Sicurezza e siti web: code injection

Dopo aver parlato di **sicurezza su web** in generale, ed aver osservato l'enorme complicazione che sta dietro alla semplice navigazione di un sito web, ti sarai reso conto dell'**impossibilità di poter garantire la sicurezza totale**. Ma come realizzatori di siti web possiamo fare molto, partendo dal rendere il nostro HTML più robusto - senza scordarci di Javascript, PHP e tutti gli altri linguaggi che dobbiamo gestire in un sito. **Mettere in sicurezza** il proprio codice è il primo passo, e quello in cui noi, come realizzatori di siti web, abbiamo maggior controllo (e responsabilità). Come fare?

La prima cosa da evitare è pensare esclusivamente in termine di HTML. Un sito web è un'opera che integra numerosi linguaggi (uso una definizione un po' più ampia di quello che è in termini informatici un linguaggio, non si offendano i puristi):

- html/http
- javascript
- css
- php
- sql
- shell di comandi/navigazione file system
- espressioni regolari

Nella creazione di un sito web, io in realtà incrocio e collego tra di loro tutti questi linguaggi: creo HTML e CSS partendo da PHP, creo query in SQL ed espressioni regolari dinamiche magari tramite Javascript, fino ad eseguire comandi di sistema (sebbene non tutti se ne rendano conto!). La semplicità con cui lo facciamo spesso è tale che riusciamo persino a telefonare mentre arrangiamo e componiamo parti in sette linguaggi differenti...

In questo incrociarsi e sovrapporsi sta la potenza dei linguaggi del web, soprattutto perchè possiamo stabilire cosa e come incrociare basandoci sulle richieste degli utenti, che possono interagire tramite form, o url, o database. Io ricevo il mio modulo da una form in html, salvo i dati generando un comando di INSERT in linguaggio sql che poi uso per recuperarlo e visualizzarlo come html e javascript su un'altra pagina: il classico ciclo del blog, in cui senza rendercene conto abbiamo usato ed incrociato almeno tre o quattro linguaggi diversi, insieme all'input degli utenti da due sorgenti differenti!

Ma l'incrociarsi di linguaggi e di input utente nasconde l'insidia maggiore odierna dei siti web: **il code injection**.

Code Injection

Con questo termine vengono indicate le **aggiunte di codice imprevisto all'interno di un sito web**, per alterarne il funzionamento o l'aspetto (nell'uso comune del termine si sottintendono fini malevoli, intenzionali o meno). Il **code injection** nasce dal passaggio da un linguaggio ad un altro effettuato utilizzando dati utente (php a html, php a javascript, sql a html, etc). I dati utente, che possono essere tanto una url scritta sul client, quando qualcosa depositato in qualche modo (compilazione form, etc) su un database o nel file system e poi caricato successivamente, sono la causa che scatenano il code injection, quindi il punto da controllare.

È il nemico numero uno, la causa principale della stragrande maggioranza dei defacement, ed un rischio notevole considerando quanto spesso ed in quanti modi diversi puoi passare da un linguaggio all'altro e magari cambiando pure notazioni. Pensiamo solo a quante notazioni differenti puoi usare per il javascript:

codice in linea:

codice separato:

closure:

JSON:

Notazioni che possono essere dinamiche e create tramite PHP:

E magari generare HTML e Javascript, o Javascript che ... genera HTML:

In quest'ultimo banale caso, faccio notare che abbiamo appena incrociato tre linguaggi tra di loro! Avevi mai pensato in questi termini?

Un obiettivo facile

Perché il code injection è così popolare?

Primo, perché la causa che lo può scatenare (incroci di linguaggi + input) ha moltissimi punti di nascita in pressoché ogni pagina HTML. Questo l'abbiamo visto, e per averne conferma guardiamo un nostro sito dinamico con attenzione ...

Secondo, perché essendo spesso una problematica che richiede un'attenzione molto "informatica" a quanto si scrive, molto spesso il web designer medio è molto più artista che tecnico, ed è molto diffuso l'uso in cut & paste di parti di codice di cui spesso ignoriamo il perché funzionano - ma funzionano, e questo basta: il pirata fa tesoro di questo ragionamento.

Terzo, perché l'economia che ci viene (spesso giustamente) richiesta ci porta ad utilizzare sovente moduli pubblici come CMS, framework, librerie,a volte molto grosse e molto conosciute, quindi statisticamente prede più probabili - tanto probabili che **la maggior parte degli attacchi** sui moduli pubblici **avviene addirittura tramite script, senza intervento umano**. Ebbene sì, il tuo sito è stato conquistato da un programma, non da un brillante cracker!

I siti web attuali hanno raggiunto, a livello di codice, livelli di complessità a volte impressionanti, e la complessità è spesso difficile da gestire, specialmente (ma non solo...) per chi è alle prime armi. Così, capita che si creino inavvertitamente canali che portano direttamente dal browser (in mano al nemico!) al codice del nostro sito. Facciamo un terrificante esempio (NON USARE QUESTO CODICE!):

Chiamiamolo guestbook.php. Immaginiamo che faccia parte del sito http://www.vergognosi.IT.

Molto in voga negli anni 90, il guestbook, antesignano dei blog, permetteva di lasciare messaggi sui siti web - ed in moltissimi casi era il punto più attaccabile per impadronirsi del sito. Se pensate che un codice come sopra non sia mai stato usato - sbagliate, e di grosso.

I punti di debolezza qui sono due:

1 - il valore della textarea viene mostrato direttamente sul sito (passaggio da PHP ad HTML, input utente da file che prima scriviamo e poi leggiamo). Finché quello che contiene è normale testo, non ci sono problemi, ma se per sbaglio (o per fini malevoli) invece di un bel messaggio di saluti contiene una tag html, questa viene aggiunta senza batter ciglio in testa alla tua pagina -

alterandola più o meno radicalmente, immagina se la tag fosse qualcosa del tipo:

che avviasse uno script che mandasse in sostituzione l'intero modello DOM del sito con "altro" ... ti troveresti "altro" nel browser - e lo sarebbe sotto al bel nome del dominio del tuo cliente!

2 - il valore del campo radio viene usato come nome del file - direttamente. Qualcuno può pensare *"è un campo radio, quindi i valori li decido io"* .... ma il lato server non fa distinzioni tra radio, checkbox, textarea ... sono tutti testi! Tant'è che possiamo chiamare il nostro script con valori sulla url:

http://www.vergognosi.IT/guestbook.php?quanto=tanto&testo=ciao

e funzionerebbe perfettamente. Così come funzionerebbe:

http://www.vergognosi.IT/guestbook.php?quanto=index2.php&testo=**hacked%by%20me**

che mi creerebbe un nuovo bellissimo files php. Questo perchè mi *fido* del mio client javascript, e non controllo i valori passati! E cosa succederebbe se, invece di un nuovo file, con quel comando andassi a sovrascrivere pezzi a caso del mio sito perché magari il server web ha per errore permessi di scrittura dove non dovrebbe? Spesso, il code injection non è solo un semplice problema di HTML, ma una vera e propria **backdoor** che viene creata ed aperta sull'intero sito, se non sull'intero server, scatenando effetti domino su tutti gli altri anelli della catena.

La domanda più frequente

Quando spiego questi concetti, la domanda più spontanea che mi viene proposta è questa:

"ma solo io ho il codice del sito! Non possono capire cosa attaccare!"

Questo può non essere vero - è tranquillamente possibile che altri abbiano visto il tuo codice, magari perché il server ftp del rivenditore di spazio web è mal configurato e permette di vedere i files degli altri utenti inclusi i tuoi php, o perché è stato infiltrato ed è a tutti gli effetti in mano al nemico, che aspetta che tu lo sposti su un server web nuovo per poterti seguire ... Mai trarre conclusioni sulla sicurezza basandoci su presunte segretezze!

O può essere vero - ma nell'html sono sempre nascoste tracce come i nomi dei campi delle form, il codice javascript che genera eventuali chiamate ajax, tutte cose che possono essere utilizzate tramite forza bruta - semplicemente provo a passare valori "strani" ai campi, e vedo come si comporta il sito. Non è neppure necessario conoscere il codice del sito!

In conclusione

Chiudo l'articolo facendo notare che, ancora una volta, alla base di tutto c'è sempre un problema di fiducia non giustificata: mi fido del navigatore, mi fido del mio javascript, mi fido delle mie conoscenze specifiche di HTML e HTTP (i pulsanti radio), mi fido delle mie capacità di scrivere codice sicuro, mi fido della sicurezza dello spazio web.

Nel prossimo articolo presenterò alcune semplici tecniche di programmazione che, se usate opportunamente, ridurranno notevolmente le possibilità di essere vittime dei code injection - perlomeno nel codice scritto da noi.

Indice