SICUREZZA
SAPIENZA – INFORMATICA TRIENNALE

ATTACCO XSS
CROSS-SITE SCRIPTING

Autori: Oleksandr Bazhura (2087354), Arvin Paolo Babasa Evangelista (2084249)

INTRODUZIONE

L'attacco **XSS (Cross-Site Scripting**) è una vulnerabilità di sicurezza molto comune nelle applicazioni web. Esso sfrutta il fatto che alcune pagine web accettano e mostrano input dell'utente senza una corretta "sanificazione" o "filtraggio". In pratica, se un campo di input, un parametro URL o un form non vengono adeguatamente controllati, un attaccante può inserire del codice HTML o JavaScript arbitrario che verrà eseguito dal browser di chi visita la pagina.

Vantaggi dell'attacco.

Lo scopo principale di un attacco XSS è **iniettare codice malevolo** all'interno di una pagina web. Se l'attaccante riesce a trovare una vulnerabilità per farlo, potrà:

1. Reindirizzare la vittima verso altri siti.
2. Manipolare contenuti visibili o i comportamenti della pagina.
3. Effettuare azioni in nome dell'utente, anche senza che se ne accorga.
4. Rubare cookie di sessione o altri dati sensibili agli altri utenti.
5. Dirottare la sessione dell'utente (per accedere senza credenziali).

Per la semplicità e per la chiarezza della relazione, successivamente verranno mostrate le implementazioni e le possibili conseguenze dei punti (1) e (2).

I TRE TIPI DI ATTACCO XSS

Seppur simili, è importante distinguere gli attacchi in base a come iniettano e attivano il loro codice.

1. Reflected XSS

L'attacco è **temporaneo**, quindi le sue modifiche **non verranno memorizzate** dal server. Si verifica quando il payload malevolo viene inviato come parte di una richiesta (URL, form, ecc.) e viene subito mostrato nella **risposta del server**.

2. Stored XSS

In questo caso, l'attacco ha come obiettivo quello di **salvare permanentemente** il codice malevolo sul server, che sia nei database o all'interno di un file. Risulta più pericoloso del Reflected XSS poiché una pagina compromessa **compilerà automaticamente il codice malevolo** appena visitata da un utente legittimo.

3. DOM-based XSS

Simile al Reflected XSS in quanto **temporaneo**, l'attacco DOM-based avviene completamente sul **lato client**, senza il coinvolgimento del server. Se il codice della pagina è vulnerabile all'input dell'attaccante, sarà il browser in locale a processarlo.

IMPLEMENTAZIONI PRATICHE

In questa sezione, attraverso esempi e scenari ipotetici, vedremo sia l'evoluzione che la motivazione dietro gli attacchi XSS, sia dal punto di vista dell'attaccante che del difensore.

Un esempio banale.

Si presenta una semplice pagina che saluta un utente con il nome passatogli dall'URL:

esempio_banale > saluto.php

```php
<!DOCTYPE html>
<html lang="it">
<head> ···
</head>
<body>
    <div class="container">
        <?php
        // Recupera 'nome' dall'URL
        $nome_utente = isset($_GET['nome']) ? $_GET['nome'] : 'sconosciuto';
        ?>
        <h1>Ciao <?php echo $nome_utente; ?>!</h1>
        <p>Ti volevo solo salutare 😄</p>
    </div>
</body>
</html>
```

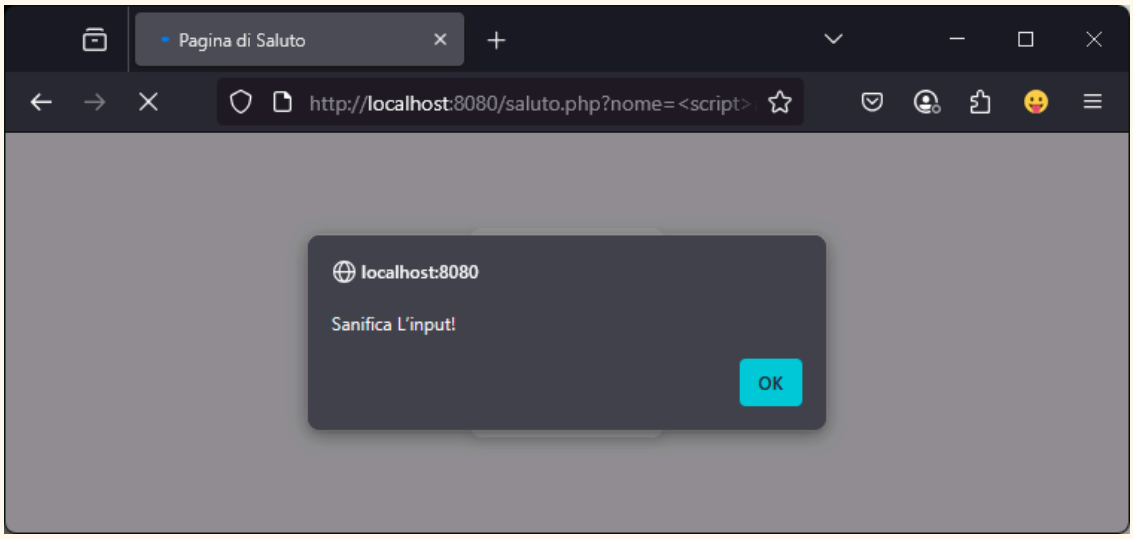
Questo è il risultato che si ottiene utilizzando il link [`http://localhost:8080/saluto.php?nome=Toni`]

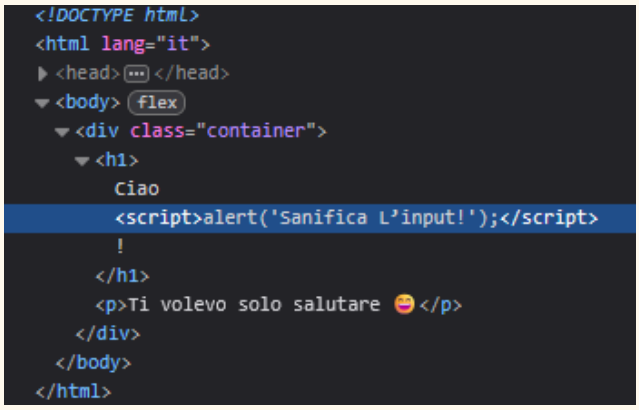
Il programmatore, concentrandosi più sulla funzionalità che sulla sicurezza (come spesso succede), ha correttamente scritto il codice per il compito assegnato. Il problema è che, non controllando la struttura dell'URL e lasciandoci tutta la libertà, **ha lasciato una vulnerabilità**, che sfrutteremo per mostrare un attacco **Reflected XSS**.

Inviando come link

`[http://localhost:8080/saluto.php?nome=<script>alert('Sanifica L'input!');</script>]`

Iniettiamo del codice esterno nella pagina, che verrà subito "riflesso" appena caricato.



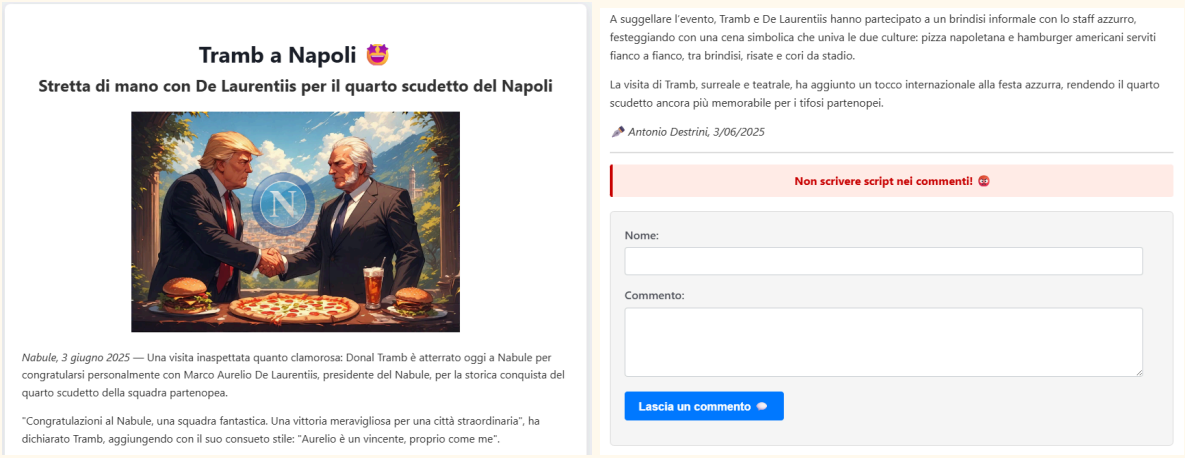


Analizzando la pagina tramite i tool del browser, vediamo che siamo riusciti a modificarla correttamente.

In questo caso, la soluzione è quella di sanitizzare l'input utilizzando la funzione `[htmlspecialchars()]` sulla variabile del nome utente, in modo da tradurre caratteri come "<" in **codifiche alternative** e renderizzarli come testo invece che interpretarli come codice.

Forum Culturale.

Mettiamo caso di avere una pagina rappresentante un articolo su una notizia sportiva, con la possibilità da parte degli utenti di commentare la loro opinione.



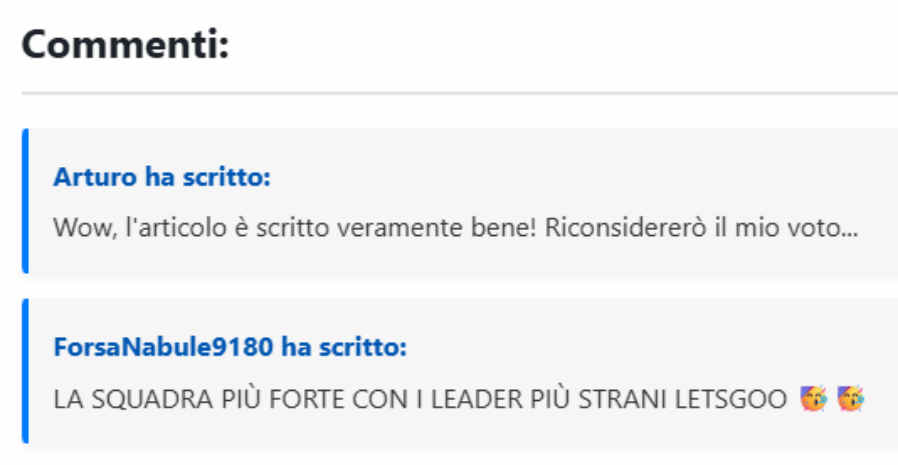
Il programmatore ha imparato la lezione precedente e ha messo la sanificazione sul form, purtroppo limitandosi solamente al campo del nome. Siccome i commenti **vengono salvati sul server** e il loro inserimento non è controllato, possiamo eseguire un attacco **Stored XSS**.

```php
/* serve per aggiungere comm in file */
function aggiungi_commento($nome,$messaggio){
    global $file_commenti;

    $nome_controllato=htmlspecialchars($nome,ENT_QUOTES,'UTF-8');

    $record="<div class='comment-item'><strong>".$nome_controllato."
    ha scritto:</strong><p>".$messaggio."</p></div>\n";

    if(file_put_contents($file_commenti, $record,FILE_APPEND|LOCK_EX)
    ===false) {
        // se si rompe qualcosa mostra errore
        error_log("non funge :(".$file_commenti);
    }
}
```

Le immagini qui sotto mostrano il meccanismo: nel file txt vengono salvati gli elementi HTML pre-impostati che successivamente verranno stampati all'interno della pagina web.

public > esempio_politico1 > comments.txt

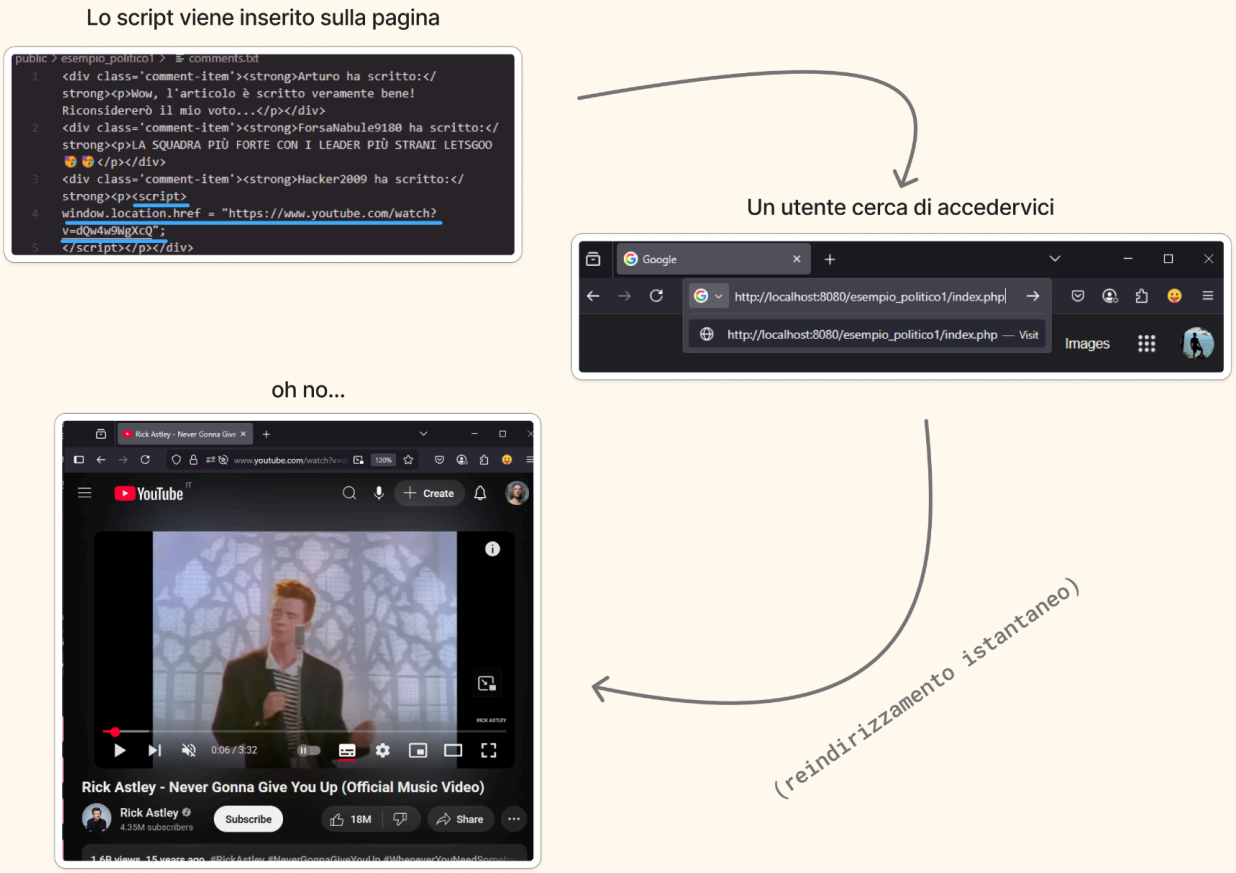
```
1  <div class='comment-item'><strong>Arturo ha scritto:</
   strong><p>Wow, l'articolo è scritto veramente bene!
   Riconsidererò il mio voto...</p></div>
2  <div class='comment-item'><strong>ForsaNabule9180 ha scritto:</
   strong><p>LA SQUADRA PIÙ FORTE CON I LEADER PIÙ STRANI LETSGOO
   🥳🥳</p></div>
```



Mettiamo caso la nostra intenzione sia di fare uno scherzo a chi visita l'articolo, indirizzando tutti gli utenti ad un sito esterno (1). Iniettiamo uno script nel file txt così verrà ricaricato ogni volta che un utente accede alla pagina.

Questa modifica risulta **molto pericolosa** poiché un attaccante malintenzionato potrebbe reindirizzare tutti gli utenti ad una **copia finta** del sito originale, gestita dall'attaccante stesso per ingannare gli utenti legittimi ad inserire informazioni sensibili, per esempio tramite la simulazione di un form di login.

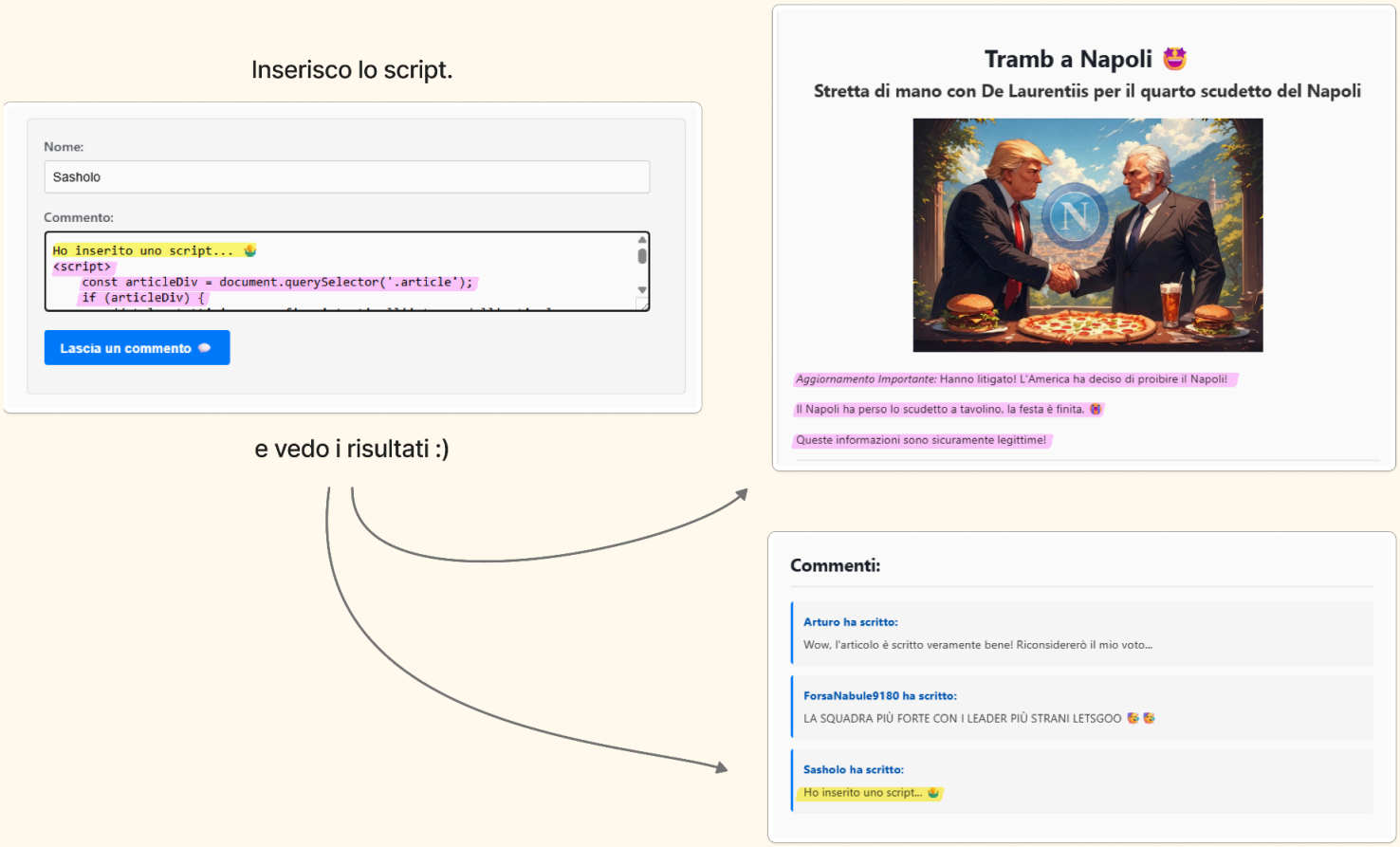
Inviando il commento succede questo:



Mettiamo caso le nostre intenzioni siano quelle di rovinare la reputazione delle persone citate nell'articolo, perciò vogliamo apportare delle modifiche al contenuto della pagina stessa (2) per influenzare le opinioni degli utenti.

JavaScript risulta molto flessibile in questi termini: anche se non possiamo modificare direttamente il file index.php su cui risiedono i contenuti, tramite un workaround possiamo cancellare dinamicamente gli elementi HTML e sostituirli usando uno script!

NB: tutto il codice e tutti gli script utilizzati sono disponibili nella cartella.



CONCLUSIONE

La varietà di attacchi possibili che si aprono con questa vulnerabilità spaventa alquanto, da un semplice Denial-of-Service si può passare al furto dei cookie di sessione (per accedere sul profilo di qualcuno senza usare le loro credenziali).

Come citato dal libro "*Computer Security: Principles and Practice*", è fondamentale avere procedure che possano evitare danni del genere, come l'utilizzo di espressioni regolari (regex) per la sanificazione oppure il fuzzing dell'input per trovare scenari di inserimento imprevisti.

Sebbene sia facile da capire che ci sia uno script all'interno di un link, anche un principiante riuscirebbe a nasconderlo tramite dei link shortener, embedding nel testo, oppure tramite siti proxy.

Siccome la prevenzione rimane sempre la strategia migliore nella sicurezza informatica, l'utilizzo di software ed estensioni browser opportune, come blocker di script e annunci, sono fortemente consigliati per maggiore protezione.

EXTRA (XSS Through HTTP Headers) (non implementato)

L’attacco XSS può essere fatto anche in altri modi, un esempio è la manipolazione degli header HTTP. Di solito viene realizzato tramite proxy, estensioni del browser o script JS dal lato client che inviano delle richieste modificate.

Esempio:
Supponiamo che un'applicazione logghi l'header **`User-Agent`** (che identifica il browser dell'utente) e poi visualizzare questi log in una pagina web senza sanificazione.

Un attaccante potrebbe impostare il suo **`User-Agent`** a:
➡️ `<script>alert('Sanifica l’input! :D');</script>`

Quando l'amministratore visualizza i log, lo script viene eseguito, le conseguenze di questa vulnerabilità sono già state discusse precedentemente.

Ricordiamo che usare **`htmlspecialchars()`** è sempre una buona pratica. ☺️