



ATTACCO

XSS

*RELAZIONE PROGETTO
INTERNET SECURITY*

*PENNISI ANDREA
Matricola: X81001014*

Sommario

ATTACCO XSS.....	1
RELAZIONE PROGETTO INTERNET SECURITY.....	1
PENNISI ANDREA.....	1
Matricola: X81001014	1
Introduzione.....	3
Pagine Web dinamiche	3
XSS (Cross-site scripting)	3
• Reflected.....	3
• Persistent.....	4
Esperimenti.....	4
Google XSS game	4
Con PHP	6
Not-persistent	6
Persistent	7
Con JavaScript	7
Con Angular	8
Strumenti utilizzati	8
Conclusioni	9
Riferimenti.....	10

Introduzione

Lo scopo di questo progetto è quello di attuare un attacco di tipo XSS, di seguito si illustrerà una breve descrizione del problema, quali sono gli exploit noti e in fine anche come è indicato agire per cercare di arginare il problema.

Pagine Web dinamiche

Una pagina web dinamica, diversamente dalle prime pagine web, è un tipo di implementazione che permette di creare, rimuovere o modificare contenuti e comportamenti della stessa in funzione a certi parametri, caratteristiche o input esterni ricevuti. Spesso i nuovi parametri arrivano direttamente dall'utente per cui diventa necessario il controllo, la validazione e talvolta lo scarto di alcuni input che potrebbero non essere coerenti con ciò che viene richiesto, o peggio potrebbero rappresentare un pericolo per il sito stesso e tutte le sue utenze.

Proprio questo tipo di pericoli prendono il nome di "Injection attack", di questi fa parte proprio la vulnerabilità di tipo "XSS".

XSS (Cross-site scripting)

Il cross-site scripting (noto come XSS) è una vulnerabilità di tipo informatica che caratterizza i siti web dinamici che non prestano particolare attenzione nel controllo degli input nei form.

La vulnerabilità permette ad un malintenzionato di "iniettare" codice all'interno appunto delle pagine web e tramite queste raccogliere, manipolare e reindirizzare dati più o meno sensibili, ma non solo, si potrebbe pure alterare il comportamento di un sito web e molto altro.

Esistono principalmente due tipi di vulnerabilità XSS:

- **Reflected:** sono sicuramente le più comuni. È possibile sfruttarle quando i dati forniti dall'utente (di solito tramite form HTML, ma anche su richieste di tipo GET tramite injection delle URL) sono usati immediatamente dallo script lato server per costruire le pagine risultanti senza controllare la correttezza della richiesta. Un attacco non persistente è tipicamente inviato via mail o da un sito web neutrale. L'esca è un URL dall'aspetto innocente, che punta a un sito attendibile ma che contiene un vettore XSS. Se il sito affidabile è

vulnerabile a quel vettore, il click sul link può causare l'esecuzione di script iniettato nel browser della vittima.

- **Persistent:** è una variante più devastante di cross-site scripting: si verifica quando i dati forniti dall'attaccante vengono salvati sul server, e quindi visualizzati in modo permanente sulle pagine normalmente fornite agli utenti durante la normale navigazione. Le vulnerabilità XSS di tipo persistente possono essere molto più significative rispetto alle altre perché lo script dannoso dell'attaccante è fornito automaticamente senza la necessità di indirizzare la vittima o attirarla nel sito di terze parti. In particolare, nel caso di siti di social-network, il codice potrebbe essere progettato per propagarsi autonomamente tra gli account, creando un tipo di worm lato client. Tra i principali dati rubati tramite questo tipo di XSS vi sono i cookie, che potrebbero essere poi utilizzati dall'attaccante per impersonificare la vittima agli occhi del sito colpito

Esperimenti

Gli attacchi di XSS possono essere molteplici, molto dipende dall'impostazione del sito che si vuole attaccare:

Google XSS game

1.



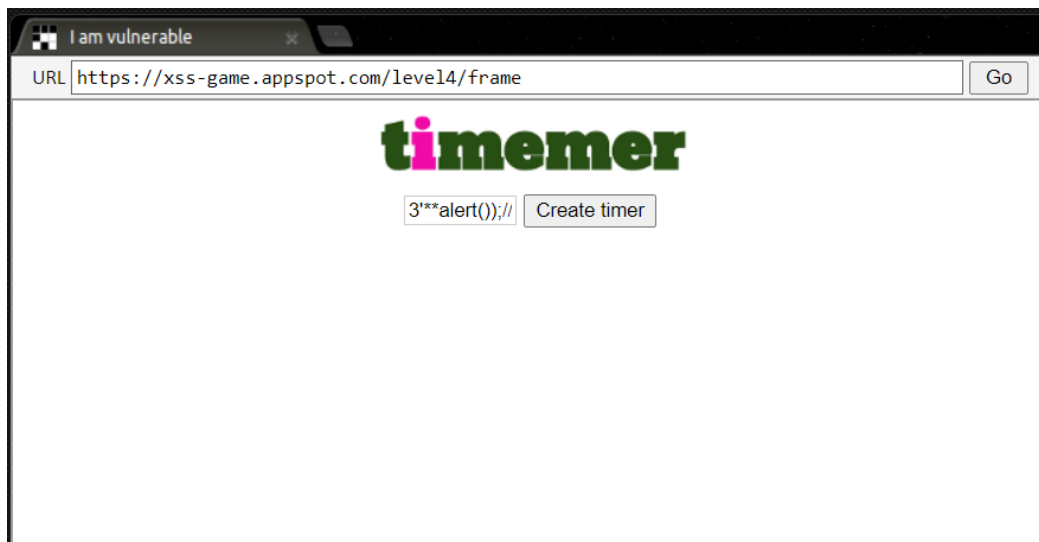
Nel primo caso la vulnerabilità è presente direttamente nel form di ricerca, poiché la query che noi andiamo a sottomettere nella textbox verrà mostrata successivamente per indicarci a cosa si riferiscano i risultati, questa operazione fa sì che l'esatto contenuto della textbox sia riportato in maniera esatta e senza nessun controllo nel contenuto HTML della pagina di risultato

permettendo quindi l'esecuzione dello script nella pagina seguente



Come si può notare alla fine della stringa “Sorry, no results were found for” avrebbe dovuto esserci la query di ricerca, che però non viene visualizzata in quando la pagina lo interpreta come uno script.

2.



In questo caso la vulnerabilità è intrinseca nel codice eseguito per far funzionare il timer: `` il valore inserito viene assegnato alla variabile “timer” che grazie all’escape “{{ }}" viene eseguita come uno script, e questo permette l'esecuzione di uno script inserito da input dopo l'esecuzione del timer stesso (in questo caso 3”).

3.



Qui si può notare come invece si sfrutti una cattiva gestione degli errori, in questa chat tutti gli elementi vengono inseriti nell'HTML della pagina stessa, ogni messaggio crea un container e successivamente viene inserito tutto al suo interno, gli script inseriti direttamente vengono rigettati da form e non permettono un attacco XSS come nel primo caso. Stavolta però basta inserire un'immagine inesistente che scatena un evento di errore e iniettare lo script come "gestione dell'errore" che però verrà reinterpretato, inserito nella pagina ed eseguito.

Con PHP

Not-persistent

Un'altra dimostrazione si può avere costruendo un semplice form PHP che invia una richiesta (per semplicità a se stesso che poi a sua volta gestisce la richiesta ricevuta, ma possiamo immaginarlo come una pagina che ne interroga una diversa e il risultato non cambia). Non avendo nessun controllo sugli input la pagina che verrà generata avrà tutto il contenuto degli input e sarà così strutturata:

PHP Form Validation Example

* required field

Name:







```
<body>
  <h2>PHP Form Validation Example</h2>
  <p>...</p>
  <form method="post" action="/index1.php">...</form> == $0
  <h2>Your Input:</h2>
  "Andrea "
  <script>alert('xss attack')</script>
  " Pennisi"
  <br>
  "andry.pennisi@gmail.com"
  <br>
  <br>
  <br>
</body>
```

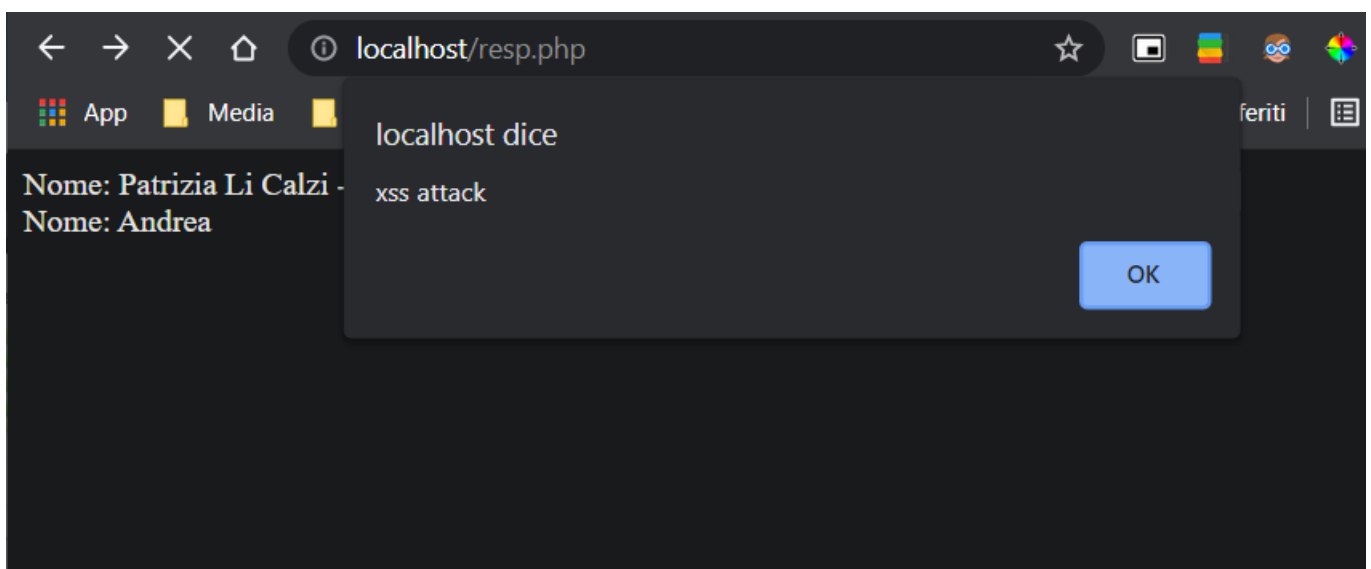
Come si può facilmente intuire il contenuto della input text viene stampato nella pagina HTML che interpreta il tag `<script>` come nativo e lo esegue come script javascript.

Persistent

In questo caso utilizzeremo sempre lo stesso form, ma gestito in modo differente:

- Alla compilazione del form e il relativo submit si verrà reindirizzati ad una pagina che memorizzerà i dati in un DataBase
- Successivamente visiteremo una pagina che prenderà i record della tabella e li stamperà nella pagina

<input type="checkbox"/>	 Modifica	 Copia	 Elimina	4	Andrea <code><script>alert()</script></code>	andry.pennisi@gmail.
<input type="checkbox"/>	 Modifica	 Copia	 Elimina	5	Andrea <code><script>alert('xss attack')</script></code> Pennis...	andry.pennisi@gmail.



Come possiamo osservare una volta inserito il codice malevolo ogni qual volta si carica la pagina, quindi ogni volta che viene interpretato l'HTML verrà eseguito lo script che è stato memorizzato nel DB. Questo appunto ha conseguenze molto peggiori rispetto al primo e chiunque visiti la pagina "resp.php" verrà colpito dallo script.

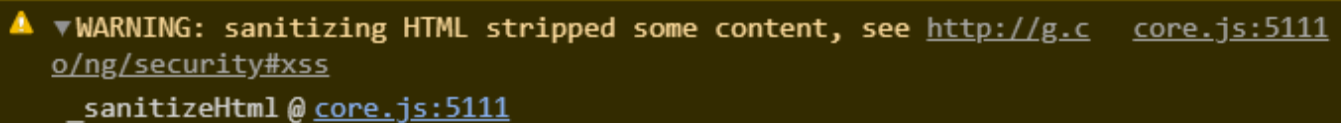
Con JavaScript

Con JavaScript puro e input banali un attacco del genere non può avvenire poiché si predispone di sanificazione automatica dell'input. Questo ovviamente non vale qual ora si riuscisse a capire quali escape sono necessari per evadere i controlli del sito attaccato.

Al giorno d'oggi le vulnerabilità di tipo XSS si sono molto ridotte visti i frequenti attacchi. Anche i browser si sono evoluti ed insieme ai framework permettono una gestione quanto meno basilare degli attacchi xss.

Con Angular

Di seguito un esempio di form creato in Angular 10 in cui al submit del form viene fatto un "innerHTML" senza alcun controllo:



```
▼ WARNING: sanitizing HTML stripped some content, see http://g.co/ng/security#xss
    _sanitizeHtml @ core.js:5111
```

Come si può notare è il framework stesso ad accorgersi di un possibile attacco XSS ed esegue lo stripping del contenuto, vanificando di fatto l'effetto dello script.

Link git a tutti i file: <https://github.com/apnnux/internetsecurity>

Per quanto riguarda attacchi di tipo persistenti oggi è sempre più difficile trovare vulnerabilità di questo tipo, sia per l'evoluzione delle infrastrutture quali appunto framework, sia perché la conoscenza di questo tipo di vulnerabilità fa sì che anche gli sviluppatori di web-app e website siano più attenti nella gestione delle stesse.

Strumenti utilizzati

I componenti utilizzati per le varie prove sono i seguenti:

- **Macchina principale (Windows):** Hosting dei vari servizi web di prova per l'attacco
- **Framework Angular:** utilizzato per la prova del form di registrazione
- **Typescript:** Linguaggio utilizzato su Angular per gestire gli script
- **Node.js:** moduli utilizzati per il framework
- **HTML, Javascript:** Utilizzato per la creazione di un sito-prova dove inserire un input e cercare di inserire lo script nel codice della pagina. *In particolare il

linguaggio javascript viene utilizzato pure per lanciare gli script all'interno dell'attacco.

- **Visual studio Code:** editor per la scrittura del codice
- **XSS Google Game**
- **UwAmp server:** Come hosting locale del DB e delle pagine PHP
- **SQL:** Per le query al DB
- **PHP:** Per i siti prova

Conclusioni

In conclusione possiamo affermare che un attacco di tipo XSS al giorno d'oggi, dove tutto viene svolto tramite web, dove si effettuano dalle operazioni più basilari come piccoli calcoli a pagamenti di qualunque importo, oltre alla gestione automatica delle nostre identità e tutti i nostri servizi, potrebbe risultare estremamente dannoso quantomai distruttivo anche, potenzialmente, a livello globale. Per questo motivo negli ultimi anni prevenire questo tipo di attacchi è diventata una delle priorità nello sviluppo web, una gestione accurata degli input esterni è fondamentale e in tal senso anche le moderne tecnologie di sviluppo ci vengono in contro fornendo strumenti per difendersi da questo tipo di attacchi.

Riferimenti

- <https://www.acunetix.com/blog/articles/persistent-xss/>
- <https://medium.com/iocscan/persistent-cross-site-scripting-p-xss-557c70377554>
- <https://xss-game.appspot.com/>
- https://it.myservername.com/cross-site-scripting-attack-tutorial-with-examples#2_Stored_XSS
- <https://thehackernews.com/2011/09/20-famous-websites-vulnerable-to-cross.html>
- <https://www.softwaretestinghelp.com/cross-site-scripting-xss-attack-test/>
- <https://www.cybersecurity360.it/nuove-minacce/cross-site-scripting-come-funziona-un-attacco-xss-e-come-proteggersi/>
- https://it.wikipedia.org/wiki/Cross-site_scripting
- <https://owasp.org/www-community/attacks/xss/>
- <https://angular.io/>
- <https://angular.io/guide/security#xss>
- https://www.w3schools.com/php/php_mysql_intro.asp
- https://www.w3schools.com/php/php_form_complete.asp