittografare un messaggio $m$, suddiviso nei blocchi $m_1, m_2, ..., m_n$, la formula è:

$$C_i = E_K(m_i \oplus C_{i-1})$$



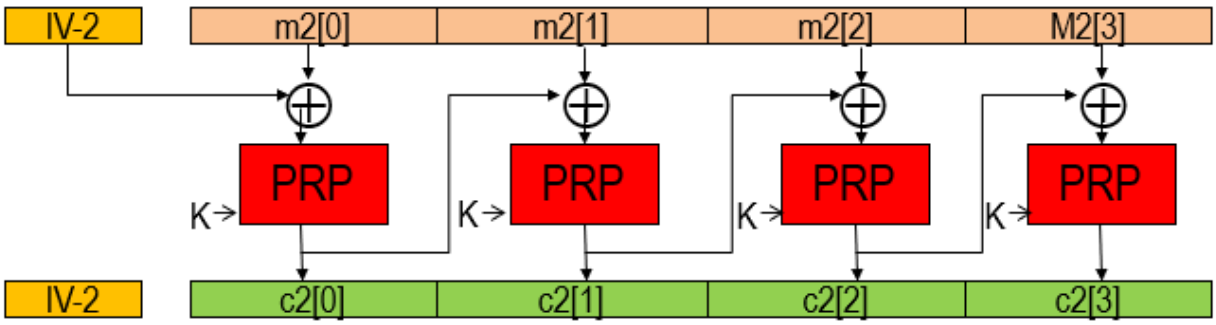

Dove $E_K$ è la funzione di cifratura con chiave $K$, e $C_i$ è il blocco di testo cifrato $i$.

Per il primo blocco ($m_1$), non esiste un $C_0$ precedente. Si utilizza quindi un **Initialization Vector (IV)**:

$$C_1 = E_K(m_1 \oplus IV)$$

Per garantire la sicurezza (sicurezza semantica):

- L'IV deve essere diverso per ogni messaggio crittografato con la stessa chiave. Altrimenti, due messaggi identici produrrebbero lo stesso testo cifrato.
- L'IV non deve essere solo diverso, ma anche **imprevedibile** per un attaccante.



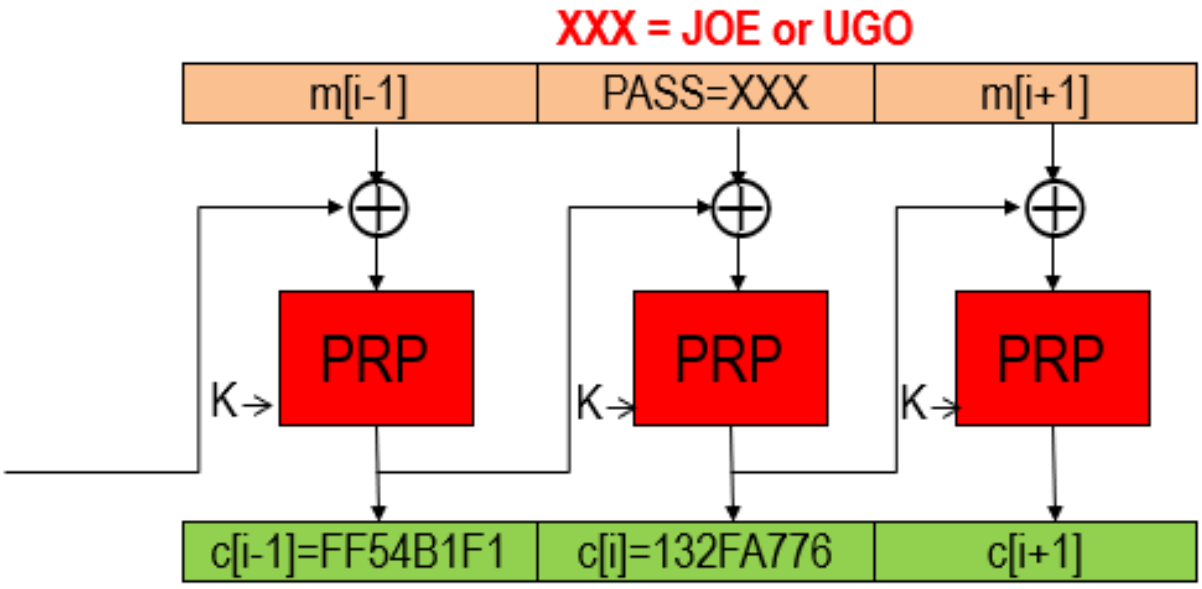

## 1.2 La Vulnerabilità in TLS 1.0

Il protocollo TLS 1.0 corregge la prima problematica ma fallisce sulla seconda. In TLS 1.0, l'IV per un nuovo messaggio non è generato casualmente, ma è semplicemente **l'ultimo blocco di testo cifrato ($C_n$) del messaggio precedente**.

Poiché l'attaccante può osservare il traffico di rete, egli conosce $C_n$ e quindi può *prevedere* perfettamente l'IV del messaggio successivo. Questa prevedibilità è il cuore della vulnerabilità sfruttata da BEAST. Le versioni successive, TLS 1.1+, hanno corretto questo problema utilizzando un IV esplicito e casuale per ogni messaggio.

## 1.3 Meccanismo dell'Attacco (Chosen Plaintext)

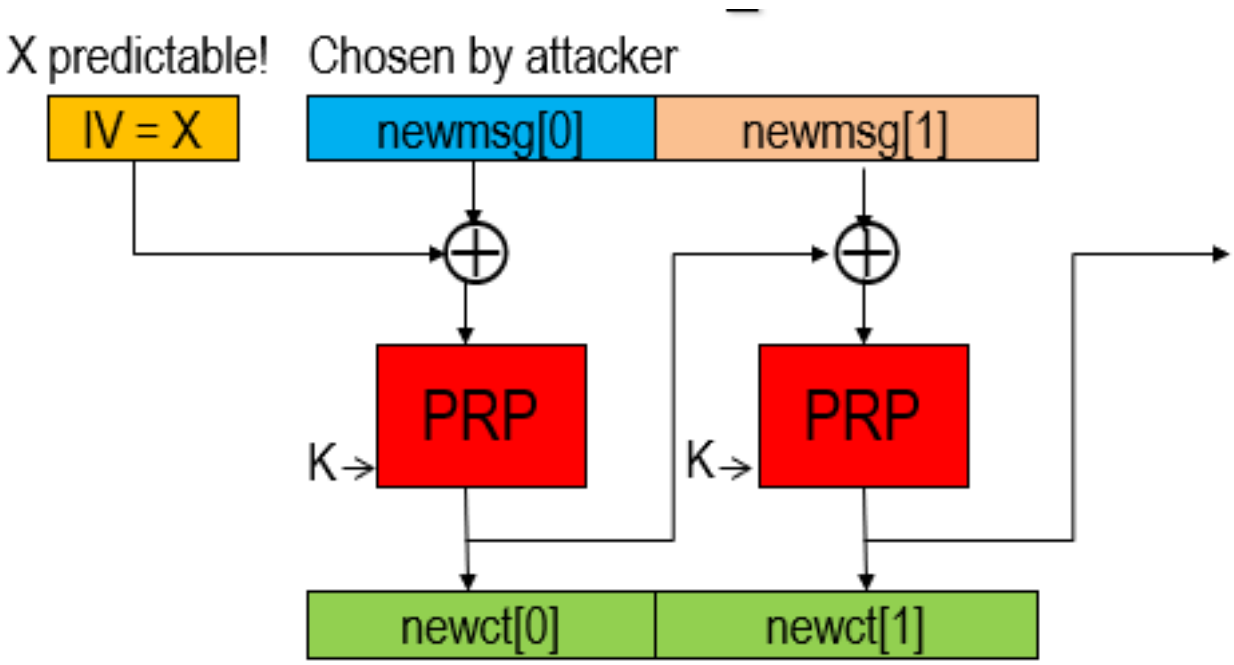
BEAST è un attacco di tipo **Chosen Plaintext Attack** (CPA). L'attaccante deve essere in grado di far crittografare al sistema dei dati da lui scelti (plaintext scelti) e osservare il relativo ciphertext. Questo è reso possibile da codice malevolo (es. JavaScript) in esecuzione nel browser della vittima.




Supponiamo che l'attaccante voglia decifrare un blocco di messaggio segreto $m_i$ (es. un cookie di sessione), di cui conosce il ciphertext $C_i$ e il blocco di ciphertext precedente $C_{i-1}$.

- L'attaccante conosce: $C_i$ e $C_{i-1}$.
- L'attaccante vuole trovare: $m_i$.
- L'attaccante sa che: $C_i = E_K(m_i \oplus C_{i-1})$.

L'attacco procede così:



1. L'attaccante fa una **supposizione (guess)** per il blocco $m_i$, chiamiamola $PW_{GUESS}$.
2. L'attaccante predice l'IV del prossimo messaggio, che sa essere $X = C_n$ (l'ultimo blocco precedente).
3. L'attaccante "convince" l'implementazione (es. il browser) a crittografare un nuovo messaggio $newmsg[0]$ da lui costruito ad arte.
4. Il blocco scelto è: $newmsg[0] = X \oplus C_{i-1} \oplus PW_{GUESS}$.
5. Il sistema calcola il nuovo ciphertext $newct[0]$ come:

$$newct[0] = E_K(newmsg[0] \oplus IV_{next})$$

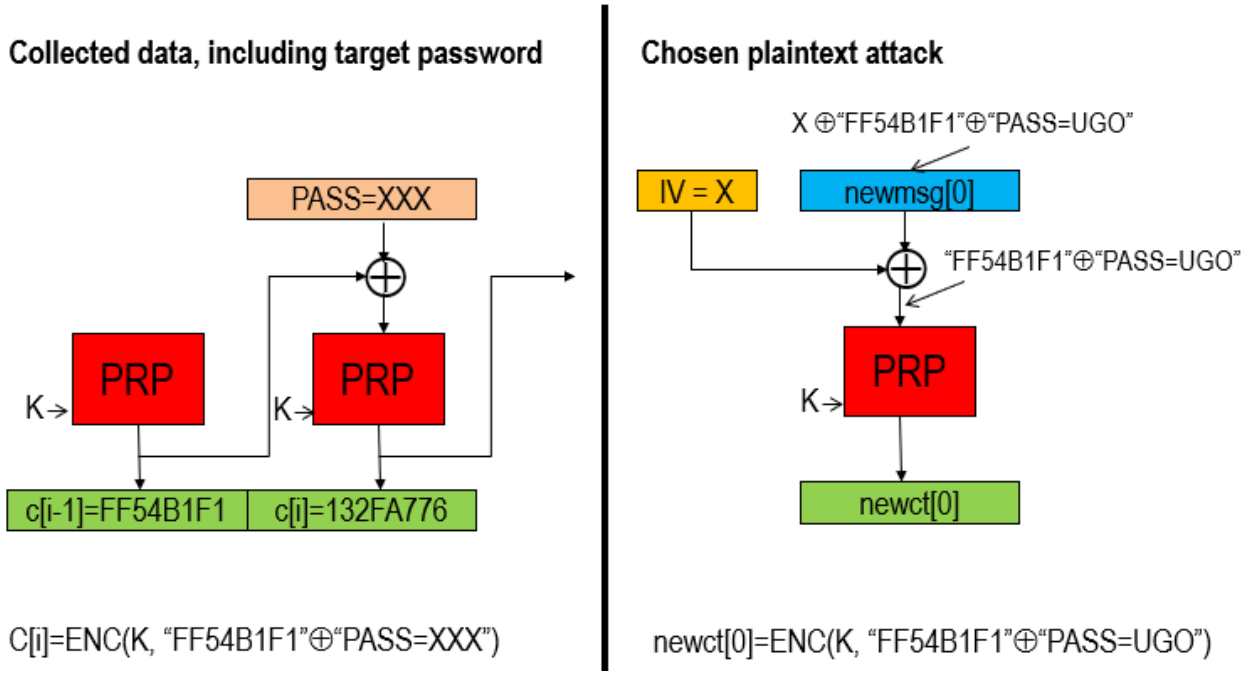
   Sostituendo $IV_{next} = X$ e la nostra $newmsg[0]$:

$$newct[0] = E_K((X \oplus C_{i-1} \oplus PW_{GUESS}) \oplus X)$$

6. Grazie alla proprietà dell'operatore XOR ($A \oplus A = 0$), l'IV $X$ si annulla:

$$newct[0] = E_K(C_{i-1} \oplus PW_{GUESS})$$

7. L'attaccante ora confronta il $newct[0]$ che ha ottenuto con il $C_i$ originale che voleva decifrare.




**Risultato:** Se $newct[0] == C_i$, significa che $E_K(C_{i-1} \oplus PW_{GUESS}) == E_K(C_{i-1} \oplus m_i)$. Ne consegue che la supposizione era corretta: $PW_{GUESS} = m_i$. Se è diverso, l'attaccante prova con una nuova supposizione.

## 1.4 Rendere Pratico l'Attacco: Chosen Boundary Attack

Questo attacco sembra impraticabile. Se si usa AES, un blocco è di 128 bit (16 byte). Provare tutte le $2^{128}$ possibili supposizioni è impossibile.

Qui entra in gioco il "colpo di genio" di Duong e Rizzo: il **Chosen Boundary Attack**. L'obiettivo non è indovinare tutti i 16 byte del blocco in una volta, ma un solo byte alla volta.

1. L'attaccante non controlla solo il blocco $newmsg[0]$, ma può inserire dati di "preambolo" (padding) prima di esso. 2. L'attaccante inietta una quantità di dati tale da **allineare il blocco** AES (il "confine" del blocco) in modo che il blocco che deve indovinare contenga 15 byte di dati noti (scelti da lui) e solo *1 byte* di segreto. 3. Ad esempio, se il segreto è `PASSWD=ALICE`, l'attaccante inserisce un preambolo per far sì che il blocco da indovinare sia `[preambolo...]PASSWD=A`. 4. Ora l'attaccante deve indovinare solo l'ultimo byte. Il numero di tentativi si riduce da $2^{128}$ a $2^8 = 256$. 5. Una volta trovato il primo byte ('A'), l'attaccante rimuove un byte dal suo preambolo. Il nuovo blocco da indovinare diventa `[preambolo..]PASSWD=AL`. 6. L'attaccante ripete l'attacco, indovinando il secondo carattere (di nuovo, al massimo 256 tentativi).

La complessità dell'attacco diventa **lineare** con la lunghezza $N$ del segreto ($256 \times N$), non esponenziale ($256^N$).

# 2 L'attacco CRIME (Compression Ratio Info-leak Made Easy)

CRIME, presentato sempre da Duong e Rizzo nel 2012, è un attacco completamente diverso che sfrutta la **compressione** dei dati.

A differenza di BEAST, che è specifico per CBC, CRIME funziona **indipendentemente dal cifrario utilizzato** (AES, RC4, ecc.). La sua vulnerabilità risiede nel fatto che la compressione, se applicata *prima* della crittografia, fa trapelare informazioni sul plaintext.

## 2.1 Il Concetto: Fuga di Informazioni dalla Compressione

Il principio è semplice: un algoritmo di compressione (come DEFLATE) riduce la dimensione dei dati trovando e sostituendo sequenze ripetute.

- Se si comprime `ABCDEF`, che non ha ripetizioni, la dimensione rimane 6 byte.
- Se si comprime `AAAABC`, che ha ripetizioni, diventa qualcosa come `4ABC`, che è più corto (4 byte).

L'attaccante non può vedere il contenuto compresso (perché è crittografato), ma può osservare la **lunghezza totale del pacchetto crittografato**. Poiché la crittografia (in genere) non cambia la dimensione dei dati, la lunghezza del ciphertext riflette la lunghezza del *compressed text*.

Un attaccante può quindi dedurre se una certa stringa ha causato una buona compressione (risultato più corto) o no (risultato più lungo).

## 2.2 Meccanismo dell'Attacco (DEFLATE e LZ77)

L'attacco CRIME sfrutta l'algoritmo LZ77 (parte di DEFLATE). LZ77 sostituisce le stringhe ripetute con puntatori (offset, lunghezza).

Esempio: `...Cookie: twid=flavia...`

L'attaccante, come in BEAST, può iniettare dati scelti (es. in una richiesta HTTP) che vengono inviati nella stessa richiesta (e quindi nello stesso flusso di compressione) del segreto della vittima (es. il cookie).

L'attacco (semplificato) procede così per indovinare il cookie `twid=flavia`:

1. L'attaccante inietta una supposizione per il primo carattere, es. `...twid=a...`.
2. Il messaggio completo da comprimere è: `[...twid=a...] [Cookie: twid=flavia]`.

3. LZ77 non trova corrispondenze. La lunghezza compressa è $L$.

4. L'attaccante prova con un'altra supposizione, `...twid=f...`.

5. Il messaggio completo è: `[...twid=f...] [Cookie: twid=flavia]`.

6. LZ77 trova una corrispondenza! Sostituisce `twid=f` nel cookie con un puntatore, es. `...Cookie: (-24,6)lavia` (dove 24 è l'offset e 6 la lunghezza).

7. La stringa compressa è ora **più corta** di prima.

8. L'attaccante osserva che la lunghezza totale del pacchetto crittografato è diminuita e conclude che la sua supposizione ('f') era corretta.

9. L'attaccante passa al carattere successivo, provando `...twid=fa...`, `...twid=fb...`, ecc. fino a `...twid=fl...`, che causerà un'ulteriore riduzione di lunghezza.

## 2.3 Impatto e Contromisure

Come BEAST, questo attacco recupera il segreto un carattere alla volta, con una complessità **lineare** rispetto alla lunghezza del segreto.

L'attacco è stato dimostrato contro TLS e SPDY (un precursore di HTTP/2).

La contromisura più efficace e ampiamente adottata è stata semplicemente quella di **disabilitare la compressione a livello TLS/SSL**. La vulnerabilità non è nel protocollo TLS in sé, ma nell'interazione tra compressione e crittografia. Varianti successive come BREACH hanno dimostrato lo stesso attacco a livello di compressione HTTP.