

# Web Security

LaSER

Università Statale degli Studi di Milano

16/04/2021

# Indice

1 Introduzione

2 HTTP

# Obiettivi e difficoltà

## Navigare sicuri all'interno del web

L'utente dovrebbe avere la possibilità di navigare senza incorrere in *pericoli*:

- sottrazione di informazioni sensibili, se l'utente non intende condividerle
- danni/modifiche ai propri file memorizzati sul computer o alla macchina stessa
- visita al sito  $X$  distinta dalla visita al sito  $Y$

## Applicazioni web sicure

Vogliamo avere a disposizione delle applicazioni, accessibili tramite web, che siano sicure tanto quanto le normali applicazioni stand-alone

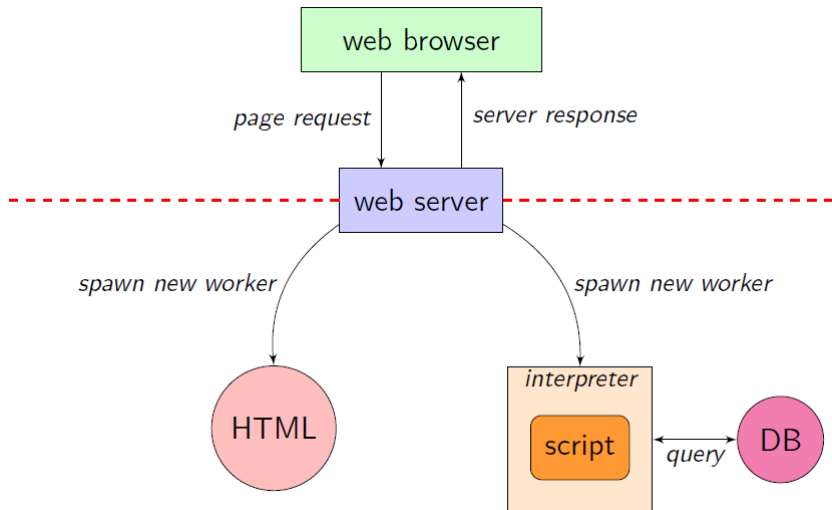
# Perché è così importante?

## Alcune statistiche

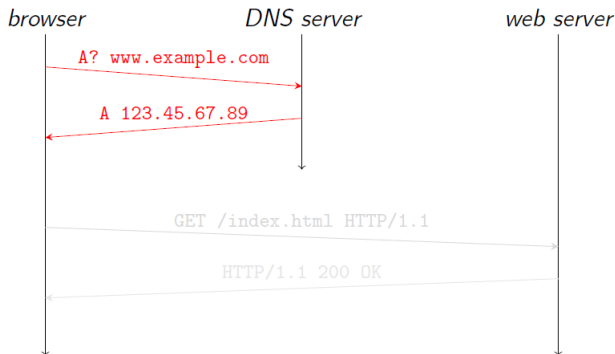
Fonte: <https://owasp.org/www-project-top-ten/>

- 1 Injection
- 2 Broken Authentication
- 3 Sensitive Data Exposure
- 4 XML External Entities (XXE)
- 5 Broken Access Control
- 6 Security Misconfiguration
- 7 Cross-Site Scripting (XSS)
- 8 Insecure Deserialization
- 9 Using Components with Known Vulnerabilities
- 10 Insufficient Logging & Monitoring

# Architettura infrastruttura web



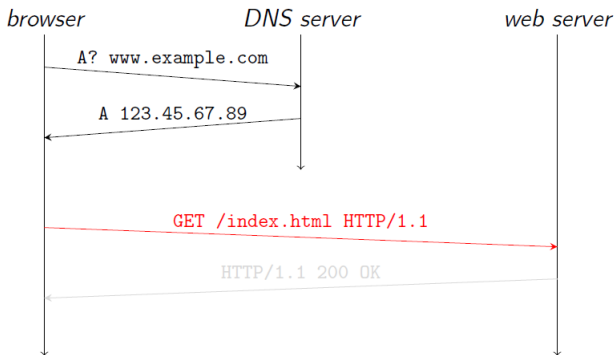
```
wget http://www.example.com/test.html
```



## Client ↔ Server DNS

- 1 il browser invia una *query* al DNS per ottenere l'indirizzo IP associato ad un particolare dominio

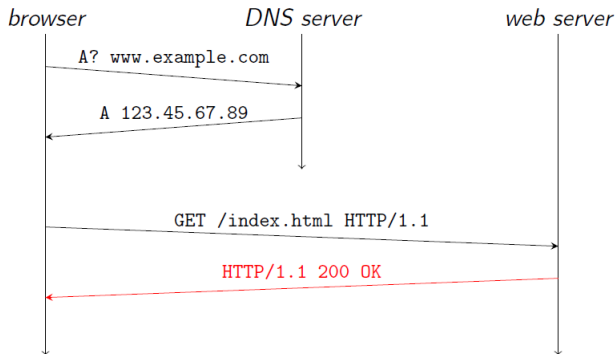
```
wget http://www.example.com/test.html
```



## Client → Server Web

- 2 il browser si connette al server web (su porta TCP 80 in assenza di altre indicazioni) ed invia la richiesta per ottenere la risorsa

```
wget http://www.example.com/test.html
```



Client ← Server Web

- 3 il server web processa la richiesta ricevuta e invia in output al client la risorsa (ad esempio una pagina HTML)



# Identificazione univoca delle risorse

## URL

Le risorse disponibili sulla rete (i documenti sui server) devono essere identificabili in modo univoco

I browser identificano le risorse tramite indirizzi detti *Uniform Resource Locator*

Un indirizzo URL ha una forma del tipo:

protocollo://server:porta/pathname

# Identificazione univoca delle risorse

## URL

Le risorse disponibili sulla rete (i documenti sui server) devono essere identificabili in modo univoco

I browser identificano le risorse tramite indirizzi detti *Uniform Resource Locator*

Un indirizzo URL ha una forma del tipo:

protocollo://server:porta/pathname

# Identificazione univoca delle risorse

## URL

Le risorse disponibili sulla rete (i documenti sui server) devono essere identificabili in modo univoco

I browser identificano le risorse tramite indirizzi detti *Uniform Resource Locator*

Un indirizzo URL ha una forma del tipo:

protocollo://server:porta/pathname

# URL

- **protocollo**: tipo di protocollo utilizzato
- **server**: indirizzo IP (numerico o simbolico) del server cui si vuole accedere
- **porta**: porta cui il protocollo fa riferimento
- **pathname**: percorso completo del file

Alcune informazioni sono superflue e possono essere omesse; il server e/o il client sceglieranno implicitamente dei valori di default, cioè predefiniti:

- protocollo → default HTTP
- porta → default 80, associata ad HTTP
- il nome del file → default `index.html`, `home.html` ...

# URL

- **protocollo**: tipo di protocollo utilizzato
- **server**: indirizzo IP (numerico o simbolico) del server cui si vuole accedere
- **porta**: porta cui il protocollo fa riferimento
- **pathname**: percorso completo del file

Alcune informazioni sono superflue e possono essere omesse; il server e/o il client sceglieranno implicitamente dei valori di default, cioè predefiniti:

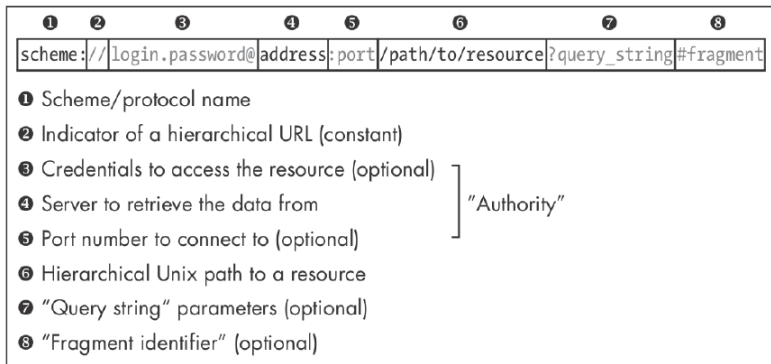
- protocollo → default HTTP
- porta → default 80, associata ad HTTP
- il nome del file → default `index.html`, `home.html` ...

# URL

I seguenti URL sono equivalenti:

- <http://security.di.unimi.it:80/index.html>
- <security.di.unimi.it:80/index.html>
- <http://security.di.unimi.it/index.html>
- <http://security.di.unimi.it:80>
- <http://security.di.unimi.it>
- <security.di.unimi.it>
- [159.149.145.240](http://159.149.145.240)

# Struttura di un URL



FONTE: "The Tangled Web", di Michael Zalewski, ED. No Starch Press, 2011.

# Componenti di un URL

## Scheme

- stringa *case-insensitive* che termina col carattere ':'
- protocolli comunemente utilizzati: http, https, ftp

## Indicator of a hierarchical URL

Per soddisfare i requisiti del RFC 1738, si richiede che ogni URL assoluto e gerarchico faccia precedere la stringa "//" all'*Authority*



# Componenti di un URL

## Credentials to access the resource

*Basic access authentication*: come le credenziali vengono inviate al server, dipende esclusivamente dal protocollo ed è opzionale

## Server address

Consentito che sia rappresentato tramite:

- un nome simbolico (*case-insensitive DNS name*)
- un indirizzo IPv4
- un indirizzo IPv6 racchiuso tra i caratteri '[' e '']

## Server port

Ogni protocollo, basato su TCP o UDP, sfrutta porte di 16 bit per distinguere messaggi destinati a diversi servizi in esecuzione sulla stessa macchina: ogni schema ha una propria porta di default

# Componenti di un URL

## Hierarchical file path

Utilizza la semantica UNIX per indicare una particolare risorsa del server

## Query string

```
name1=value1&name2=value2...
```

## Fragment ID

- pensato per fornire informazioni aggiuntive al client
- solitamente utilizzato per memorizzare una *anchor* nel documento HTML e agevolare la navigazione dell'utente
- può però venire impiegato per memorizzare particolari informazioni sullo stato

# URL encoding

## Caratteri riservati

Lista di caratteri indicati nel RFC 1630 come *non consentiti*:

: / ? # [ ] @ ! \$ & ' ( ) \* + , ; =

## Domande

- perché non sono consentiti?
- come fare se c'è necessità di utilizzarli?

# URL encoding

## Esempi di *percent encoding*

PROTOCOLLO: http

HOST: gyros

PORTA: 8080

RISORSA: /l@ser.html

IP: 192.168.1.8

- http:gyros/l@ser.html

## URL encoding

### Esempi di *percent encoding*

PROTOCOLLO: http

HOST: gyros

PORTA: 8080

RISORSA: /l@ser.html

IP: 192.168.1.8

- http:gyros/l@ser.html ✗
- http://gyros:8080/l%40ser.html

## URL encoding

### Esempi di *percent encoding*

PROTOCOLLO: http

HOST: gyros

PORTA: 8080

RISORSA: /l@ser.html

IP: 192.168.1.8

- http:gyros/l@ser.html ✗
- http://gyros:8080/l%40ser.html ✓
- http://%67%79%72%6F%73:8080/l%40ser.html

# URL encoding

## Esempi di *percent encoding*

PROTOCOLLO: http

HOST: gyros

PORTA: 8080

RISORSA: /l@ser.html

IP: 192.168.1.8

- http:gyros/l@ser.html ✗
- http://gyros:8080/l%40ser.html ✓
- http://%67%79%72%6F%73:8080/l%40ser.html ✓
- http://%25%36%37%79%72%6F%73:8080/l%40ser.html

# URL encoding

## Esempi di *percent encoding*

PROTOCOLLO: http

HOST: gyros

PORTA: 8080

RISORSA: /l@ser.html

IP: 192.168.1.8

- `http:gyros/l@ser.html` ✗
- `http://gyros:8080/l%40ser.html` ✓
- `http://%67%79%72%6F%73:8080/l%40ser.html` ✓
- `http://%25%36%37%79%72%6F%73:8080/l%40ser.html` ✗
- `http://gyros%3A%38%30%38%30/l%40ser.html`



# URL encoding

## Esempi di *percent encoding*

PROTOCOLLO: http

HOST: gyros

PORTA: 8080

RISORSA: /l@ser.html

IP: 192.168.1.8

- `http:gyros/l@ser.html` ✗
- `http://gyros:8080/l%40ser.html` ✓
- `http://%67%79%72%6F%73:8080/l%40ser.html` ✓
- `http://%25%36%37%79%72%6F%73:8080/l%40ser.html` ✗
- `http://gyros%3A%38%30%38%30/l%40ser.html` ✗
- `http://192.168.1.%38:8080/l%40ser.html`

# URL encoding

## Esempi di *percent encoding*

PROTOCOLLO: http

HOST: gyros

PORTA: 8080

RISORSA: /l@ser.html

IP: 192.168.1.8

- `http:gyros/l@ser.html` ✗
- `http://gyros:8080/l%40ser.html` ✓
- `http://%67%79%72%6F%73:8080/l%40ser.html` ✓
- `http://%25%36%37%79%72%6F%73:8080/l%40ser.html` ✗
- `http://gyros%3A%38%30%38%30/l%40ser.html` ✗
- `http://192.168.1.%38:8080/l%40ser.html` ✓

# Codifica *Punycode*

## Funzionamento

- i nomi di dominio sono sempre codificati in ASCII
- introduzione degli *International Domain Names* (IDN) e quindi della possibilità di registrare domini con caratteri Unicode
- soluzione → *Punycode*, definito nel RFC 3492
- consente di mappare univocamente una stringa Unicode nel suo equivalente ASCII
- compito del browser convertire da Unicode a Punycode

www.ręczniki.pl



www.xn--rczniki-98a.pl

## Punycode e *phishing* omografico

### Attenzione

L'attaccante sfrutta a proprio vantaggio caratteri simili ma diversi per attirare la vittima inconsapevole su un sito malevolo:

- `www.apple.com`
- `www.apple.com` → `www.xn--80ak6aa92e.com`

Seppure visivamente identiche, le due stringhe identificano due host differenti:

- il sito legittimo della nota azienda *Apple Inc.* usa il carattere ASCII 'a' (U+0061)
- il sito di phishing creato a scopi dimostrativi usa il carattere cirillico 'a' (U+0430)

---

<https://www.xudongz.com/blog/2017/idn-phishing/>

# HTTP

## HyperText Transfer Protocol

- originariamente pensato per il trasferimento di documenti in formato HTML, oggi è alla base del Web
- protocollo di livello applicazione, usato per trasferire dati tra *client* e *web server*
- *text-based* e *stateless*
- versioni: 1.0 (RFC 1945), 1.1 (RFC 7231), 2.0 (RFC 7540)
- incapsulato all'interno di connessioni TCP, di default su porta 80
- oggi utilizzato per trasportare anche altre informazioni

# HTTP Request

## Struttura

- *request line* (GET /index.html HTTP/1.1 ...)
- *header*: opzionali (User-Agent: Mozilla/5.0 (X11; U; Linux i686)) ma Host obbligatorio da HTTP 1.1
- linea vuota
- *payload* (opzionale)

## Note

- *request line* e *header* terminati da CRLF: *carriage return + line feed* (`\r\n`)
- linea vuota formata da CRLF
- implementazioni piuttosto flessibili: req. accettate anche con linee terminate dal solo LF

# HTTP Request

```
GET /index.html HTTP/1.1
Host: security.di.unimi.it
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64 ...) ...
Accept: text/html,application/xhtml+xml,application/xml; ...
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Referer: http://security.di.unimi.it/index.html
Upgrade-Insecure-Requests: 1
If-Modified-Since: Wed, 06 Mar 2019 13:59:31 GMT
If-None-Match: "16fe-5836d661fdc47-gzip"
Cache-Control: max-age=0
```

# HTTP Response

## Struttura

- *status line* (e.g., HTTP/1.1 200 OK)
- *header*: opzionali (e.g., Server: Apache/2.4.25 (Debian))
- linea vuota
- payload (opzionale)



# HTTP Response

```
HTTP/1.1 200 OK
Date: Mon, 09 Dec 2019 08:21:27 GMT
Server: Apache/2.4.25 (Debian)
Last-Modified: Wed, 06 Mar 2019 13:59:31 GMT
ETag: "16fe-5836d661fdc47-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Length: 5886
Connection: close
Content-Type: text/html
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
```

## Referer

Header utilizzato per indicare l'URL che ha generato la richiesta ad una specifica risorsa

### Problemi?

- rivelare informazioni riguardo alla navigazione dell'utente
- contenere informazioni sensibili nei *query parameters* dell'URL:  
`http://www.example.org/private_admin_section.php?action=logout&user_id=0` → `http://www.google.com`

### Situazioni in cui non è incluso tra gli header?

- inserimento manuale dell'URL nella barra o visita selezionando il link dai bookmark
- arrivo all'URL destinazione passando da uno *pseudo-url*
- URL di partenza all'interno di una sessione cifrata, ma quello destinazione non lo è
- tramite il blocco esplicito per mezzo di appositi *plugin* e *add-on*

# Metodi HTTP

## GET

- metodo per inviare dati usando il protocollo HTTP
- comunemente utilizzato per le interazioni client-server
- non è prevista la presenza di payload nella request
- secondo specifica, i dati sono preceduti dall'indirizzo della pagina richiesta e un punto interrogativo

## POST

- metodo per inviare dati usando il protocollo HTTP
- pensato per l'invio di informazioni tipicamente sotto forma di form HTML
- sempre presente l'header *Content-Length*
- secondo specifica, i dati sono inviati dopo che tutti gli header sono stati inviati dal client al server

# Metodi HTTP

## HEAD

- analogo a GET, ma il server restituisce in output solo gli header
- tipicamente utilizzato da *search engine bots* e altri tool automatici

## OPTIONS

- specifica quali metodi è possibile utilizzare per richiedere una particolare risorsa
- con \* al posto dell'URL, restituisce i metodi supportati dal server

# Metodi HTTP

## PUT + DELETE

- originariamente pensati per consentire invio e rimozione di file sul server
- nella pratica non vengono mai impiegati: sostituiti da POST/GET + scripts lato server

## TRACE

- sorta di ping
- restituisce utili informazioni sui proxy intermedi presenti tra client e server

## Tipi di HTTP response

RFC 2616 definisce  $\simeq 50$  differenti status code tra cui il server può scegliere per costruire la response da inviare al client: nella pratica meno di  $\frac{1}{3}$  di essi viene effettivamente utilizzato

### 200 - 299 → success

Questi codici indicano che la request è andata a buon fine:

- **200 OK:** risposta normale per GET/POST request che ha avuto esito positivo. Il contenuto del payload viene visualizzato all'utente
- **204 No Content:** richiesta andata a buon fine, ma non c'è nessun contenuto da visualizzare
- **206 Partial Content:** utilizzato in combinazione con le *range request*; i dati vengono ri-assemblati utilizzando le informazioni contenute nell'header `Content-Range`

## Tipi di HTTP response

### 300 - 399 → redirect

Questi codici indicano che non si è verificato un errore, ma il browser deve effettuare un'operazione specifica:

- **301 Moved Permanently, 302 Found, 303 See Other:** si richiede al browser di inoltrare la richiesta altrove, utilizzando le informazioni contenute nell'header `Location`
- **304 Not Modified:** la risorsa non è cambiata rispetto alla versione di cui il client è già in possesso (compare in presenza di header come `If-Modified-Since`)
- **307 Temporary Redirect:** analogo a 302, ma la richiesta non viene inviata utilizzando esclusivamente il metodo `GET`

## Tipi di HTTP response

### 400 - 499 → client-side error

Questi codici indicano errori legati al comportamento del client:

- **400 Bad Request:** il server non è in grado (o non intende) di processare quella specifica richiesta; ulteriori informazioni dettagliate vengono solitamente incluse all'interno del payload
- **401 Unauthorized:** richieste credenziali prima di poter accedere la risorsa
- **403 Forbidden:** la risorsa esiste ma non è possibile accedervi
- **404 Not Found:** l'URL richiesto non esiste

### 500 - 599 → server-side error

Questi codici indicano errori legati a problemi (principalmente di configurazione) lato server



## Passaggio di parametri - metodo GET

### caso 1: passaggio parametri tramite *form*

```
<form action="submit.php" method="get">  
  <input type="text" name="var1" />  
  <input type="hidden" name="var2" value="b" />  
  <input type="submit" value="invia" />  
</form>
```

### caso 2: parametri *embedded* nell'URL

```
<a href="submit.php?var1=a&var2=b">link</a>
```

### Richiesta HTTP corrispondente

```
GET /submit.php?var1=a&var2=b HTTP/1.1  
Host: www.example.com  
...
```

# Passaggio di parametri - metodo POST

## parametri POST

```
<form action="submit.php" method="post">  
  <input type="text" name="var1" />  
  <input type="text" name="var2" value="b" />  
  <input type="submit" value="invia" />  
</form>
```

```
POST /submit.php HTTP/1.1  
Host: localhost  
...  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 13  
  
var1=a&var2=b
```

## GET + POST

```
<form action="test.php?var3=c&var4=d"  
method="post">  
  <input type="text" name="var1" />  
  <input type="text" name="var2"  
    value="2 + 3" />  
  <input type="submit" value="invia" />  
</form>
```

```
POST /test.php?var3=c&var4=d HTTP/1.1  
Host: localhost  
...  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 13  
  
2+%2B+3
```

# Sessioni e Cookie

## Problema

- *stateless*: ogni richiesta è indipendente dalle precedenti
- le applicazioni web dinamiche richiedono concetto di *sessione*

## Cookie

- dati creati dal server e memorizzati sul client
- trasmessi tra client e server utilizzando header HTTP
- creazione cookie: `Set-Cookie: param=value<CRLF>`
- il client memorizza localmente l'info e nelle req. successive aggiunge: `Cookie: param=value<CRLF>`
- cookie standardizzati in RFC 2109 (*HTTP State Management Mechanism*)

# Sessioni e Cookie

## Sessione

Una sessione permette di gestire l'interazione tra client e server web (*stateful*)

## Componenti

- variabili di sessione
- identificativo di sessione

## Caratteristiche

- informazioni e stato devono essere memorizzati
- ogni richiesta HTTP deve contenere un ID di sessione
- le sessioni devono avere un timeout

# Sessioni e Cookie

## Sessione

- il concetto di *sessione* è implementato dall'applicazione web
- le informazioni di sessione devono passare tra client e server
- la trasmissione può avvenire tramite:

❶ header HTTP

```
GET /page.php HTTP/1.1  
Host: www.example.com  
...  
Cookie: sessionid=7456  
...
```

❷ URL

```
http://www.example.com/page.php?sessionid=7456
```

❸ payload HTTP

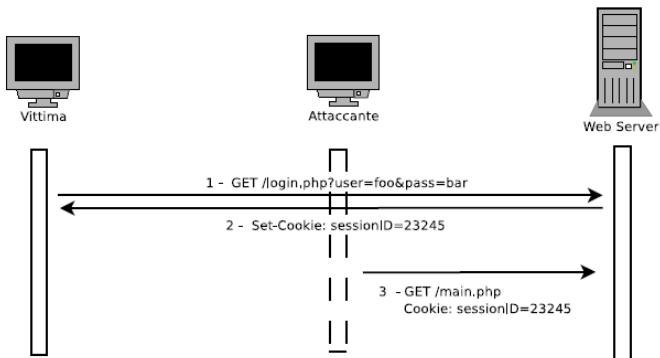
```
<INPUT TYPE="hidden" NAME="sessionid" VALUE="7456">
```

# Sessioni e Cookie

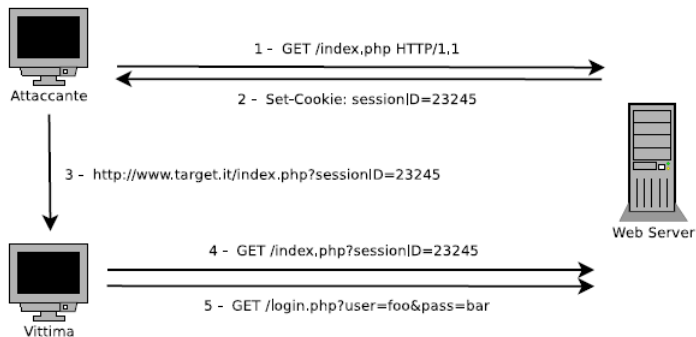
## Sessione

- è un elemento critico
- bypass del sistema di autenticazione
- deve essere valida per un periodo di tempo limitato
- attacchi:
  - intercettazione → SSL/TLS
  - predizione → *strong pseudonumber*
  - brute force → ID length
  - session fixation → controllo IP, Referer, rigenerazione ID, ...

# Session Hijacking



# Session Fixation





# Analisi di traffico HTTP

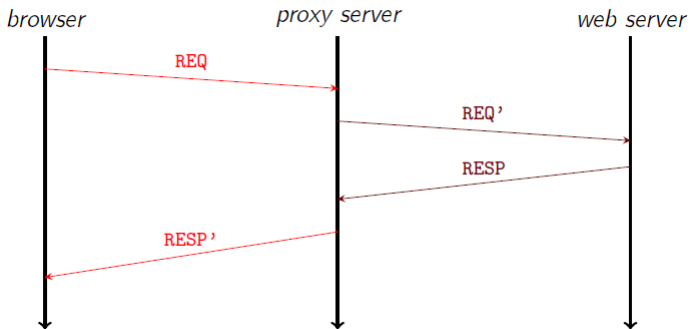
- payload HTTP incapsulato dentro il segmento TCP
- comunicazione *in chiaro*
- osservazione del traffico HTTP per analisi
- analisi tramite strumenti di *sniffing*: Wireshark, ...

# Proxy

## Intermediario

- componente che media la comunicazione tra due parti comunicanti
- separa la comunicazione tra due componenti, ponendosi in mezzo e disaccoppiandola, rendendola quindi indiretta
- agisce sia da client (rispetto al server originale) che da server (rispetto al client originale)

# Proxy



# Proxy

## HTTP header

HTTP\_FORWARDED  
HTTP\_VIA  
HTTP\_X\_FORWARDED\_FOR  
HTTP\_X\_FORWARDED\_HOST  
HTTP\_X\_FORWARDED\_PROTO

## Livelli di anonymizing

- *transparent proxy*: in HTTP\_X\_FORWARDED\_FOR rimane visibile l'indirizzo IP del client originale che fa partire la richiesta
- *anonymous proxy*: nasconde l'indirizzo del client originale
- *distorting proxy*: falsifica l'indirizzo del client originale
- *high anonymity proxy*: HTTP\_VIA vuoto

# Proxy

## HTTP proxy

- funziona come *man-in-the-middle* tra il browser e l'applicazione target
- modifica del traffico HTTP/HTTPS
- indipendenti dall'applicazione
- intercettando traffico HTTPS, il browser notifica l'errore di verifica del certificato SSL

# Proxy

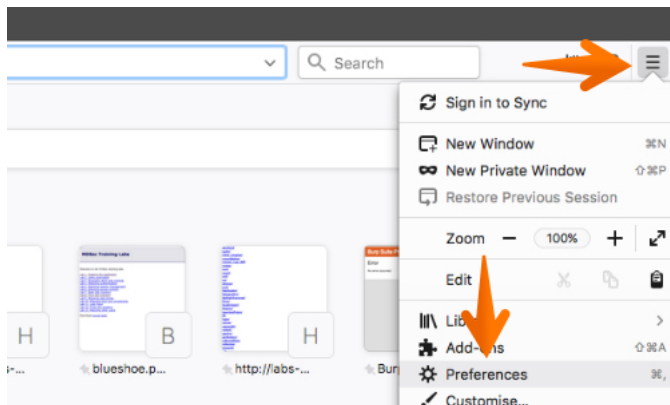
## HTTP proxy

- funziona come *man-in-the-middle* tra il browser e l'applicazione target
- modifica del traffico HTTP/HTTPS
- indipendenti dall'applicazione
- intercettando traffico HTTPS, il browser notifica l'errore di verifica del certificato SSL

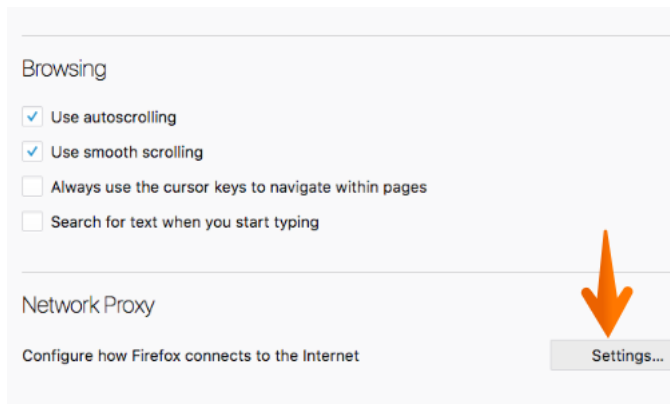
## Esempi di HTTP proxy

- Burp - <https://portswigger.net/burp>
- WebScarab - <https://www.owasp.org>

# BURP



# BURP





# BURP

Connection Settings

Configure Proxies to Access the Internet

☐ No proxy

☐ Auto-detect proxy settings for this network

☐ Use system proxy settings

☒ Manual proxy configuration

HTTP Proxy  Port

☒ Use this proxy server for all protocols

# BURP

No Proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24

Automatic proxy configuration URL

Reload


Do not prompt for authentication if password is saved

Proxy DNS when using SOCKS v5

Help

Cancel

OK



# Esercizi

## Natas

0x00

- connettersi a `http://natas0.natas.labs.overthewire.org`: la password per ogni livello successivo deve essere trovata
- username: `natas0`
- password: `natas0`

# Esercizi

## Natas

### 0x00

- connettersi a `http://natas0.natas.labs.overthewire.org`: la password per ogni livello successivo deve essere trovata
- username: `natas0`
- password: `natas0`

### Soluzione

- vedere il sorgente dello script: la password per il livello successivo è nei commenti
- `<!--The password for natas1 is  
gtVrDuiDfck831PqWsLEZy5gyDz1clto ->`

# Esercizi

## Natas

### 0x01

- connettersi a  
`http://natas1.natas.labs.overthewire.org`
- username: natas1
- password: gtVrDuiDfck831PqWsLEZy5gyDz1clto

# Esercizi

## Natas

### 0x01

- connettersi a  
`http://natas1.natas.labs.overthewire.org`
- username: natas1
- password: gtVrDuiDfck831PqWsLEZy5gyDz1clto

### Soluzione

- il sito non permette *right click* per vedere il file sorgente
- possibile controllare i sorgenti per mezzo dei *Web Developer Tools*
- `<!--The password for natas2 is  
ZluruAthQk7Q2MqmDeTiUij2ZvWy2mBi ->`

# Esercizi

## Natas

### 0x02

- connettersi a  
`http://natas2.natas.labs.overthewire.org`
- username: natas2
- password: ZluruAthQk7Q2MqmDeTiUij2ZvWy2mBi

# Esercizi

## Natas

### 0x02

- connettersi a  
`http://natas2.natas.labs.overthewire.org`
- username: natas2
- password: ZluruAthQk7Q2MqmDeTiUij2ZvWy2mBi

### Soluzione

- nel file sorgente dell'index c'è un tag `<img src="files/pixel.png" ...`
- `http://natas2.natas.labs.overthewire.org/files`
- `users.txt` (non compare tra i sorgenti perché non indicizzato da nessuno script)
- ...natas3:sJIJNW6ucpu6HPZ1ZAchaDtwd7oGrD14 ...



# Esercizi

## Natas

### 0x03

- connettersi a  
`http://natas3.natas.labs.overthewire.org`
- username: natas3
- password: sJIJNW6ucpu6HPZ1ZAchaDtwd7oGrD14

# Esercizi

## Natas

### 0x03

- connettersi a  
`http://natas3.natas.labs.overthewire.org`
- username: natas3
- password: sJIJNW6ucpu6HPZ1ZAchaDtwd7oGrD14

### Soluzione

- nell'index è presente il suggerimento: `<!-- No more information leaks!! Not even Google will find it this time... ->`
- `robots.txt`: `disallow /s3cr3t/`
- `/s3cr3t/users.txt`
- `natas4:Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ`

# Esercizi

## Natas

### 0x04

- connettersi a  
`http://natas4.natas.labs.overthewire.org`
- username: natas4
- password: Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ

# Esercizi

## Natas

### 0x04

- connettersi a  
`http://natas4.natas.labs.overthewire.org`
- username: natas4
- password: Z9tkRkWmpt9Qr7XrR5jWRkgOU901swEZ

### Soluzione

- nella home è presente l'avviso: Access disallowed. You are visiting from while authorized users should come only from  
`"http://natas5.natas.labs.overthewire.org/"`
- *Referer* di HTTP: cambiare valore in  
`http://natas5.natas.labs.overthewire.org/`