ISTITUTO NAZIONALE DI FISICA NUCLEARE

INFN-14-17/CCR
10th December 2014

ASPETTI DI SICUREZZA NELLA PROGRAMMAZIONE DI SITI WEB IN AMBIENTE ACCADEMICO

Michele Michelotto[1], Dario Vettore[2]

[1)]INFN-Sezione di Padova, Via F. Marzolo, 8, I-35131 Padova, Italy
[2)]Garr presso INFN Sezione di Padova Via F. Marzolo, 8, I-35131 Padova, Italy

Abstract

L'obiettivo di questa guida è fornire un supporto a chi deve gestire un server web in ambito accademico e deve sviluppare codice per il server web. Dopo aver installato un server web è importante anche mettere in sicurezza il codice per evitare attacchi da malintenzionati.

CCR-50/2014/P

Pubblicato da **SIS–Pubblicazioni**
Laboratori Nazionali di Frascati

1 INTRODUZIONE

1.1 A chi è rivolta questa guida

L'obiettivo di questa guida è fornire un supporto a chi installa e deve gestire un server web in ambito accademico e a chi deve sviluppare codice per i server web.

Dopo aver correttamente configurato un server web è importante anche saper scrivere codice il più sicuro possibile. Gli argomenti trattati in questa guida sono suddivisi in tre tematiche:

1. L'autenticazione ad un sito, lo studio dei messaggi di errore e la gestione delle sessioni.
2. La validazione dell'input ricevuto dall'utente, proteggersi dall'SQL Injection, accorgimenti sull'uso del linguaggio PHP e del database MySQL
3. Protezione da attacchi XSS

2 L'AUTENTICAZIONE

L'autenticazione è il processo attraverso il quale un individuo o una entità verificano la loro identità. Può avere diversi livelli di sicurezza, in base alla priorità della nostra applicazione: nella maggioranza dei casi viene tipicamente effettuata tramite uno **username** o **ID** con una **password** associata, in altri invece con delle chiavi o con dei certificati, per esempio X509 emessi da Terena per utenti GARR. Tutte informazioni che si ritengono in possesso di una sola ed unica persona (o in alcuni casi ristretta quando le credenziali sono condivise).

2.1 User ID e password

Lo user ID deve essere **case insensitive**, in alcuni casi viene utilizzato l'indirizzo email come identificativo, è consigliato di utilizzare sempre il case insesitive, sopratutto deve essere **unico**.

Come mai?

Potrebbe succedere che **Stefano** e **stefano** possano essere considerati due utenti diversi.

La password invece deve essere robusta, per evitare che venga scoperta (ad esempio con attacchi di forza bruta), alcuni criteri per giudicare se la password può essere considerata robusta oppure no sono:

Lunghezza: lunghezza superiore a dieci caratteri, altrimenti è considerata debole, l'applicazione stessa dovrebbe essere in grado di valutare la password che utilizziamo ed in caso negativo avvisare l'utente.

Complessità: le applicazioni che scriviamo devono invitare l'utente ad utilizzare una password complessa, scoraggiando l'utente dall' utilizzare normali parole comuni o quotidiane. **Devono** essere **case sensitive** per fare crescere la loro complessità. Ecco alcune regole per una buona password:

- un numero di caratteri compreso tra 10 e 128
- almeno un carattere maiuscolo (A-Z)
- almeno un carattere minuscolo (a-z)
- almeno un numero (0-9)
- almeno un carattere speciale (*,+,# ecc)
- non più di due caratteri uguali

2.2 Messaggio d'errore

La gestione dei messaggi di autenticazione merita una considerazione particolare, l'errore deve essere generico, non deve in nessun caso informare del perché non ci si è autenticati, in particolare:

Messaggi di errore non corretti

```
Login fallito User:stefano password non valida
Login fallito username non valido
Login fallito account disabilitato
```

Messaggi di errore corretto

```
Login fallito il nome utente o la password inseriti non sono validi.
```

2.2.1 Combattere gli attacchi di forza bruta

Inserire dei time out nella nostra applicazione possono scoraggiare dei possibili attacchi di forza bruta:

Alcuni tempi consigliati

- timeout di 5 secondi al primo login fallito
- timeout di 10 secondi al secondo login fallito
- timeout di 30 secondi dal terzo al decimo login fallito
- blocco totale di dieci minuti delle richieste di login per quel particolare userID dopo dieci login falliti

È possibile in alcuni casi bloccare le richieste in base anche alle molteplici richieste arrivate da uno specifico indirizzo IP, o da un range se persistono le richieste.

3 LE SESSIONI

Una sessione è una sequenza di richieste HTTP (e relative risposte) oppure di pagine web visitate associate allo stesso utente. Le applicazioni web richiedono la memorizzazione delle informazioni dell'utente durante la navigazione di un sito, evitando ad esempio così il continuo inserimento delle credenziali dell'utente. Le sessioni inoltre permettono a memorizzazioni di particolari variabili che contengono per esempio le informazioni dell'utente specifico, le quali verranno mantenute per tutta la durata della sessione.

3.1 Proprietà dell'ID di sessione

L'applicazione che sviluppiamo dovrebbe provvedere a fornire un ID di sessione assegnato nella fase di creazione condiviso con l'utente. La durata della validità dell'ID coincide con il tempo di attività dell'utente, l'ID di sessione deve rispettare alcune regole:

- deve essere lungo al massimo 128bits
- non deve essere prevedibile, es. potrebbe essere numero casuale generato in maniera automatica dall'applicazione
- non deve in nessun modo contenere informazioni che appartengono all'utente

3.2 Cliclo di vita delle sessioni

Il meccanismo più sicuro ed ovviamente utilizzato per la gestione delle sessioni è quello di crearne una **solo successivamente** ad una corretta autenticazione da parte dell'utente. Bisogna fare delle considerazioni su quando interrompere la sessione, questa infatti è sensibile ad un **hijack attack**, un attacco di questo tipo tende a sfruttare i dati di una normale sessione di lavoro per accedere alle risorse o alle informazioni di un host. Questo attacco avviene rubando le informazioni della sessione di un utente.

Come ci comportiamo con la chiusura della sessione?

- Timeout. Fortemente consigliato impostare un tempo durante il quale la sessione rimane attiva e un tempo dopo il quale la sessione scade. Il tempo di durata della sessione varia a seconda della criticità della nostra applicazione: 2-5 minuti se è ad alto rischio, 20-30 minuti per quelle a basso rischio.
- Inattività. In tutte le sessioni è necessario implementare un meccanismo di logout dopo un certo periodo di tempo di inattività, riconducibile ad una interruzione di richieste HTTP o ad una interruzione di caricamento di pagine web; necessario anche in questo caso è procedere alla distruzione della sessione. È utile perchè limita le possibilità di un attaccante di introdursi all'interno della nostra applicazione tramite la sessione di uno attualmente

autenticato, quindi potrebbe rimanere all'interno del sistema per il tempo necessario a compiere azioni pericolose.

- Timeout assoluto. Tutte le sessioni dovrebbero utilizzare dei time out di **tipo assoluto**, in maniera da definire il tempo limite entro il quale la sessione può rimanere attiva, oltre quel tempo la sessione deve essere distrutta.

- Rinnovo. Il rinnovo della sessione è un aspetto da non sottovalutare, può essere implementato nel caso in cui non ci siano particolari rischi di sicurezza, ma è un operazione complicata. Può essere fatto in particolari momenti della sessione e/o a prescindere dell'attività o meno dell'utente. Dopo un particolare lasso di tempo l'applicazione può rigenerare un nuovo ID di sessione. Per un periodo di tempo molto breve è necessario che siano attivi entrambi gli ID di sessione per dare tempo al client di venire a conoscenza del nuovo ID. Solo successivamente il nuovo ID verrà utilizzato dal client al posto di quello precedente. In questo modo si riesce a ridurre ulteriormente le possibilità di un 'hijack attack' anche quando l'utente è attualmente autenticato, l'attaccante infatti potrebbe aver recuperato il precedente ID. Questa tecnica potrebbe appesantire notevolmente la programmazione della nostra applicazione comporta inoltre una complicata gestione dei relativi timeout assoluti.

- Logout. Infine è importante gestire anche la naturale scadenza della sessione. La nostra applicazione deve garantire una naturale chiusura della connessione tramite un pulsante di **logout**, sempre presente in ogni pagina. Ha come compito quello di distruggere e cancellare tutti i dati della sessione corrente e reindirizzare l'utente vero la pagina di autenticazione. Sfortunatamente non tutte le applicazioni facilitano questa funzionalità. Ci vengono in soccorso dei plugin per firefox[1].

[1] https://addons.mozilla.org/it/firefox/addon/popup-logout/

4 INTRODUZIONE

L'SQL injection è una delle vulnerabilità che hanno un impatto enorme [2] sulle nostre infrastrutture. È una tipologia di vulnerabilità che si verifica quando un attaccante riesce oppure è in grado di sfruttare le vulnerabilità delle query SQL usate per interagire con un database di tipo MySQL sottostante. Si può essere in grado non solo riuscire a sottrarre dati dal database sottostante, ma a riuscire ad eseguire particolari comandi di sistema operativo. Non è una vulnerabilità solo a livello di applicazioni web, infatti qualsiasi tipo di codice che accetta input da una sorgente che non è sicura e riutilizzando come input per degli statement SQL può essere vulnerabile. È opportuno sottolineare che tutti i database sono soggetti a dei possibili attacchi di SQL Injection e che non ci sono linguaggi di programmazione web (PHP, ASP, Perl, ecc) che non soffrono di questi possibili attacchi.

4.1 SQL Injection

Sono sempre più diffuse anche in ambiente accademico le applicazioni web **Database-driven**, ovvero quelle applicazioni che si presentano all'utente con una interfaccia web scritta in PHP, ASP, ecc, al quale è agganciato un database di tipo SQL sul medesimo o su un altro server all'interno della stessa rete. Un esempio è illustrato in figura 1 a pagina 6:

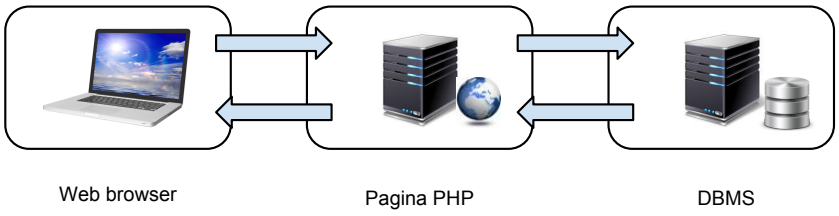

Figura 1: Una semplice architettura di tipo Web Database-driven

[2] https://www.owasp.org/index.php/Top_10_2013-Top_10

4.2 Come si procede

Un programmatore tipicamente inizia con la creazione della query **al volo** o a **runtime**, in questa maniera è possibile creare una query in maniera diversa sulla base dei parametri inseriti (per esempio nella compilazione di un form). Si può ottenere lo stesso risultato creando un query non in maniera dinamica, ma utilizzando delle query parametriche. Le query parametriche contengono al loro interno uno o più parametri, passati in alcuni casi a runtime (dopo il submit del form) e che contengono i dati di input. L'utilizzo dei parametri è più sicuro rispetto alla creazione dinamica delle query, ma un attaccante può comunque inserire delle ulteriori dichiarazioni SQL da eseguire, injection, che non dovrebbe in nessun modo essere possibile.

Differenza tra query dinamiche (1) e query parametriche (2):

```
(1) $cerca = "select * from utenti where id = $_GET["userid"]"
(2) $cerca = "select * from utenti where id = $userid;
```

4.3 Il carattere '

I database SQL interpretano tipicamente il carattere **'** come divisore tra i dati inseriti e la struttura della query. Un primo test verificare se siamo soggetti a possibili attacchi di injection è verificare come si comporta la nostra applicazione se inseriamo questo particolare carattere. Se viene inserito un singolo apice infatti dovremmo ottenere un errore simile a:

```
1) Warning: supplied argument is not a valid MySQL result resource
```

intrapreso come un errore generico, mentre un messaggio come questo:

```
2) You have an error in your sql syntax check the manual that
 corresponds to your mysql server version for the right
 syntax to use near ''VALUE''
```

è fortemente da evitare.

Motivo?

Il messaggio di errore **1** è un messaggio di **errore generico**, mentre il messaggio **2** ci avvisa

che il carattere ’ viene usato come divisore tra i dati inseriti nella query e la struttura della query stessa. Bisogna essere in grado di saper gestire il carattere di apice in maniera corretta perché un attaccante è in grado di aggiungere la sua query da iniettare. Altri caratteri da gestire con attenzione sono: | (pipe), virgola (,), punto (.), doppi apici () ed asterisco (*).

4.4 Il recupero dei parametri

Purtroppo filtrare gli apici e validare l’input non è sufficiente. Quando trattiamo numeri interi non è necessario utilizzare il carattere apice come delimitatore, tuttavia il numero intero viene trattato come una stringa. Anche in questo caso il programmatore potrebbe creare una query dinamica, nella maniera descritta sotto:

```
$cerca = "select * from utenti where id = $_GET["userid"]"
```

invitando l’attaccante ad iniettare del codice sql, per esempio utilizzando la funzione *LOAD_FILE* che legge un file e ritornare una stringa. Bisogna però sottolineare che l’utente per eseguire questa query deve avere i diritti necessari per poter leggere quel particolare file. Un esempio di attacco potrebbe essere:

```
select id from connessione  where id=1
  UNION ALL SELECT LOAD_FILE(’/etc/passwd’);
```

4.5 I messaggi di errore

Una gestione impropria della visualizzazione degli errori riscontrati può provocare seri rischi per la sicurezza. Si possono rivelare ad un attaccante delle informazioni preziose. I messaggi molto dettagliati posso usati per estrapolare informazioni del database, ad esempio se riprendiamo la query di prima, un attaccante la può modificare in questa maniera:

```
select id from connessione  wheRE id = ’ and 1 in  (SELECT @@version)
```

in questa maniera può ottenere molteplici informazioni riguardanti il database presente nel server.

5 MODI DI SQL INJECTION

Con dei piccoli semplici test è possibile mettere alla prova la nostra applicazione per scoprire se è vulnerabile all'injection.

5.1 Inferenza sui dati

È il modo più semplice. Si procede inviando dei dati **inaspettati**. Con questo sistema siamo già al corrente del fatto che la nostra applicazione ha saltato il passaggio della verifica dei dati inseriti e si prova ad eseguire la query nel server mysql per vedere come risponde. È necessario ricordare che queste applicazioni sono tipicamente di tipo client/server, un client genera una richiesta, il server la elabora e la rimanda al client. Noi come possiamo agire quindi? Per testare l'inferenza dobbiamo inviare al server una richiesta al quale non sa come rispondere. La prima operazione da compiere è quella di capire che tipo di dati il server accetta, ad esempio se aspetta dei dati tramite il metodo **GET** o il metodo **POST**. Qui sotto sono descritte le differenze tra i metodi:

GET: è un metodo di *HTTP* che se usato indica tutte le informazioni dentro lo stesso *URL*, ha un limite di 2028 caratteri di input, un esempio può essere:

```
GET /search.php?text=lcd%20monitors&cat=1&num=20 HTTP/1.1
```

in pratica la richiesta viene inviata al server seguendo lo standard:

```
?parameter1=value1&parameter2=value2&parameter3=value3...
```

in questo esempio quindi possiamo vedere che il client invia tre parametri *text*, *cat* e *num*, il server quindi riceve questi dati, li elabora e li inoltra al client. Manipolare una GET è molto facile, basta cambiare il testo direttamente sul link.

POST: è un altro metodo di *HTTP*, questo metodo viene tipicamente utilizzato quando si compila un form, il formato con cui si inviano i dati è sempre lo stesso, ma in questo caso non

appaiono sull'URL, non ha limiti sulla dimensione dell'input. Si suggerisce di utilizzare quest'ultimo metodo in quanto nasconde all'utente le informazioni che passiamo con il nostro form.

Sia i metodi POST e GET sono vulnerabili ad attacchi di tipo SQL Injection.

Altri esempi?

Ci sono altri esempi di modi in cui è possibile iniettare del software maligno: utilizzando i **cookies** che sono tipicamente utilizzati per l'autenticazione e vengono scambiati tra client e server.

Come possiamo quindi manipolare i parametri?

Un esempio molto semplice può essere:

```
http://www.test.it/showuser.php?gruppo=amministrazione
```

a questo punto possiamo modificare i dati da input per vedere come si comporta:

```
http://www.test.it/showuser.php?gruppo=attacco
```

se otteniamo una risposta come questa:

```
Warning: mysql_fetch_assoc(): supplied argument is not a valid MySQL
  result resource in /var/www/test.com/showuser.php on line 34
```

è facile intuire che non vi è una corretta validazione dell'input della nostra applicazione. A questo punto è possibile continuare con l'inferenza aggiungendo il carattere apice ' in questa maniera:

```
http://www.test.it/showuser.php?gruppo=attacco'
```

e la nostra applicazione reagisce così:

```
You have an error in your SQL syntax; check the manual that corresponds
 to your MySQL server version for the right syntax to use near
 ''attacco''' at line 10
```

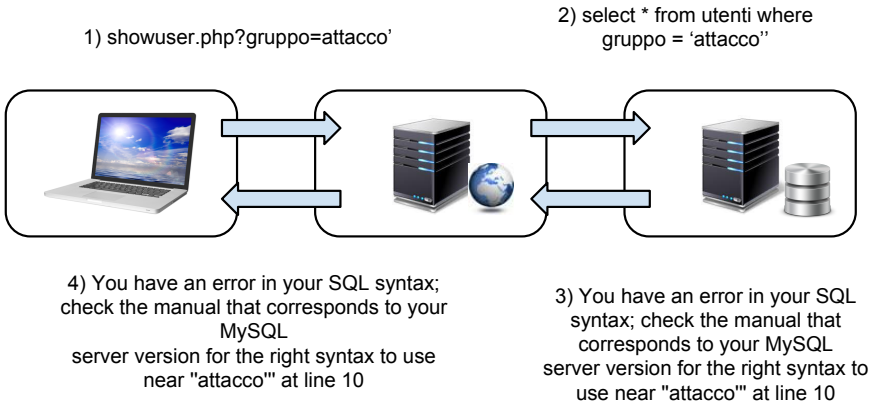

Figura 2: Elenco dei passi che avvengono durante un attacco di tipo SQL injection

Attenzione É opportuno fare una precisazione, la ricezioni di questi messaggi anomali non sono tutti riconducibili a delle vulnerabilità di SQL injection, ma abbiamo una buona probabilità che sia vulnerabile.

5.2 Gli errori dei database

Come ci si comporta tipicamente?

Osserviamo cosa succede tipicamente seguendo la figura 2 a pagina 11:

- 1 L'utente invia un richiesta per verificare se il server è vulnerabile ad un attacco di tipo SQL injection (molto semplice);
- 2 Il server web recupera i dati ed invia la query SQL al server mysql (può benissimo essere lo stesso server), crea la query e la passa al DB;
- 3 Il database esegue la query, sbagliata e produce un errore;
- 4 Il web server riceve l'errore e lo mostra a video.

OK, come andrebbe fatto invece?

L'errore ricevuto nel punto **4** dal web server dovrebbe essere gestito in maniera diversa dal server web, ad esempio:

1. Reindirizzando l'utente verso un altra pagina **preformattata** quando si ottiene un errore;

2. L'applicazione cattura l'errore in maniera corretta, non mostra nessun risultato e/o mostra un generico messaggio di errore.

Cosa succede? Evitando quindi di mostrare a video cosa succede in caso di questo errore restringiamo il campo dei possibili attacchi che possono essere compiuti verso di noi.

Come comportarci?
Nascondere a tutti tranne che all'amministratore di sistema i messaggi di errore di PHP e MySQL. Modificare o inserire una direttiva apache nel web server in un file .htacess o nel file httpd.con php_flag display_errors off, ed inserendo il carattere prima di una potenziale pericolosa istruzione PHP.

5.3 Blind Injection

Questo tipo di SQL injection effettua degli attacchi al DB ponendo delle domande che hanno come possibile risposta **true** o **false** sulla base della risposta interna dell'applicazione. Questo particolare tipo di attacco è spesso utilizzato quando la nostra applicazione è configurata per mostrare a video dei messaggi di errore generici, risulta essere più difficile poter sfruttare l'SQL injection, ma non impossibile.

6 CONTROLLO E VALIDAZIONE DEL CODICE

Spesso l'azione che tipicamente viene intrapresa per scovare se ci sono potenziali aree del nostro codice a rischio SQL Injection è analizzare il codice sorgente. Per questa tipologia di attività ci sono principalmente due metodologie: una procedura di **analisi statica** ed una di **analisi dinamica**. Il primo metodo non prevede esecuzione del codice, mentre il secondo invece lo prevede.

6.1 Codice pericoloso

Evitare query dinamiche: Sono da evitare le query contenenti stringhe generate dinamicamente (1) o con i parametri (2), come discusso nel paragrafo 4.2:

```
1) mysql_query("SELECT * FROM table WHERE field = '$_GET["input"]'");
```

```
2) $sql = "SELECT * FROM table WHERE field = '$_GET["input"]'";
   mysql_query($sql);
```

L'input non deve essere nemmeno passato direttamente alle procedure

```
CREATE PROCEDURE SP_ StoredProcedure (input varchar(400))
BEGIN
SET @param = input;
SET @sql = concat('SELECT field FROM table WHERE field=',@param);
PREPARE stmt FROM @sql;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
End
```

e quindi a sua volta utilizzarla **al volo**:

```
$result = mysql_query("select SP_StoredProcedure($_GET['input'])");
```

Il codice mostrato precedentemente mostra come l'input può essere passato a delle procedure dell'input senza essere correttamente filtrato e controllato. Questo meccanismo non diminuisce il fattore di rischio.

Attenzione ai metodi GET e POST È opportuno stare attenti a come si invia l'input stesso alla nostra applicazione, come descritto nel paragrafo 5.1 è possibile usare il metodo *get* e *post* di HTTP. Quando si usa PHP e uno di questi metodi è opportuno effettuare dei controlli su come sono inizializzate queste variabili:

- $_GET un array che contiene le variabili passate via GET
- $_POST un array che contiene le variabili passate via POST
- $_COOKIE un array che contiene tutti i valori passati con i COOKIE
- $_REQUEST un array che contiene le variabili passate via GET,POST e COOKIE
- $_SERVER informazioni sul server e l'ambiente di esecuzione

Variabili globali: **Consigliato disabilitarle, dove?** Nel file di configurazione di PHP (php.ini direttiva: *register_globals*). Se impostata ad **on** l'URL: `http://www.victim.com/process_input.php?foo=input` creerà una variabile globale *$_foo* senza l'utilizzo di nessun codice, questo può causare seri problemi di sicurezza.

Lunghezza dell'input ricevuto: controllo della lunghezza dell'input ricevuto dall'esterno

```
$id = $_GET["iduser"];
if (strlen($user) < $limit)
{
 //eccezione
error_handler("il parametro è troppo lungo per il campo
   richiesto: ERRORE")
}
else
$result = mysql_query("SELECT * FROM utenti WHERE id = '$id'");
```

in questo caso viene controllata la lunghezza dell'input, (**non il contenuto**) questa operazione può essere fatta quando abbiamo un'idea del range sulla lunghezza di caratteri che deve avere il nostro parametro di input. Nel caso in cui superi una certa lunghezza viene lanciata un'eccezione ed il programma termina la sua esecuzione. È possibile così anche tracciare il nostro input e le possibili esecuzioni del codice, identificando gli input infetti. Successivamente è possibile riunire tutte le parti del codice che utilizzano un input

proveniente dall'esterno e applicare delle funzioni che controllano in maniera standard l'input.

Utilizzo di espressioni regolari: l'utilizzo di espressioni regolari serve ed aiuta molto per filtrare l'input, alcuni esempi:

- data in formato gg/mm/aaa:

  ```
  /^[0-9]{2}\/[0-9]{2}\/[0-9]{4}$/
  ```

- email: permette email tipo jirwk@hep.s.asdfa-u.ac.jp

  ```
  /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+
    ([a-zA-Z0-9]{2,})+$/
  ```

- nome: da 1 a 30 caratteri di nome

  ```
  [zA-Z]{1,30}
  ```

- telefono: da 2 a 8 cifre

  ```
  [0-9]{2,8}
  ```

un esempio di uso delle espressione regolari:

```
$username = $_POST['username'];
if (!preg_match("/^[a-zA-Z]{8,12}$/D", $username) {
// handle failed validation
}
```

solo caratteri alfabetici sono ammessi per questo username in un range compreso tra 8 e 12.

Controllo dei tipi e cast: il controllo dei tipi (dove un tipo è un **nome** dato ad un insieme di valori che può assumere una variabile) è molto utile, ad esempio per forzare anche tramite un cast (ovvero una conversione di una variabile da un tipo di dato ad un altro) un certo dato al suo tipo corrispondente, in PHP:

- **is_numeric(input)** o **is_string(input)**
- **$n = (int)$numero;**

per esempio in PHP:

```
if(is_numeric($eta)) //controllo del valore
$eta = (int)$_POST['eta']; //cast ad intero

if (!preg_match("/^[0-9]{1-3}$/D", $eta) {
// handle failed validation
}
```

Codifica dell'output: Oltre ad effettuare un controllo dell'input, è necessario anche controllare i valori che sono scambiati tra le varie parti della nostra applicazione, molto spesso se alcune parti possono essere state realizzate da un altra persona la quale potrebbe non aver effettuato tutti i controlli necessari.

Perchè? Perché potrebbero non essere state validate precedentemente in maniera corretta.

Blacklist: definire una blacklist con i caratteri che non vogliamo siano inseriti all'interno del form. Se vogliamo escludere dall'input caratteri come l'apice, il simbolo percentuale, i due trattini, parentesi quadra aperta, ecc la regex che dobbiamo utilizzare è la seguente:

- `'|%|--|;|[|/\*|\\\*|_|\[|@|xp_`

Memorizzazione dei dati sensibili: È opportuno stare attenti anche a come ci comportiamo quando memorizziamo informazioni sensibili nel nostro db, uno degli obbiettivi degli attaccanti è proprio quello di rubare dati sensibili come ad esempio username, password ecc.

- **Password**: non bisogna **mai** salvare in chiaro la password nel db, si devono preferire alternative come le funzioni one-way (Es. SHA56).

Nomi di tabelle e campi: per procurare ad un attaccante un maggior lavoro è utile dare a tabelle e campi sensibili un nome non semplice o non banale.

Perchè? Per rendere un attacco più difficile è buona idea dei nomi delle tabelle dove si salvano utenti e password non ovvi, questo non fermerà l'attaccante, ma sicuramente per lui sarà più complicato recuperare le informazioni.

Honeypot table: per essere avvisati se qualcuno tenta di leggere le informazioni dal nostro db, può essere utile installare un honeypot table con un campo **password** il quale contiene al suo interno delle informazioni fasulle, facendo il modo che l'amministratore si sistema riceve un avviso via email nel caso in cui succeda un tentativo di accesso (Database Oracle), in pratica un Virtual Private DB. In MySQL non sono presenti queste funzionalità, in alternativa è possibile abilitare i log delle query effettuate sul db, per poi realizzare un parser che legge il log e ci avvisa se qualcuno sta accedendo alla nostra Honeypot table.

7 INTRODUZIONE ALL'XSS

I cross-site scripting sono degli attacchi che consento di iniettare del codice maligno all'interno di codice benigno in siti web che l'utente considera **fidati**. Attacchi XSS come SQL Injection hanno un impatto enorme [3]. Un attaccante inietta codice maligno tipicamente all'interno di un form, per attaccare l'utente che si collega o compila quel particolare form, lo sfortunato (e ingenuo) rimane ignaro all'attacco per il fatto che si fida del form oppure del sito che sta visualizzando. Il codice maligno così può avere accesso a cookie, id della sessione ed altre informazioni dell'utente corrente recuperate direttamente dal browser dell'utente. Cerchiamo in questa sezione di descrivere alcune tecniche per prevenire questo tipo di attacchi in particolare utilizzando delle tecniche di escape e encoding ed eventualmente modifiche sulla configurazione del server web.

[3] https://www.owasp.org/index.php/Top_10_2013-Top_10

7.1 Come viene impostato un attacco XSS

Questa tipologia di attacco può avvenire in due modi sulla base di come vengono ricevuti i dati:

1. I dati arrivano ad una applicazione web da una sorgente non attendibile (richiesta web forum);

2. I dati sono inclusi dinamicamente nel contenuto web che viene inviato all'utente, senza essere correttamente validati.

Tipicamente il codice malizioso è codice javascript, ma può talvolta includere codice HTML, Flash o qualsiasi altro tipo di codice che viene inviato ed eseguito dal browser. Sono pochi i possibili attacchi XSS, ma possono comportare il recupero di informazioni private e sensibili dell'utente, reindirizzare l'utente verso un sito web malevolo o in certi casi riuscire a compiere operazioni pericolose sulla macchina stessa dell'utente.

7.2 Stored & reflected

Gli **stored** attacks sono quei particolari tipi di attacco dove il codice viene iniettato in maniera definitiva nel bersaglio (per esempio in un database), la vittima quindi recupera il codice maligno ogni volta che si collega a quel determinato server. Nei **reflected** attacks invece il codice maligno è iniettato direttamente durante l'utilizzo delle richieste (esempio GET e POST) oppure in messaggi di errore, risultati di ricerca, effettuati dallo stesso client che subisce l'attacco. Si procede tipicamente con l'invio dei una email che contiene link malevoli oppure tramite la visualizzazione di un pop-up, se l'utente sbadatamente clicca sul link o sul pop-up il codice verrà iniettato nella pagina che l'utente sta visualizzando ed eseguito lato client.

7.3 Conseguenze

Come già descritto le conseguenze sono molto serie per l'utente. Un attaccante è in grado infatti di recuperare informazioni dalle sessioni dell'utente, dai cookies, installazione di virus o trojan, reindirizzamento dell'utente verso un altro sito web o una interfaccia simile al solo scopo di sottrarre informazioni riservate.

7.4 Siamo vulnerabili?

Gli attacchi XSS sono molto difficili da identificare e da rimuovere, la soluzione migliore sarebbe effettuare un'analisi di controllo del codice sorgente delle pagine web in cui viene richiesta la compilazione di un form, purtroppo in quasi tutti i tag HTML può essere iniettato codice maligno, l'operazione quindi è molto difficile. Programmi come Nessus (OpenVas come alternativa open source) e nikto possono venirci in aiuto per quest'attività.

8 COME PROTEGGERSI?

8.1 Disabilitazione TRACE HTTP

Disabilitazione lato server del supporto **HTTP TRACE** dove sono memorizzati informazioni dell'utente all'interno del web server.

Come capire se è attivo?

Un primo modo semplice e veloce per capire se questo servizio è attivo sta nell'utilizzare il comando **telnet** sulla porta 80.

```
telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
TRACE / HTTP/1.0
Host: foo
Any text entered here will be echoed back in the response
    <- ENTER twice to finish

HTTP/1.1 200 OK
Date: Sat, 20 Oct 2007 20:39:36 GMT
Server: Apache/2.2.6 (Debian) PHP/4.4.4-9
```

```
    mod_ruby/1.2.6 Ruby/1.8.6(2007-06-07)
Connection: close
Content-Type: message/http

TRACE / HTTP/1.0
Host: foo
Any text entered here will be echoed back in the response

Connection closed by foreign host.
```

Un secondo modo è utilizzando i tool Nessus/OpenVas

Soluzione?

Aggiungere queste righe nel file di configurazione di Apache e/o per ogni virtual host.

```
RewriteEngine on
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)
RewriteRule .* - [F]
```

8.2 Filtraggio e validazione dei dati ricevuti

8.2.1 Dati non sicuri

Vengono considerati **dati non sicuro** i dati che provengono da una richiesta HTTP, tramite i parametri inseriti nell'URL, campi di un form, headers e cookies. Questi possono contenere al loro interno input modificati e che possono contenere minacce di attacco.

8.2.2 Alcune semplici regole

MOLTO IMPORTANTE!

Tutti i controlli descritti devono essere fatti lato server ad esempio con il linguaggio PHP, ASP o Java. Perchè? semplicemente perchè è facile evitare i controlli effettuati lato client (es. l'utente ha JavaScript disabilitato.)

1) Non inserire dati non verificati: la prima regola è quella di vietare tutto, non inserire dei dati non sicuri/non verificati all'interno della tua pagina HTML:

```
<script> non inserire dati non sicuri/verificati qui </script>
   in uno script
<!-- non inserire dati non sicuri/verificati qui -->
  dentro un commento HTML
<div non inserire dati non sicuri/verificati qui =test />
  in un attributo name
< non inserire dati non sicuri/verificati qui href="/test" />
  in un nome di tag
<style> non inserire dati non sicuri/verificati qui </style>
  dentro il CSS
```

Nota: non eseguire codice javascript recuperato da una sorgente non fidata.

2) HTML escape prima di inserire contenuti non sicuri o dei dati non verificati: Alcuni importanti framework hanno al loro interno già dei metodi per le operazioni di HTML escape (trasformazione di simboli e caratteri speciali in codice ASCII), è necessario effettuare la sostituzione dei caratteri come mostrato:

- **&** cambiarlo con **&**;
- **<** cambiarlo con **<**;
- **>** cambiarlo con **>**;
- ¨ cambiarlo con **&quot**;

3) Javascript Escape: Bisogna effettuare l'escape anche del codice JavaScrip, degli URL che andiamo ad inserire nelle nostre pagine web o dei dati non verificati.

Come?

Utilizzando delle funzioni specifiche offerte da JavaScrip (ncodeURIComponent, encodeURI() ed escape()).

Un esempio dell'utilizzo della funzione può essere:

```
var uri = "http://w3schools.com/my test.asp?name=ståle&car=saab";
var res = encodeURIComponent(uri);
//res diventa
http%3A%2F%2Fw3schools.com%2Fmy%20test.asp%3
    Fname%3Dst%C3%A5le%26car%3Dsaab
```

4) Elementi da validare: Ci sono alcuni dati/elementi di un sito web che devono essere **sempre** validati prima di essere utilizzati all'interno della pagina web:

- URL
- dati ricevuti da variabili GET & POST
- windows.location(): funzione di javascript per reindirizzare l'utente ad un altra pagina web
- document.referrer: funzione di javascript per ritornare l'indirizzo della pagina precedente (non confondere con la cronologia)