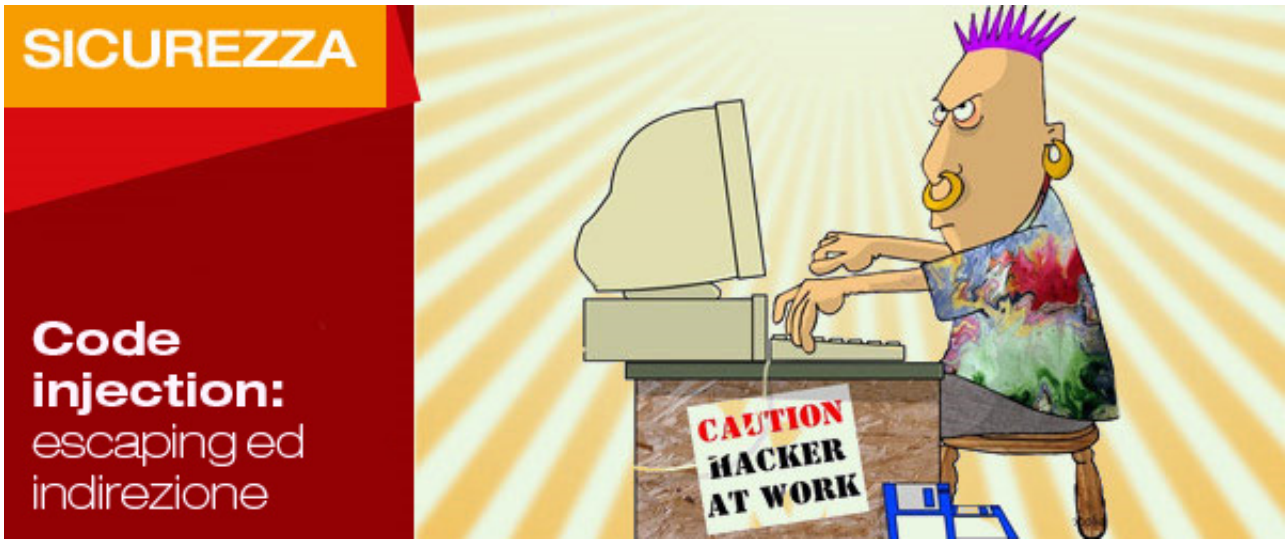
Sicurezza e siti web: code injection, escaping ed indirezione



Hai visto nell'articolo precedente che un'ottima fase di validazione, oltre che migliorare significativamente la qualità dei tuoi dati, può impedire il passaggio di valori indesiderati al tuo sito, specialmente quelli che possono causare problemi di sicurezza. Validare, però, è solo metà dell'opera: occorre anche essere sicuri che i dati 'validi' non causino problemi. Come?

Se ricordi (in questo articolo) i **code injection** si generano nel passaggio da un linguaggio ad un altro: e questo succede quando i dati che usiamo nel passaggio di linguaggio contengono caratteri che hanno un significato speciale, come ad esempio per l'HTML, o " e ' per Javascript. Ad esempio

Se "frutto" contiene "mela", tutto funziona. Se "frutto" contiene "", abbiamo un problema. Questo dovrebbe oramai essere molto chiaro, ma ... perché, in sostanza, è un problema? Perchè contiene "", che in HTML hanno un significato particolare: di apertura e chiusura di una tag.

In situazioni come queste, la domanda che devi porti è questa:

"la variabile in questione deve contenere testo normale oppure può contenere HTML?"

Nel secondo caso (HTML), dovrai **validare** la stringa, eliminando le tag non desiderate (ne parleremo più avanti in questo articolo).

Nel primo caso, invece, dovrai effettuare l'**escaping** dei caratteri con significato particolare: ovvero, sostituire questi caratteri con sequenze di caratteri normali, tradizionalmente chiamate **sequenze di escape**. Nel caso dell'HTML, ad esempio, si usano queste sequenze di escape:

- > diventa >
- & diventa &

Giusto come curiosità: quel & all'inizio delle sequenze è detto **carattere di escape**. L'ultima sequenza di escape converte ... il carattere di escape. Questa è una caratteristica di tutte le sequenze di escape: avere una sequenza di escape per il carattere di escape. Gli informatici, strano incrocio tra matematici, filosofi e psicologi, provano un gusto particolare per questo tipo di ginnastiche mentali, tanto rigorose quanto foriere di confusione per chi ha approcci meno logici e più creativi ... se a questo punto hai l'impressione del cane che si morde la coda, o del dilemma uovo/gallina, non farci caso: prendila come una curiosità e passa oltre ...

Escaping

I siti internet moderni sono tutto un passare, molto spesso assolutamente inconscio, da un linguaggio all'altro. Ad esempio, da php a html:

da php a sql:

da php a javascript:

Come hai visto, il passaggio di linguaggio diventa un punto di aggressione quando utilizza valori di provenienza esterna:

In ogni caso, è comunque buona norma effettuare l'escaping nei passaggi a prescindere dalla provenienza dei dati: ricordati sempre che **i siti sono soggetti a cambiare, i programmi ad essere riscritti parzialmente, le logiche adattate**: quello che oggi è un valore statico e fisso, domani potrebbe essere variabile, fornito dall'utente o letto da un database.

Le tecniche di escaping, nel dettaglio, variano a seconda dei linguaggi coinvolti: mentre i concetti di base sono certamente identici, è importante conoscere nel dettaglio cosa fare nelle varie situazioni. Come più volte indicato in questa guida, mi limito a considerare php, html, javascript, sql ed i comandi a sistema operativo, che sono di gran lunga i casi più frequenti nello sviluppo web odierno. Gli altri linguaggi hanno equivalenti e meccanismi simili, che vi invito ad approfondire nel caso dobbiate usarli.

Da PHP a HTML

Di gran lunga il passaggio più usato nello sviluppo dei siti internet, fortunatamente è anche il più

semplice da controllare. Come abbiamo visto sopra, occorre convertire i caratteri e &, che hanno significato particolare. Il php ci viene incontro con una funzione ad hoc: htmlspecialchars.

A volte, è necessario passare dell'html come parametro "valido" di una form. In molti forum o applicazioni particolari, è possibile inserire tag html semplici (es. strong, em>, a) nel testo, magari tramite Rich Text Editor come **tinyMCE**. In questo caso, occorre **controllare in php che le tag presenti nel valore letto siano quelle che ci aspettiamo**: può venirci incontro la funzione strip_tags , che ci aiuta ad eliminare tutte le tag al di fuori di quelle che permettiamo:

È buona norma anche validare:

Se le variabili contengono dati di cui conosciamo il tipo, **imponilo tramite conversione** prima di usarlo:

ATTENZIONE:

Non reinventare mai l'acqua calda! Per quanto possa essere semplice scriversi la propria funzione, **usiamo sempre ed in ogni posto possibile le funzioni di conversione fornite dal linguaggio**. Oltre a gestire tutti i possibili casi, anche quelli meno noti, sono molto spesso più efficenti e, se per qualche ragione cambiassero in un futuro i modi od i simboli da convertire, i nostri siti sarebbero tutti aggiornati automaticamente per gestire i nuovi casi.

Da Javascript a HTML

Si tratta di un passaggio molto in uso nei siti moderni: spesso, è l'anello finale di una catena che parte da una richiesta AJAX, che ritorna un valore da visualizzare, poi, a livello HTML. Sfortunatamente, **Javascript non contiene funzioni di escaping HTML**... dobbiamo pensarci noi:

ma la soluzione migliore in assoluto è sfruttare i framework. Ad esempio, con **jQuery** il tutto si riduce a:

Nel caso di jQuery, esistono due metodi: html che scriverà il testo con tag e tutto, e text che invece farà correttamente l'escaping della stringa. Usiamo sempre il secondo, a meno che non dobbiamo usare testo html - ed in quest'ultimo caso, validiamo per essere certi che siano le tag che ci interessano.

Esistono - e vanno usate - le funzioni di conversione a tipo. Ad esempio, **parseInt**:

Da PHP a Javascript

Con l'aumentare esponenziale dell'importanza che ha il Javascript nei nostri siti, fenomeno cui assistiamo in questi anni, questo è un passaggio che diventerà sempre più importante saper gestire correttamente. Il Javascript, nella notazione JSON, viene e verrà utilizzato molto spesso come "contenitore" per il passaggio di dati strutturati dai siti al lato server - questo oltre a tutti gli usi "normali" all'interno dei nostri siti, per la gestione delle interfacce utente.

Anche in questo caso, il php ci è d'aiuto. La funzione più importante è addcslashes, che ci permette di gestire l'escaping dei caratteri ',' e \, oltre che i caratteri di controllo:

Quanto detto per i valori di testo html ed i tipi, vale allo stesso modo anche qui:

Merita un discorso a parte la funzione (disponibile solo con php 5.2 e superiore) json_encode[link]. Sebbene il nome dia l'idea di qualcosa "da usare solo quando creiamo strutture json", in realtà la funzione è spesso usata al posto di addcslashes:

Il problema di questa funzione, è che **non abbiamo alcun controllo sul tipo di conversione effettuata**. Potrebbe essere una stringa - ed allora verrebbe convertita correttamente, inclusi i quote " attorno al valore. Potrebbe essere un numero, o essere NULL - in questo caso, ci troveremmo con valori numerici o null, o magari addirittura array ... Se preferiamo questo approccio, usiamo sempre un po' di validazione prima:

Nel caso usiamo json_encode per creare strutture complesse come array o oggetti, verifichiamo sempre che le strutture contengano valori corretti, validandole prima di convertirle - json_encode converte tutto a prescindere dal tipo, e possiamo trovarci in situazioni molto scomode, altrimenti.

Da PHP a SQL

Qui, caratteri speciali e funzioni da utilizzare variano a seconda del database utilizzato. Rimanendo sui due più in voga, per **mysql** si può usare mysql_real_escape_string, mentre per **postgres** si usa pg_escape_string:

Porta particolare attenzione, nel caso di salvataggi su database, all'uso di operatori o funzioni SQL

che accettano **wildcards o espressioni regolari**, come ad esempio l'operatore **ILIKE** di postgresql! In questo caso, l'escaping può essere davvero complesso, e va affrontato caso per caso (raramente sono disponibili funzioni di escaping così specializzate): sempre meglio riferirsi alla documentazione!

Anche in questo caso, è importante forzare i tipi delle variabili convertendoli, dove necessario. La tentazione è cadere in ragionamenti del tipo "tanto ci pensa il database!" e passare tutte stringhe. Non farlo mai!

I numeri diventino numeri, le date e le ore, siano forniti in un formato unico e magari dichiarato in anticipo. I campi vuoti diventino NULL dove sono effettivamente vuoti, o stringhe vuote o zeri dove invece rappresentano campi dichiarati ma vuoti ... in questo modo, il database ci darà una mano effettuando, anziché conversioni impreviste, controlli sui tipi salvati e letti.

Da PHP ad espressione regolare

Chiunque usi le espressioni regolari, sa con assoluta certezza quanto complicato sia scriverle, visto che praticamente ogni simbolo ha un significato particolare ... per aiutarci, ci viene incontro la funzione preg_quote:

In questo caso non solo contribuiamo a migliorare la sicurezza del codice, specialmente se la RE è usata per prendere decisioni sull'autorizzazione ad eseguire parti di codice, ma rendiamo la nostra vita molto, molto più semplice.

Da PHP a sistema operativo (shell)

Ultimo in ordine di utilizzo, ma in assoluto quello a rischio maggiore perché un errore nella gestione di questo passaggio può aprire le porte direttamente al file system ed al sistema operativo, **il passaggio a shell deve essere assolutamente limitato e controllato in modo paranoico**:

Cosa succederebbe se $nome contenesse, invece che un nome valido, qualcosa del tipo "/dev/null ; cat index.php" ? O magari "/dev/null ; rm -rf . &" ? Se a questo punto state guardando quella massa di caratteri senza capire, ve lo dico io: nel primo caso, vi prelevano index.php in formato sorgente, magari per cercare altri punti deboli per compromettere il sito. Nel secondo, vi cancellano intere porzioni di sito.

La chiave è quel ";" che nelle shell sta ad indicare, in modo non esatto ma per scopo di semplicità accontentiamoci, **la fine del comando e l'inizio di uno da eseguire subito dopo.** Quel "/dev/null serve per far eseguire il comando da saltare (grep) molto velocemente - hanno fretta. Quello che segue dopo il ";" è il comando che ti hanno appena iniettato - il tuo sito è compromesso, ai massimi

livelli: hai appena creato una porta di accesso completo a tutto.

Nei prossimi articoli amplierò questo argomento, per ora mi limito a segnalare le funzioni di escaping escapeshellarg e escapeshellcmd:

escapeshellarg converte un argomento in una stringa che non sia interpretabile dalla shell in altri modi, effettuando l'escaping dove necessario e quotandola tra '. escapeshellcmd fa la stessa cosa, ma per un intero comando: normalmente, è sempre più chiaro e pulito costruire il comando da passare alla shell pezzo per pezzo, come ho fatto sopra, ma possono esserci casi in cui può essere conveniente usare la conversione per intero... o almeno credo. Personalmente, credo di non averne mai sentito la necessità.

Quando NON usare l'escaping

Uno degli errori più frequenti è quello di considerare solo la prima fase della vita del valore, e dichiarare il problema chiuso una volta che, per esempio, il dato è salvato nel database (o in un file: il discorso è identico) dopo un regolare escaping. La tentazione, in questi casi, è di effettuare, oltre all'escaping necessario per il passaggio php-sql, anche quello per la successiva visualizzazione html, e sentirci al riparo da ogni problema.

ERRORE!

Occorre sempre considerare l'uso che faremo del dato per il resto della sua vita! Se, per esempio, i dati nel database verranno letti e visualizzati nell'HTML dopo, dovrà essere eseguito l'escaping dei valori letti: quindi, non ha senso farlo salvandoli!

Perchè questo? Per due ragioni.

Primo, **perchè il database è in mano al nemico** - che ha provveduto a salvarci dentro valori tali da causare problemi non durante la scrittura, ma durante la lettura! Quindi, se i dati vengono salvati da qualche parte, facciamoci la domanda: ho gestito l'escaping correttamente nei successivi caricamenti?

Secondo, perchè **l'uso dei dati può cambiare in futuro** - è inutile effettuare l'escaping, per esempio, di quello che scriviamo in un database ipotizzandone l'uso per l'HTML. Magari in un futuro prossimo verranno utilizzati, su nuove richieste del cliente, in un applicazione Flex, o tradotti in CSV,o usati per creare un PDF dinamico. Meglio sempre mantenerli il più possibile indipendenti dal formato d'uso!

In breve: **è assolutamente fondamentale non cercare di proteggere in anticipo gli usi futuri mentre salviamo i dati**, ma farlo mentre usiamo i dati. Pratichiamo l'escaping durante l'uso, non

durante il salvataggio se non sul solo linguaggio necessario a salvare i dati.

Indirezione

Una delle cose più complesse, in uno scenario di sicurezza, è passare come parametri ad uno script il nome di un file o di una tabella sql o un comando shell o sql. La ragione è ovvia: se posso modificare ogni parametro, posso anche imporre i miei comandi, e se ho in bella mostra il mio comando sql ... posso creare il mio ad hoc, magari per leggerti i record a sbafo, o peggio.

Ma ... posso farlo?

La risposta è sì, se proprio devi con mille precauzioni e con un milione di distinguo. Se si può evitare (e si può molto più spesso di quanto non si possa) meglio evitare. Ma se vogliamo cercarci grane ... validiamo, molto pesantemente:

- per nessuna ragione, ancor più di ogni altro caso, fidiamoci di quello che ci arriva come parametro;
  se si tratta di nomi di tabelle o comandi shell o sql, trattiamoli come etichette e validiamoli con uno switch o con un array;
- se si tratta di nomi di files, assicuratevi che siano nomi di files e non path! Se non possono essere validati come etichette, può aiutare l'utilizzo di funzioni come basename:

  e arricchite il codice che segue con controlli sull'esistenza del file (se dovete leggerlo), il fatto che sia scrivibile la directory dove dovete scrivere, e così via ... sempre avere piedi di piombo;
- sempre se lavorate con i files, non trascurate i file speciali . e .. ! Potreste avere sorprese poco piacevoli, quindi scartate sempre files di quel genere. Basta una semplicissima espressione regolare:

In ogni caso, visto che il browser è in mano al nemico, ricordati che dare troppe informazioni sulla struttura del lato server è da evitare: già questo è un ottimo motivo per non passare direttamente nomi di files o di tabelle, ma usare la tecnica chiamata comunemente **indirezione**, quando possibile.

Indirezione, in parole povere, significa sostituire ogni valore che l'utente ci passa senza usarlo direttamente. È anche una forma molto più elegante, chiara e pulita che passare nomi di tabelle o files direttamente, perchè ti perme di cambiare facilmente il nome delle tabelle stesse senza dover toccare il lato client. L'indirezione paga bene sugli interessi futuri, credimi.

Per esempio, invece che passare il comando SQL o il nome della tabella, puoi indicare

l'operazione da compiere:

L'indirezione si applica tanto nella fase di validazione quanto in quella di passaggio ad altro linguaggio. Lo strumento più usato è, appunto, il comando switch, ma in certi frangenti può anche essere comodo usare, ad esempio, un array:

Conclusione

Abbiamo visto ed approfondito cause e rimedi al code injection, soprattutto nei lati che riguardano lo sviluppo web. A questo punto, l'errore è considerare il capitolo come chiuso e completo: non farlo! Tutto ciò che riguarda la sicurezza è sempre e costantemente in evoluzione, nuove tecniche e nuove trappole sono all'ordine del giorno: tutto quanto ho raccontato, serve soprattutto a **capire qual è la logica che causa questi problemi, e quale tipo di soluzione può essere adottata per impedirli**. La caccia non ha mai termine: chiedi a Wiley il coyote, se non ci credi.

I concetti, più che gli esempi, sono la parte fondamentale da comprendere: da questi - e dagli esempi che ho riportato - puoi partire per rinforzare le tue applicazioni, imparando nel frattempo a migliorare le tecniche usate adattandole al tuo stile di programmazione, e crearne di tue.

Nei prossimi articoli tratteremo altri problemi legati alla sicurezza, questa volta concentrandoci sul lato server: quali sono i rischi che corriamo quando pubblichiamo un sito in php? Da cosa dobbiamo guardarci? Nel frattempo, lascio spazio alle riflessioni finali sull'argomento code injection con il finale più classico della letteratura didattica ...

"Domande?"

Indice

Come trovare le cause di code injection, tecniche di validazione.

Code injection: escaping ed indirezione