

# SSL/TLS

Differenze tra le varie versioni di SSL e TLS,  
Attacchi,  
Security Assessment

# Protocollo SSL e architettura

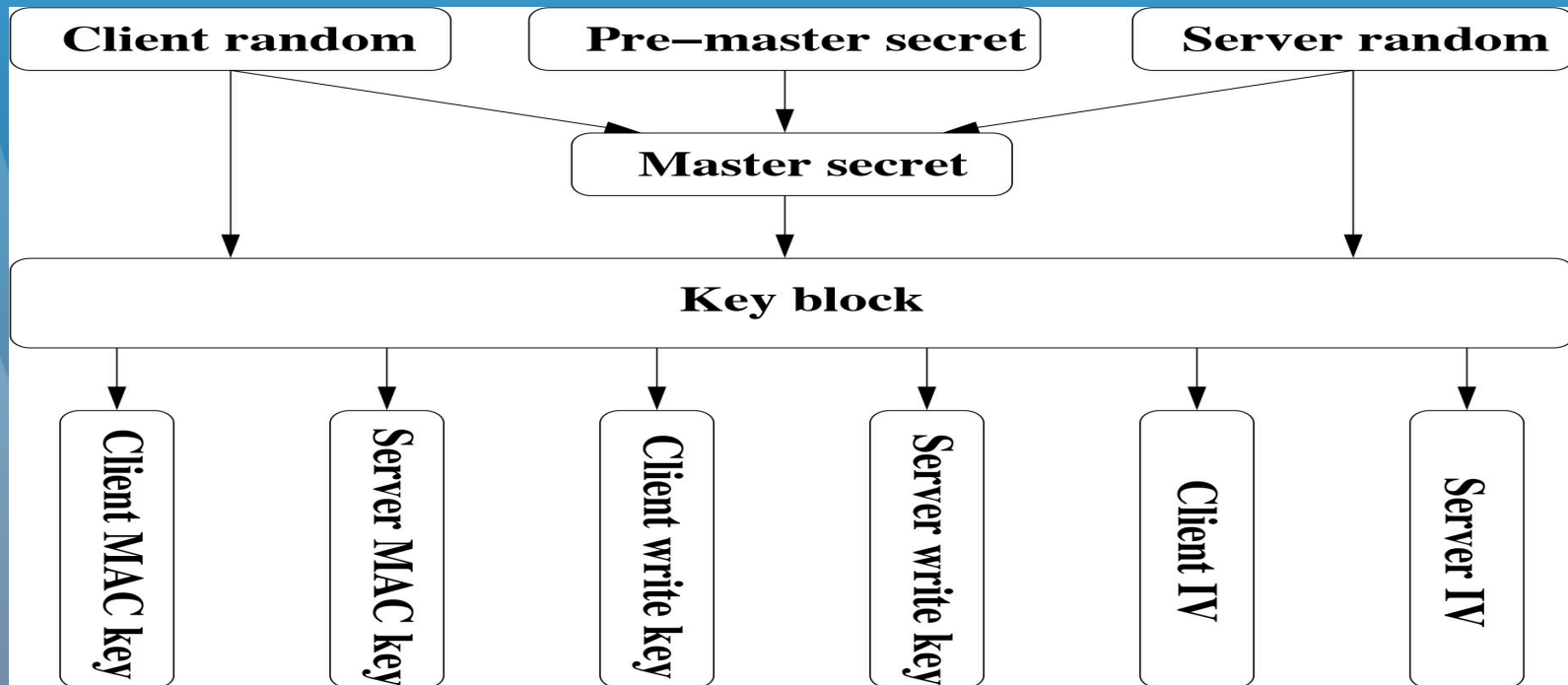
- Utilizzato per garantire :
  1. Una comunicazione privata
  2. Autenticazione del server e del client (opzionale)
  3. Integrità dei dati
- Si presenta come un ulteriore layer nel modello ISO/OSI che si colloca tra il livello di trasporto e il livello applicazione.
- E' composto da due sottolivelli:
  1. Handshake
  2. Record

# Sottolivelli SSL

- Il primo è utilizzato per stabilire una connessione sicura tra due peer (tipicamente un client ed un server). L'ultimo messaggio di Handshake sarà il primo messaggio protetto.
- Il secondo per la trasmissione dei dati.

# Creazione delle chiavi

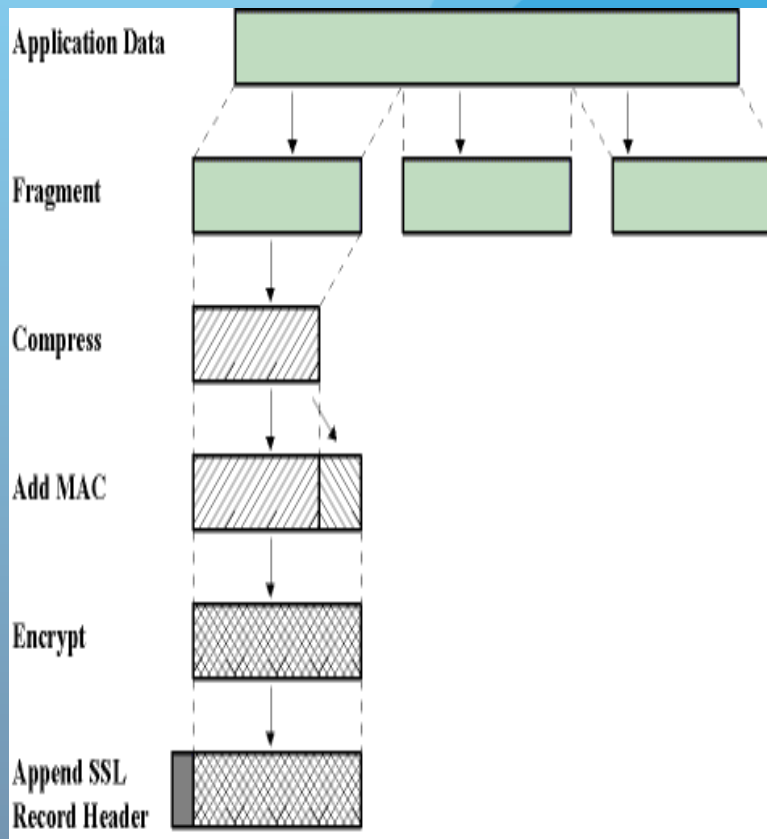
- Prima di utilizzare tecniche di crittografia a chiave segreta bisogna stabilire il “keying material”.
- Inizialmente bisogna stabilire un pre\_master\_secret di 48-byte. Per stabilirlo è possibile usare degli algoritmi crittografici a chiave pubblica come RSA, Diffie-Hellman e FORTEZZA(non supportato in TLS 1.0 e 1.1).
- Una volta determinato il pre\_master\_secret si deve generare un master\_secret. Quest’ ultimo si costruisce applicando MD5, SHA-1 e utilizzando ServerHello.random e ClientHello.random.



# Creazione delle chiavi

- A partire dal master\_secret sarà generato il key\_block (procedendo iterativamente finché se ne ottenga uno di lunghezza sufficiente). In TLS la creazione del key\_block è dovuta ad una pseudo random function(PRF).
- Il key block sarà suddiviso in MAC secrets, chiavi crittografiche e IVs(utilizzati se è in uso un block cipher in CBC mode).

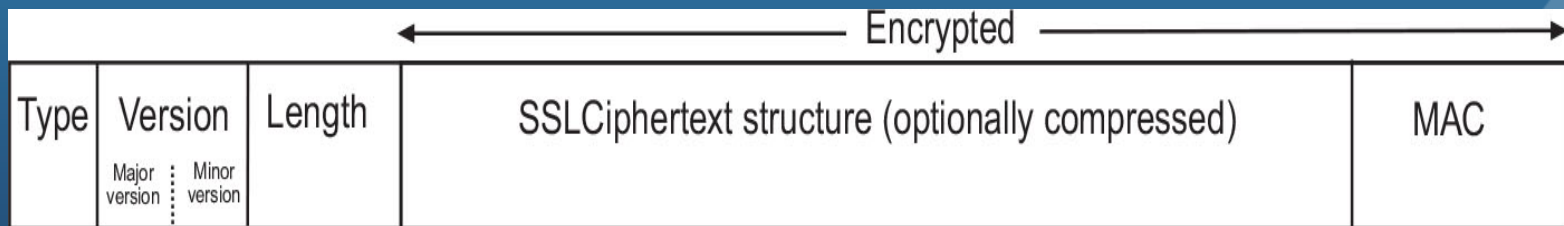
# SSL Record Processing



1. Inizialmente i dati dal livello più alto vengono frammentati in blocchi ognuno dei quali sarà impacchetato in un SSLPlaintext
2. I pacchetti ora verranno opzionalmente compressi in SSLCompressed.
3. Il pacchetto verrà protetto tramite la cipher suite scelta e diventerà un SSLCiphertext. In questa fase vengono effettuate due cose fondamentali:
  - Creazione del MAC
  - Dopo aver appeso il MAC alla struttura SSLCompressed si procede con l' Encryption. Se viene usato uno Stream Cipher non saranno necessari ne padding ne IV, al contrario se viene utilizzato un block cipher è necessario il padding e in alcuni casi anche l' IV(usato per crittografare la prima struttura).

# SSL Record Processing

4. Per terminare il processo viene appeso un header all' SSLCiphertext che rende la struttura un SSL Record. L' header comprende tre campi:
- Type, che ci indica se si tratta di un SSL Change Cipher Spec Protocol, di un Alert Protocol, di un Handshake Protocol o di un Application Data Protocol
  - Version, che indica che tipo di versione di SSL si sta usando
  - Length, lunga 16-bit.



# Protocolli usati in SSL

- SSL Record Protocol
- SSL Handshake Protocol
- SSL Change Cipher Spec Protocol
- SSL Alert Protocol
- SSL Application Data Protocol



# SSLv1.0 ed SSLv2.0

- SSLv1.0 fu sviluppato da Netscape nel 1994 ma non venne mai rilasciato. Circolò solo all' interno della Netscape Communications.
- Non prevedeva la protezione dell' integrità dei dati e non usava numeri di sequenza(replay attack possibili).
- Come stream cipher utilizzava RC4, considerato vulnerabile.
- Venne aggiunto in seguito un semplice cyclic redundancy check(CRC) come checksum invece di funzioni hash crittograficamente forti.
- Questi e altri problemi vennero risolti nel 1994 con la presentazione di SSLv2.0
- CRC venne sostituito da MD5 che veniva considerato sicuro a quel tempo.

# Da SSLv2.0 a SSLv3.0

- SSLv2.0 permette al client ed il server di avere solo un certificato, questo implica che la certificazione doveva essere firmata da una root CA. SSLv3.0 permette invece di avere delle “certificate chains” di lunghezza arbitraria.
- SSLv2.0 utilizza la stessa chiave sia per l’ autenticazione del messaggio che per l’ encryption mentre in SSLv3.0 sono utilizzate chiavi diverse.
- Per la generazione dei MAC(message authentication code) SSLv2.0 utilizza MD5 mentre in SSLv3.0 vengono usate sia MD5 che SHA-1 per avere una costruzione più sofisticata.

# TLS Overview

- Nasce dall' esigenza di trovare uno standard
- La struttura del protocollo TLS è identica a quella del protocollo SSL.
- Abbiamo tre versioni di questo protocollo:
  1. TLS 1.0 presentato nel 1999
  2. TLS 1.1 presentato nel 2006
  3. TLS 1.2 presentato nel 2008

# Da SSLv3.0 a TLS 1.0

- Retrocompatibile con SSLv3.0.
- La principale differenza tra SSL e TLS risiede nella generazione delle chiavi. TLS usa infatti una funzione denominata PRF.
- TLS 1.0 supporta gli stessi cipher suites di SSLv3.0 fatta eccezione per i cipher suites appartenenti alla famiglia FORTEZZA.
- Per l'autenticazione dei messaggi vengono utilizzati gli HMAC.
- La gestione dei certificati presenta delle novità.

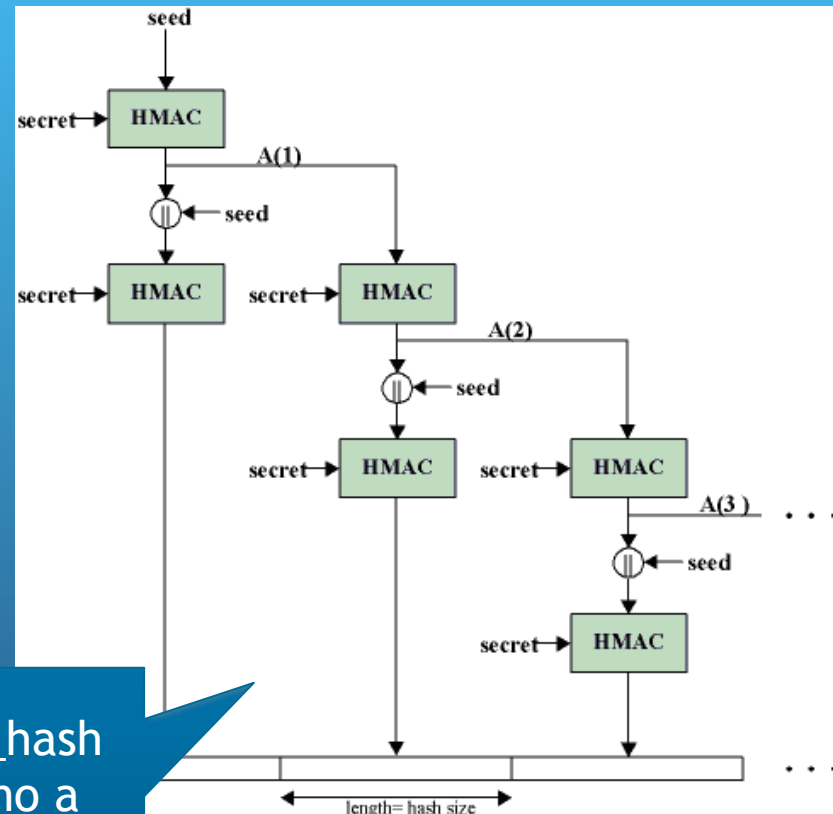
# TLS PRF

- TLS PRF prende in input un secret, un seed ed un label di identificazione e genera come output una arbitraria sequenza di bit. Viene utilizzato per creare il key\_block.
- Per renderla più sicura possibile combina due funzioni hash crittografiche. La funzione si può ritenere segreta finché almeno una delle due funzioni hash è sicura.
- Inoltre abbiamo un' ulteriore funzione di expansion data chiamata P\_hash che rende l' algoritmo ancora più sicuro.
- PRF prenderà quindi il secret lo suddividerà in due parti e su ognuna di queste verrà applicato P\_hash dell' algoritmo hash scelto.
- Se scegliamo MD5 e SHA-1 allora avremo  $\text{PRF}(\text{secret}, \text{seed}, \text{label}) = \text{P\_MD5}(S1, \text{label}+\text{seed}) \text{ XOR } \text{P\_SHA-1}(S2, \text{label}+\text{seed})$ .

# TLS PRF

- Con TLS 1.2 viene abbandonata la scelta di utilizzare MD5 assieme ad SHA-1 a favore di un'unica funzione hash più sicura.
- In TLS 1.2 la funzione hash dovrà essere parte della cipher suite (come ad esempio SHA256).

La funzione di espansione P\_hash sarà ripetuta iteramente fino a raggiungere una dimensione sufficiente



# Costruzione degli HMAC

- Un HMAC richiede una funzione crittografica di hash ed una chiave segreta K. Quindi avremo:
  - $\text{HMAC}(m) = h(k \text{ XOR opad} || h(k \text{ XOR ipad} || m))$
- Alcuni elementi possono essere precalcolati così da rendere più efficiente il calcolo dell' HMAC.
- SSLv3.0 utilizzava un MAC basato su una precedente versione di HMAC ed utilizzava MD5 e SHA mentre con TLS 1.0 possiamo adoperare algoritmi crittografici di hashing a nostra scelta.

# Gestione dei certificati

- SSLv3.0 richiedeva certificate chain complete mentre TLS 1.0 accetta certificate chains che vanno indietro ad intermediari. Questo semplifica e velocizza la validazione.
- TLS 1.0 supporta solo 4 tipi di certificati:
  1. RSA signing
  2. DSA signing
  3. RSA signing with fixed Diffie-Hellman key exchange
  4. DSA signing with fixed Diffie-Hellman key exchange



# TLS 1.0 VS TLS 1.1

- In risposta ad una vulnerabilità trovata sul padding CBC l' IV implicito viene sostituito da un IV random esplicito.
- La gestione degli errori di padding ora usa l' alert `bad_record_mac` anzichè `decryption_failed` per protezione contro gli attacchi CBC.
- Reintegrati i tipi di certificati con firma FORTEZZA.
- Tutti le cipher suite di grado export non possono essere usati in questa versione.
- Sessioni TLS riesumabili anche in casi di chiusura prematura della sessione(non della connessione!).
- Nuovi registri creati da IANA per rendere TLS più flessibile. Se un parametro va aggiunto o cambiato non sarà più necessario cambiare il protocollo ma sarà sufficiente aggiornare il registro.

# TLS 1.1 VS TLS 1.2

- TLS 1.2 utilizza una funzione PRF differente che non si basa più su MD5 e SHA-1 ma su un solo algoritmo crittografico scelto tra quelli presi nella cipher suite.
- In alcuni casi può essere utile dialogare con frammenti di una lunghezza minore rispetto alla massima possibile. Per questo tramite l'estensione di tipo `max_fragment_length` il client può negoziare una lunghezza massima diversa.
- Quando è richiesta l'autenticazione del client quest'ultimo deve inviare un certificato al server. Questo può risultare dispendioso. In TLS 1.2 possiamo inviare un URL tramite il quale è possibile recuperare il certificato.

# TLS 1.1 VS TLS 1.2

- In SSL/TLS il Server non sa quali sono le trusts CAs del client. Questo può causare che l' handshake si ripeta senza successo più volte. Usando l' estensione di tipo `trusted_ca_keys` possiamo trasmettere questa informazione da client e server per evitare il problema sopracitato.
- Si può scegliere se utilizzare un Truncated HMAC.
- In TLS 1.2 è possibile, inoltre, mandare tramite il ClientHello ed il ServerHello dei dati supplementari.
- Tramite l' estensione `cert_type` il client può indicare il supporto per altri tipi di certificati diversi dallo standard X509.

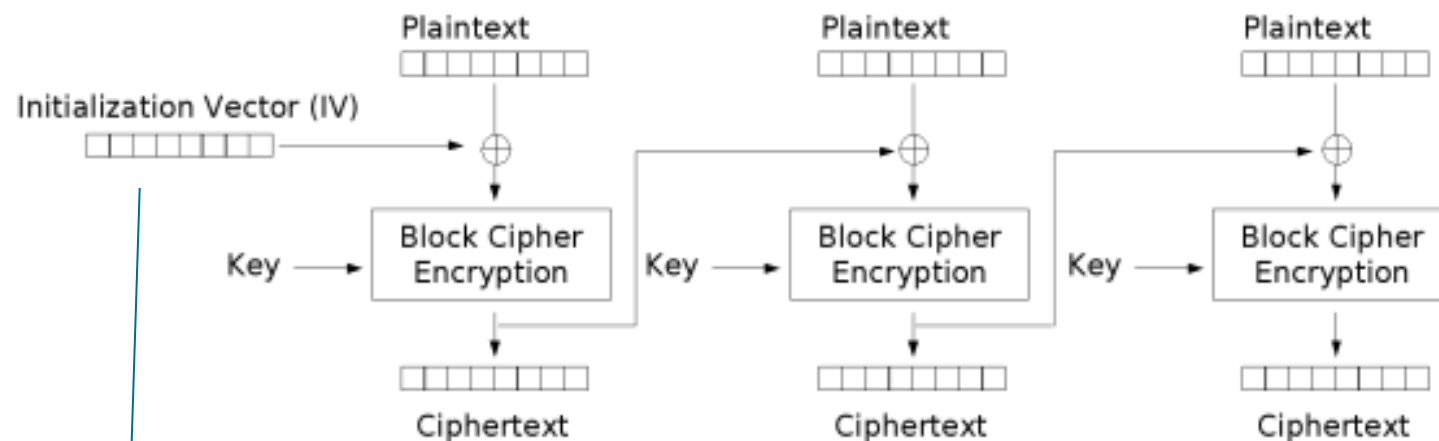
# TLS 1.1 VS TLS 1.2

- L' utilizzo delle curve ellittiche nell' uso della crittografia a chiave pubblica permette di usare chiavi più corte mantenendo inalterata la sicurezza. In TLS 1.2 abbiamo il supporto per l' ECC (Elliptic Curve Cryptography).
- Come sappiamo lo scambio di chiavi Diffie-Hellman è suscettibile al MITM. L' utilizzo di password è il metodo più semplice per ovviare a questo problema. Ma anche in questo caso è possibile essere attaccati tramite l' uso di dizionari. Per evitare anche questo tipo di attacco in TLS 1.2 si può utilizzare il protocollo SRP (Secure Remote Password) durante l' autenticazione del client nell' handshake.
- Infine per recuperare delle sessione in TLS 1.2 si può utilizzare l' estensione di tipo SessionTicket.

# Attacchi noti a SSL/TLS

- BEAST
- CRIME
- BREACH

# BEAST



Cipher Block Chaining (CBC) mode encryption

Fino a TLS 1.0 l' IV del primo blocco del primo messaggio é il cifrato dell' ultimo blocco dell' ultimo messaggio inviato. Questo rende l' IV predicibile ed é la base di questo attacco.

# BEAST

- Se l'utente sta utilizzando CBC e l'aggressore è a conoscenza dell'ultimo blocco cifrato dell'ultimo messaggio allora può provare ad indovinare il testo in chiaro.
- Questo é computazionalmente difficile, ma se l'aggressore è in grado d'inserire del testo nel plaintext dell'utente il record TLS sarà occupato in parte dal testo iniettato ed in parte dal secret.
- Supponiamo di avere un cifrato di 16byte occupandone 15 con il testo iniettato il guess da fare sarà più semplice.
- Se l'aggressore può far fare all'utente più richieste al server tramite un guess-checking attack può essere relativamente facile accedere a dati sensibili(come web cookie o token).

# CRIME

29	9.200197000	127.0.0.1	127.0.0.1	SSLv3	141 Change Cipher Spec
30	9.201664000	127.0.0.1	127.0.0.1	SSLv3	652 Application Data, /
31	9.202208000	127.0.0.1	127.0.0.1	SSLv3	833 Application Data, /
35	9.204280000	127.0.0.1	127.0.0.1	SSLv3	176 Client Hello
37	9.204682000	127.0.0.1	127.0.0.1	SSLv3	73 Alert (Level: Fatal)
43	14.203056000	127.0.0.1	127.0.0.1	SSLv3	103 Encrypted Alert

Frame 30: 652 bytes on wire (5216 bits), 652 bytes captured (5216 bits) on interface 0  
Ethernet II, Src: 00:00:00\_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00\_00:00:00 (00:00:00:00:00:00)  
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)  
Transmission Control Protocol, Src Port: 44761 (44761), Dst Port: https (443), Seq: 327, Ack: 1105  
Secure Sockets Layer

SSLv3 Record Layer: Application Data Protocol: http

Content Type: Application Data (23)

Version: SSL 3.0 (0x0300)

Length: 32

Encrypted Application Data: 65dfac90c155d52a5f971a4a690b912cb2eef0d11d7c278e...

SSLv3 Record Layer: Application Data Protocol: http

Content Type: Application Data (23)

Version: SSL 3.0 (0x0300)

Length: 544

Encrypted Application Data: 401crr5...3a0993629...


SSL/TLS non nasconde  
la length dei messaggi!



L'aggressore monitora il traffico della vittima



La vittima si autentica tramite HTTPS e negozia una compressione TLS con il server



Injection Javascript per forzare il browser della vittima a mandare ripetute richieste al server

# CRIME

- Molti server supportano la compressione TLS per ridurre la congestione o i tempi di caricamento. Il principale metodo di compressione TLS é DEFLATE.
- DEFLATE è composto da 2 sottoalgoritmi:
  1. LZ77, usato per eliminare la ridondanza di suqenze
  2. Huffman Coding, usato per eliminare la ridondanza di simboli
- Provando ad indovinare il secret ed osservando la length del messaggio inviato se quest' ultima non è più grande del solito il guess è corretto.
- L' attacco può essere mitigato disabilitando la compressione TLS.

# BREACH

- Si basa sull'analisi delle HTTP response compresse.
- Anche la compressione HTTP utilizza l'algoritmo DEFLATE che in questo caso comprime solo il body e non l'header.

## Fattibilità

- L'applicazione deve supportare la compressione HTTP.
- La response deve contenere i token o i secret nel body

## Requisiti:

1. Monitoraggio del traffico della vittima
2. Forzare la vittima a visitare una pagina web controllata dall'aggressore (injection di un iframe, phishing email o altro)

## Contromisure

Per mitigare l'attacco si può disabilitare la compressione HTTP (ma ciò è poco efficiente). Altre soluzioni possibili sono il length hiding o il masking secrets.

# Security Assessment

- Le più importanti raccomandazioni quando si usa SSL/TLS sono:
  1. Non supportare SSLv2
  2. Disabilitare cifrari deboli e preferire protocolli nuovi
  3. Usare chiavi private per i certificati di almeno 2048 bit
  4. Mitigare le problematiche conosciute
  5. Testing del web server

# Creazione di un web server sicuro

- Installare un web server(per l' esempio è stato utilizzato nginx) ed impostare nel file di configurazione dei siti abilitati la modalità ssl selezionando la porta in ascolto. Dopo di che settare le misure di sicurezza necessarie.

SSLv2 non viene supportato

```
ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2;  
#ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv3:+EXP;  
  
#BEAST Mitigation with strong cipher but not RC4  
ssl_ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL:!MD5:!DSS;  
  
#BEAST Mitigation with RC4  
#ssl_ciphers RC4:HIGH:!aNULL:!MD5;  
  
#Turn off gzip to protect against BREACH, this method brings low performance  
gzip on;
```

Vengono scelti dei cifrari non suscettibili all' attacco BEAST

Disattiviamo la compressione HTTP per difenderci da BREACH

La compressione TLS, utilizzata in CRIME, è disabilitata di default nelle nuove versioni di apache e nginx

# Test del web server

- E' possibile testare il nostro server sicuro con numerosi tool. Ad esempio:
  - i. Con sslyze o sslscan possiamo fare un test delle vulnerabilità e verificare quali protocolli e cifrari sono accettati.
  - ii. Con il comando “curl -k -I -H ‘Accept-Encoding:gzip, deflate’ https://example.com”, possiamo verificare se è abilitata la compressione HTTP.
  - iii. Con l’ applicazione java TestSSLServer possiamo controllare la vulnerabilità a BEAST o CRIME.