/*
Title: SQL Injection & Blind SQL Injection BASE
Author: k4thryn
Translate: cobra90nj
Contact: cobra90nj[at]gmail[dot]com
*/

SQL Injection

Un attacco di tipo SQL Injection, può verificarsi quando un programma e stato scritto in malo modo, di conseguenza un utente potrà manipolare le query senza che esse siano state prima convalidate. Questo tipo di vulnerabilità e molto presente in pagine web con contenuti dinamici. In giro per il web ci sono molte guide su questo tipo di attacco, cosi come molte vulnerabilità pubblicate per diverse applicazioni web.

Un semplice esempio di SQL Injection è un modulo di login
HTML di base, in cui si inviano un nome utente e una
password:

Codice:

```
<form method="post" action="process_login.php">
<input type="text" name="username">
<input type="password" name="password">
</form>
```

Dato questo semplice modulo form, si può dedurre che il
modo più semplice (e anche peggiore) per il modo in cui
lo script "process_login.php" lavori sia quello di eseguire una query nel db simile a questa:

Codice:

```
"SELECT   id
FROM   logins
WHERE   username = '$username'
and   password = '$password'";
```

In questo caso se le variabili "$username" e "$password",
vengono prese direttamente dall' input dell' utente, lo script che permette il login, può essere facilmente ingannato facendogli creadere che una password valida sia stata fornita per accedere iniettando del codice SQL. Supponiamo che la seguente stringa sia stata fornita come password:

Codice:

```
` or '' = '
```

E gli forniamo come username "bob". Una volta che le variabili vengono eseguite, la query di sopra sarà simile a questa:

Codice:

```
"SELECT   id
```

```
FROM   logins
WHERE   username = 'bob'
and   password = '' or '' = '';
```

Questa query restituisce una riga perchè la clausola finale e:

Codice:

or '' = ''

Che restituisce sempre vero (una stringa vuota è sempre uguale a una stringa vuota).

<---------->

Bling SQL Injection

Negli ultimi anni, gli attacchi SQL injection sono in aumento. L'aumento del numero di applicazioni basate su database, insieme a varie pubblicazioni che spiegano il problema e come può essere sfruttato (sia in formato elettronico che stampati), hanno portato a molti attacchi e di conseguenza gli abusi di questo tipo di attacco.

Con l' aumento di questo tipo di attacco, sono stati fatti anche molti tentativi per trovare delle soluzioni al problema. La soluzione è, e sarà sempre quella di costruire programmi in modo più sicuro. Molti documenti sono stati pubblicati in materia di garantire allo sviluppo sicuro di applicazioni Web con l' utilizzo di un database, ma non molto è cambiato. Gli sviluppatori Web, ancora non sono consapevoli della sicurezza, ed i problemi continuano ad apparire.

Di conseguenza, gli esperti di sicurezza informatica per continuare a cercare altre misure che possono essere adottate nei confronti di questo problema. Sfortunatamente, la soluzione a questo problema prese forma nel cancellare i messaggi di errore SLQ. Poichè la maggiore parte di documenti che descrivono SQL Injection contano su raggruppare informazioni attraverso i messaggi di errore (perchè alcuni affermano che specifici compiti non possono essere completati senza messaggi di errori particolari), esperti di sicurezza hanno messo appunto che l' SQL Injection non può essere sfruttata senza dettagliati messaggi di errore (o il codice stesso).

Tutta via, nascondere i messaggi di errore e solo un' altra della realizzazzione della "Sicurezza Oscura", approccio che è stato ripetuttamente dimostrato nella storia come un cattivo modo di affrontare.

Lo scopo di questo documento è di dimostrare l' errore di chi la pensa cosi, e di presentare semplici tecniche utilizzate dagli attaccanti quando i messaggi di errore non sono presenti. Queste tecniche sono anche chiamate "Blind SQL Injection". Il nome porta a pensare, ottenere una SQL Injection anche quando non ci sono messaggi di errore. Essa riflette anche l' abilità che una persona possiede, che l' SQL Injection diventa cosi semplice che può essere fatta anche ad occhi chiusi.

Per capire come si realizza ciò, prima mostriamo come l' SQL Injection può essere facilmente individuata, sulla minima reazione del server. Poi andiamo oltre i modi ad arte per creare una sintassi di richiesta valida, che puo essere sostituita in seguito con un altra valida richiesta SQL. Infine dobbiamo discutere come UNION SELECT (spesso considerato l' elemento fondamentale di attacchi SQL Injection), può essere sfruttato senza determinati messaggi di errore. Essere bendati, la tecnica contenuta in questo documento da per scontato che abbiamo zero conoscenza dell' applicazione, il tipo di database, la struttura delle tabelle ecc, e che devono essere individuate in

tutto il processo di iniezione.

Da questo, noi speriamo di chiarire al meglio quelle vulnerabilità di livello applicativo devono essere manipolati da livello applicativo, e soluzioni basandosi su messaggi di errore da SQL Injection è totalmente inutile.

Due note importanti: In primo luogo questo documento non vuole essere una guida ne un tutorial di SQL. Questo documento non scende nei dettagli di specifici attacchi SQL Injection e di sfruttamento, e si presuppone che il lettore abbia comprensione di base di che cosa sono gli attacchi SQL Injection, ma vuole spiegare come questi posso essere pericolosi anche se non vengono visulizzati messaggi di errore. La seconda nota riguarda tutti gli esempi forniti in questo documento. Anche se gli esempi si basano su MS SQL Server e Oracle, le stesse tecniche possono essere applicate ad altra database.

Identificare le Infezioni
Per effettuare un SQL Injection il primo passo da eseguire è quello di identificarla. Per fare questo l' attaccante deve prima stabilire una sorta di indicazioni per quanto riguarda gli errori del sistema. Anche se i messaggi di errore non vengono mostrati, l' applicazione deve avere la capacità di separare il bene (le richieste valide) dal male (richieste non valide), e l' attaccante facilmente impara a identificare queste indicazioni, individuando gli errori, e individuare se sono di tipo SQL o no.

Riconoscimento degli Errori
Come prima cosa, dobbiamo capire i tipi di erroi che un utente malintenzionato può affrontare. Un applicazione web, può generare due diversi tipi di errori. Il primo tipo è quello generato dal server web, con il risultato di un eccezione nel codice. Se intatta, un eccezione fin troppo familiare "500 Internal Server Error". Di norma, un iniezione di cattiva sintassi SQL (le doppie virgolette, per esempio) dovrebbe causare questo tipo di errore, anche se altri tipi di sintassi possono portare ad una tale eccezione. Un semplice processo di soppressione di errore, sostituirà i testi predefiniti di questo errore con una pagina HTML fatta su misura, ma comunque osservando la risposta si capirà che è un errore sempre causato dal server. In alcuni casi, se si vuole assumere ancora più impegno per nascondere gli errori è per esempio, realizzare un reindirizzamento alla pagina principale o un messaggio di errore generico che non fornisce alcuna informazione.

Il secondo tipo di errore è generato dalla applicazione web, e solitamente indica più la programmazione. L'applicazione prevede determinati casi non validi e può generare loro un errore su misura specifico. Anche se questi tipi di errori dovrebbero venire normalmente come componente di una risposta valida (200 OK) loro possono essere anche sostituiti, con reindirizzamento, o altri mezzi per occultarli, è molto simile al "Internal Server Error".

Un semplice esempio per distinguere i due errori: Prendiamo due eCommerce applicazione, di nome A e B. Entrambe le applicazione usano una pagina chiamata "proddetails.asp.". Questa pagina si aspetta di ricevere un paramentro chiamato ProdID. Prende il ProID ricevuto, e recupera i dettagli del prodotto dal database, poi esegue alcune manipolazioni sul record restituito. Entrambe le applicazioni denominano "proddetails.asp" con un link, quindi ProdID dovrebbe sempre essere valido.
L' applicazione A è soddisfatta e di conseguenza non fa nessun controllo. Quanto un attacker interferisce con ProID, inserendo un id che non esiste nella tabella, verrà restituito un record vuoto. Molto probabilmente, poichè non si aspetta un record vouto, verrà restituito un messaggio di errore, generando: "500: Internal Server Error". L' applicazione B tuttavia verifica che il formato del recordset sia più grande di 0, prima di qualsiasi manipolazione di esso. Se non succede questo appare un messaggio di errore affermando "Nessun prodotto", o, se il programmatore vuole nascondere l' errore, l' utente verrà reindirizzato indietro con la lista dei prodotti.

Un utente malintenzionato che tenta di compiere un attacco Blind SQL Injection perciò, per prima genera delle richieste nulle, ed impara come l' applicazione gestisce gli errori, e che cosa ci si può aspettare quando un errore SQL si verifica.

Localizzare gli errori
Con la conoscenza dell' applicazione a memoria, ora l' attaccante può procedere con la seconda parte dell' attacco, individuando gli errori che sono risultati dagli input maneggiati. Per questo, le tecniche normali applicate, per sperimentare un attacco SQL Injection, sono come ad esempio l' aggiunta di parole chiabi SQL (OR, AND, ecc), oppure i META caratteri (ad esempio ; o '). Ogni parametro è esaminato individualmente, e la risposta è esaminata in modo approfondito per determinare se si è verificato un errore o meno. Utilizzando un proxy di intercettazione, o qualsiasi altro strumento, è facile individuare altri errori apperentemente nascosti. Ogni parametro che restituisce un messaggio di errore è sospetto, in quanto potrebbero essere vulnerabili a SQL Injection.

Come sempre, tutti i parametri sono esaminati individualmente, con il risultato della richiesta che sia valida. Ciò è estremamante importante in questo caso, come questo processo deve neutralizzare ogni possibile causa di errore, tranne l'iniezione in se. Il risultato di questo processo è di solito un lungo elenco di paramentri sospetti. Alcuni di questi paramentri possono essere vulnerabili a SQL Injection e possono essere sfruttati. Gli altri errori estranei a SQL possno essere eliminati. Il prossimo passo dell' attaccante è quello di individuare gli errori SQL nel disordine, che sono quelli effettivamente vulnerabili a SQL.

Identificare i parametri vulnerabili a SQL Injection
Per capire meglio come questi sono fatti, bisogna conoscere i tipo di base di dati in SQL. SQL normalmente ne fa uso di tre tipi principali: Numero, Data o Stringa. Ogni tipo ha diverse varianti, ma questi sono irrilevanti per il processo di iniezione. Ogni paramentro trasferito dalla applicazione web per la query SQL e considerato come uno di questi tipi, e di solito è molto semplice per determinare il tipo ("abc" è ovviamente una stringa, mentre 4 è in grado di essere sia un numero che una stringa).

Nel linguaggio SQL, i paramentri numerici vengono passato al server così come sono, mentre le stringhe o date sono passati racchiusi tra le virgolette. Per esempio:

Codice:

```
SELECT * FROM Products WHERE ProdID = 4
```

vs.

```
SELECT * FROM Products WHERE ProdName = 'Book'
```

Il server SQL, tuttavia, non si cura di quale tipo di espressione riceve, finche è di un tipo relativo. Questo comportamento da l' attaccante il miglior modo di identificare se un errore è davvero uno di SQL, o estranei. Usando i valori numerici, il modo più semplice per gestire questo è utilizzando operazioni aritmetiche di base. Per esempio, diamo un' occhiata alla seguente richiesta:

Codice:

```
/myecommercesite/proddetails.asp?ProdID=4
```

Questo test è molto semplice. viene fatto un tentativo iniettando 4 come paramentro. L' altro è fatto usando 3 +1 come parametro. Passando ad una richiesta di tipo SQL questi parametri il risultato delle due prove, saranno le seguenti query:

Codice:

```
(1) SELECT * FROM Products WHERE ProdID = 4'
(2) SELECT * FROM Products WHERE ProdID = 3 + 1
```

Il primo darà sicuramente un errore di sintassi. La secondaa, invece, verrà eseguita senza intoppi, con il risultato lo stesso prodotto richiesto (4 come ProID), il che indica che questo parametro è vulnerabile a SQL Injection.

Una simile tecnica può essere utilizzata per la sostituzione del parametro con una sintasssi SQL stringa. Ci sono solo due differenze. In primo luogo, i paramentri di tipo stringa sono contenuti all' interno di virgolette. In secondo luogo diversi server SQL utilizzano diverse sintassi per il concatenamento di stringhe. Per esempio Mcft SQL Server utilizza il segno + per concatenare le stringhe, Oracle invece || per lo stesso compito. Cambia la sintassi ma la tecnica è sempre quella. Per esempio:

Codice:

```
/myecommercesite/proddetails.asp?ProdName=Book
```

Questo test per SQL Injection, consiste nel sostituire il parametro ProdName, nella prima non è valido, come l' esempio di prima B', l' altro invece genererà un espressione valida, come B'+' (o B'||' anche con Oracle). Ciò risulta con le seguenti query:

Codice:

```
(1) SELECT * FROM Products WHERE ProdName = 'Book"
(2) SELECT * FROM Products WHERE ProdID = 'B' + 'ook'
```

Ancora una volta, la prima query è in grado di generare un errore di SQL, mentre la seconda torna lo stesso prodotto come la richiesta iniziale, con book come suo valore.

Allo stesso modo la stessa espressione può essere utilizzata per sostituire i parametri originali. Specifiche funzioni di sistema possono essere usate o per tornare un numero, o una stringa o una data (per esempio, in Oracle, sysdate restituisce una data espressione, che in SQL Server getdate() fa lo stesso compito). Ci sono anche altre tecniche per individuare se si verifica SQL Injection.

Come si può vedere l' iniezione di SQL è una operazione molto semplice, anche se non si visualizzano messaggi di errore dettagliati, che permetta l' attaccante di continuare con l' attacco.

<---------->

Eseguzione del Iniezione

Una volta che l' iniezione è stata capita dall' attaccante, il prossimo passo sarà cercare di sfruttarlo. Per questo, l' attaccante deve essere in grado di generare la sintassi valida, individuare il tipo di server, e costruire il necessario per sfruttarlo.

Individuare la giusta sintassi
Questo è di solito il processo più complesso per individuare una Blind SQL Injection. Se le query originale sono semplici, anche il sorgente è semplice. Tuttavia, se la query originale è complessa, richiede maggior tempo è molti più errori. In ogni caso, le tecniche di base per eseguire questi test, sono poche.

La sinstassi di base per iniziare un processo di identificazione standard è SELECT . . . WHERE le dichiarazioni, con questo parametro iniettato si hanno parte delle clausule WHERE. Al fine di ottenere una valida sintassi, l' attaccante deve essere in grado di aggiungere dei dati nuovi alle WHERE, in modo che restituirà diversi dati da quelli che dovrebbe. In alcune applicazione semplici aggiungendo OR 1=1, può spesso risultare esatto. In molti casi, tuttavia, questo non sarà sufficiente. Spesso, ci sono anche parentesi che devono essere chiusi, in modo che corrisponda alla stringa originariamente aperta. Un altro problema che si può verificare è che una query manomessa causerà un errore che non è distinguibile da un errore SQL (per esempio, se è previsto un solo record, e OR 1=1 causa al databese un ritorno di 1000 record, l' applicazione può generare un errore).

Dal momento che ogni clausola WHERE è fondamentale per restituire un insieme di espressioni VERE o FALSE, Uniti insieme a OR, AND e le parentesi, imparate la giusta sintassi che funge correttamente, è termina la query con tentativi diversi. Per esempio, l' aggiunta di 'AND 1=2' il risultato è falso, o l' aggiunta di 'OR 1=2' ha influenza zero, fatta eccezione per gli operatori in precedenza.

Con alcune iniezioni, semplicemente modificando la clausola WHERE può essere sufficiente. Con gli altri, come UNION SELECT, le iniezioni non sono sufficiente per modificare la clausola WHERE. L' intera dichiarazione SQL deve terminare correttamente, in modo che ulteriori sintassi possono essere aggiunte. Per questo, una semplice tecnica può essere utilizzata. Dopo l' attaccante deve trovare una valida combinazione di AND, OR 1=2, e 1=1, e può essere utlizzato anche il commento in SQL.

Questo simbolo rappresentato da due trattini di consegutivo (--), incarica il server SQL di ignorare il resto che si trova dopo di esso. Per esempio, guardiano una semplce query che ci permette di accedere ad una pagina, che sia il nome utente o la password nella query come questa:

Codice:

SELECT Username, UserID, Password FROM Users WHERE Username = 'user' AND Password = 'pass'

Individuando johndoe' -- come user, possiamo eseguire la seguente clausola WHERE:

Codice:

```
WHERE Username = 'johndoe' --'AND Password = 'pass'
```

In questo caso, non solo la sinstassi era giusta, ma l' autentificazione è stata ignorata. Tuttavia, guardiamo una diversa dichiarazione di clausola WHERE:

Codice:

```
WHERE (Username = 'user' AND Password = 'pass')
```

Si noti le parentesi intorno alla dichiarazione. Iniettando ('johndoe'--), che restituirà falso:

Codice:

```
WHERE (Username = 'johndoe' --' AND Password = 'pass')
```

La query ha le parentesi ineguagliate, per quindi non verrà eseguita.

Questo esempio, quindi, dimostra come il segno del commento può essere usato per determinare se la query termina correttamente. Se il segno del commento è stato aggiunto e non si verificano errori, significa che termina correttamente la query a destra di esso. Altrimenti, viene aggiunto un errore nella query.

Identificare il Database
Il prossimo passo che l' attaccande deve prendere, prima di iniziare a sfruttare la SQL Injection, è quello di individuare il tipo di database utilizzato. Fortunatamento, (per l' attaccante), si tratta di un compito molto più facile per trovare una sintassi valida.

Ci sono diversi tipi di iniezioni che l' attacker può utilizzare per individuare il database. Gli esempi che seguono si basano sulla distinzione tra Oracle e Mcft SQL Server. Simili tecniche vengono utilizzate per identificare altri tipi di database.

Un trucco molto semplice, che è stato già accennato in precedenza, è il concatenamento delle stringhe. Presumiamo che l' attaccante conosca abbastanza la sintassi, poi esso deve essere abile nell' aggiungere altre espressioni alla clausola WHERE, un confronto semplice della stringa pue essere fatto così:

Codice:

```
AND 'xxx' = 'x' + 'xx'
```

Sostituendo il + con ||, Oracle può essere facilmente distinguibile da MS SQL Server, o di altri database.

Un altro carattere utilizato, è il punto e virgola. In SQL, un punto e virgola è usato per concatenare parecchie dichiarazioni di SQL nella stessa linea. E possono essere usati anche all' interno del codice da iniettare, Oracle non consente di utilizzare il punto e virgola in questo modo. Ipotizziamo che con la sintassi del commento la query sia andato a buon fine e abbia restituito vero, con l' aggiunta di un punto e virgola con MS SQL Server, non ha nessuno inpluenza mentre con Oracle restituisce un errore. Inoltr, per verificare se altri comandi possono essere eseguiti dopo la virgola, puo essere fatto mediante la dichiarazione COMMIT (per esempio iniettando xxx' ; COMMIT --). Assumedo il significato che le dichiarazioni possono essere iniettati li, e non dovrebbe generare un errore.

Infine alcune espressioni di sistema possno essere sostituite con le funzioni di sistema che restituiscono il giusto valore. Poichè ogni database usa diverse alcune funzioni diverse, è facile individuare il tipo di database in questo modo (un pò come la funzione getdate(), con MS SQL Server VS.sysdate() con Oracle).

Sfruttando le Iniezioni
Con tutte queste informazioni a portata di mano, l' attaccante può ora procedere con l' iniezione. Mentre si sviluppa l' exploit, l' attaccante non ha più bisogno di messaggi di errore, e può basarsi

solo sul suo sapere.

Questo documento non vuole discute sulle tecniche SQL, in quanto già sono stati introdotti in altri documenti. L' unica tecnica che sfrutta una SQL Injection che andremo ad analizzare nel seguente capitolo è l' UNION SELECT.

<---------->

UNION SELECT Injection
Con la manomissione di SELECT . . . nel dichiarare WHERE può restituire molte volte risultati positivi, ma l' attaccante tenta sempre più spesso ad eseguire un UNION SELECT. La clausola WHERE diversamente dalla UNION SELECT permette di accedere a tutte le tabelle del sistema.

Poichè l' eseguzione dell' UNION SELECT richiede la conosceza del numero dei capi, nonchè il tipo ci ciascuno, e spesso considerato che questa non può andare a buon fine senza dei messaggi di errori particolari, spqcialmente quando il numero dei campi e grande. Il prossimo capitolo vuole mostrare una serie di tecniche molto facile con le quali potremo risolvere questo problema.

Prima di procedere, è importante che l' attaccante abbia la sintassi corretta. Nei capitoli precedenti, tuttavia, abbiamo già dimostrato come si può creare, e come è fatta. L' identificazione deve avvenire nel modo che cancella tutte le parentesi nella query, e permetta l' iniezione di UNION o altri comandi senza far subbire nessuna trasformazione. (Come spiegato in precedenza, per far questo possiamo far uso dell' iniezione del commento).

Una volta che la sintassi è corretta, un UNION SELECT può essere aggiunto alla query originale. L' UNION SELECT deve avere lo stesso numero di colonne con il rispettivo tipo come se fosse una iniezione originale, o verrà generato un errore.

Contare le colonne
Contare le colonne può sembrare quasi impossibile con la normale "UNION SELECT" SQL Injection. Il modo in vuoi viene fatto è semplicemente la creazione di un UNON SELECT all' inizio della query, e iniettando diversi numeri di campi (uno in più in ogni tentativo). Quando l' errore generato indica (Numero colonna mancante), è sostituito con il "corrispondente tipo di colonna mancante", quando il numero di colonne finisce, allora possiamo fare un passo in avanti. Quando però l' applicazione non restituisce errori, allora e totalmente inutile questo.

Per questo che deve essere utilizzata una diversa tecnica per identificare il numero di colonne, e sto parlando della clausola ORDER BY. Infatti aggiungendo una clausola ORDER BY alla fine della dichiarazione SELECT, l' ordine dei risultati cambia. Questo viene fatto normalmente specificando il nome del tipo di colonna (o i nomi di altri varie colonne).

Guardando quest' esempio che inietta un parametro "Numero carta di credito: 11223344) ORDER BY CCNum --, determina la seguente query di ricerca:

Codice:

SELECT CCNum FROM CreditCards WHERE (AccNum=11223344) ORDER BY CCNum --
AND CardState='Active') AND UserName='johndoe'

Questo solitamente è trascurato, tuttavia, l' ordine della clausola ORDER BY può avere una forma numerica. In questo caso la query si riferisce ad una colonna numerica, piuttosto che il suo nome. Ciò significa che l' iniezione di 11223344) ORDER BY 1 --, andrà a buon fine, e farebbe

esattamente lo stesso, visto che CCNum è il primo campo nell' risultato della query. Mentre iniettando 11223344) ORDER BY 2 --, tuttavia genera un messaggio di errore, perchè c' è solo un capo, il che significa che non può essere ordinati per il secondo capo.

Per tanto quando dobbiamo contare i numeri dei campi ORDER BY può essere molto utile. Un attaccante come prima clausola lancia ORDER BY 1. Poichè ogni SELECT deve avere almeno un capo questa dovrebbe funzionare. Se si riceve un messaggio di errore su questo, la sintassi deve essere manomessa finche non scompare. (Anche se impossibile) ma può anche restituire un errore di applicazione. In questo caso con l' aggiunta di ASC o DESC si può risolvere il problema. Una volta che la sintassi ORDER BY funzioni senza restituire errori, l' attaccante può cambiare l' ordine da colonna 1 a colonna 100 (o 1000 o qualsiasi cosa che sia valida). A questo punto, dovrebbe essere generato un errore, il che significa che l' enumerazione sta lavorando.

L' attaccante ora il metodo che gli permette di capire i numedi di colonne che esistono, e quelle che non esistono. L' attaccante deve semplicemente aumentare questo numero, una alla volta, fino a quando si riceve un messaggio di errore (poichè alcune colonne sono di un tipo che non permettono ordinamento, è consigliabile verifica uno o due ulteriori numeri, e di assicurarsi degli errori che si ricevono). Con questa tecnica il numero di campi è facilmente enumerato, e messaggi di errore non sono richiesti.

<---------->

Identificare i tipi colonne

Quindi ora conosciamo la sintassi per estrarre il numero dei campi, l' attaccante deve identificare i tipi di tutti i campi. Ottenere i tipi di tutti i campi tuttavia, può essere difficile. Essi devono corrispondere con tutte le query originale. Se si tratta di pochi campi, possono essere raggiunti con un attacco brutale, ma se ci sono parecchi, allora sorge un problema. Come già detto in precedenza ci sono tre tipi principali (numero, stringa, data) se sono 10 campi, significa che sono 310 (quasi 60.000) combinazioni. Iniettando 20 richieste al secondo, ci vorrà quasi un ora. E se ci sono ancora più campi questo rende il processo quasi impossibile.

Pertando deve essere utilizzata una tecnica più semplice quando si lavora senza errori. Ora possiamo utilizzare la chiave NULL in SQL. Invece di iniettare dei tipi di dati, (ad esempio, stringa o un numero intero), NULL assume la corrispondenza di tutti i tipi. E' quindi possibile iniettare una UNION SELECT, dove tutti i campi sono nulli, e quindi non ci dovrebbero essere tipi di errori. Esaminiamo un iniezione simile all' esempio precedente:

Codice:

```
SELECT CCNum,CCType,CCExp,CCName FROM CreditCards WHERE (AccNum=11223344 AND CardState='Active') AND UserName='johndoe'
```

L' unico cambiamento che si nota e che nel singolo campo CCNum sono stati aggiunti altri diversi, cosi ci sono più campi. Presupponiamo che l' attaccante abbia contato il numero di colonne del risultato di questa iniezione (4 nel nostro esempio), è ora di iniettare una semplice dichiarazione UNION con tutti NULL, con la slausola FROM che non generi errori di autorizzazioni (nuovamente viene fatto un tentativo di isolare ogni problema che tratteremo più avanti). Con MS SQL, la clausolo FROM può essere omessa. Questo è valido. Con Oracle, utilizzando una tabella chiamata doppia, può essere utile. Aggiungendo WHERE che valida sempre come falso (WHERE 1=2) garantisce che nessun record set contenente solo i valori NULL verrà restituito, eliminado ulteriori possibili errori (alcune applicazioni non riesco a gestire correttamente i valori NULL).

Possiamo guardare ora un esempio fatto sotto MS SQL Server, che vale anche per Oracle. Iniettando 11223344) UNION SELECT NULL,NULL,NULL,NULL WHERE 1=2 --, la query si trsformerebbe cosi:

Codice:

```
SELECT CCNum,CCType,CCExp,CCName FROM CreditCards WHERE (AccNum=11223344) UNION SELECT NULL,NULL,NULL,NULL WHERE 1=2 --AND CardState='Active') AND UserName='johndoe'
```

Le iniezioni di NULL hanno due scopi. Lo scopo principale è quello di ottenere una dichiarazione di UNION che non dia errori. Anche se questa UNION non fornisce ancora dati reali, ma ci dice che comunque, la sua iniezione funziona. L' altro scopo di questa UNION vuota è quello di ottenere un identificazione al 100% dal database (e usando una specifica nome di tabella, con FROM).

Una volta che la dichiarazione del NULL-based UNIONE sia operativa, si esegue un banale processo per identificare i tipi di ciascuna colonna. In ogni iterazione ogni singolo campo è esaminato al relativo tipo. Tutti e tre i tipi (numero, intero, stringa), sono esaminati per il campo, e uno di loro dovrebbe funzionare. Qindi le colonne voglio fino a tre tipi differenti, pittosto che tre colonna di numeri. Supponiamo che CCNum sia un intero, e che tutti gli altri campi siano stringhe, con le seguenti tipi di UNION possiamo individuare i tipi:

Codice:

```
 *  11223344) UNION SELECT NULL,NULL,NULL,NULL WHERE 1=2 --
  No Error - Sintassi esatta MS SQL Server Used. Proceeding.
* 11223344) UNION SELECT 1,NULL,NULL,NULL WHERE 1=2 --
  No Error - La prima colonna è un intero.
* 11223344) UNION SELECT 1,2,NULL,NULL WHERE 1=2 --
  Error! - - La seconda colonna non è un intero.
* 11223344) UNION SELECT 1,'2',NULL,NULL WHERE 1=2 --
  No Error - La seconda colonna è una stringa.
* 11223344) UNION SELECT 1,'2',3,NULL WHERE 1=2 --
  Error! - La terza colonna non è un intero.
* 11223344) UNION SELECT 1,'2','3',NULL WHERE 1=2 --
  No Error - La terza colonna è una stringa.
* 11223344) UNION SELECT 1,'2','3',4 WHERE 1=2 --
  Error! - La quarta colonna non è un intero.
* 11223344) UNION SELECT 1,'2','3','4' WHERE 1=2 --
  No Error - La quarta colonna è una stringa.
```

L' attaccante ha creato una valida UNION. Utilizzando sempre il numero più grande di quello precedente, quindi è possibile capire quale tipo vuole. Tutto quello che ora è esatto va sfruttato. Per questo che l' iniezione può essere usata per recuperare dati dalle tabelle di sistemi (o i dati nelle loro colonne). Questo documento tuttavia non entra nei dettagli, in quanto si possno trovare altri tutorial all' SQL Injection per la rete.

Dopo la lettura di questo documento il lettore capisce ora anche il motivo per cui SQL injection è una minaccia reale per qualsiasi sistema, con o senza la lettura dei messaggi di errore, e contanto solo su di essi non è abbastanza sicuro.