Secure Programming **Input Validation 3: Metacharacters** **Feb. 18, 2013**

Today

- Path manipulation
- Command Injection
- SQL Injection
- The failure of blacklisting

Path manipulation

A simple design: _____ user data in a file named _____

UI: “Add a note to your profile!”

Code:

```
String note = request.getParamter(“note”);
String name = request.getParamter(“name”);
File userFile = new File(“/var/www/funservice/users/” + name);
AddLineToFile(file, note);
```

Attacker’s “name”: _____

Attacker’s “note”: _____

Fix: _____ using a _____ (TPS)

Command Injection

If your program _____, you must be very careful!

Unix: _____ Windows: _____

Enables the attacker to _____

Ex:

```
String cmd = new String(“cat ” + name);
system(cmd);
```

Attacker’s “name”: _____

SQL Injection

Code:

```
query = “SELECT * FROM users WHERE name = ‘” + name + “’”;
```

Developer’s idea: (TPS)

Attacker's "name": _____

Fundamental Issue: _____ vs. _____. User input should only affect _____.

Solution: _____ (TPS)

```
query = "SELECT * FROM users WHERE name = ?";
PreparedStatement stmt = conn.prepareStatement(query);
Stmt.setString(1, name);
```

Not a solution (but a good idea): _____.

Use with parameterized queries _____.

An OK solution: _____. Parameterize queries: _____.

PHP: mysql_real_escape_string

```
$id = (int) $_GET['id'];
$pass = mysql_real_escape_string($_GET['pass']);
$result = mysql_query("SELECT id,name,pass FROM users WHERE id = $id AND pass =
'$pass' ");
```

Blacklisting #Fail

SQL Injection: just block _____?

Numeric fields: _____

```
$id = $_GET['id'];  // NOTE THE CHANGE! _____
...
if(preg_match('/\s/', $id))
    exit('attack'); // no whitespaces
if(preg_match('/[\'"]/', $id))
    exit('attack'); // no quotes
if(preg_match('/[\/\\\\]/', $id))
    exit('attack'); // no slashes
```

Attacker's response:

```
id=(1)and(1)=(0)union(select(null),group_concat(column_name),(null)from(information
_schema.columns)where(table_name)=(0x7573657273))#
```

Escaping quotes: magic_quotes_gpc, addslashes? Turns _____ into _____

GBK character set: 0xbf5c is 0xbf (¿) followed by 0x5c (\); ¿\. And 0xbf27 is 0x27 (') following a 0xbf (¿); ¿'.

So send: _____, and then you get 0xbf5c27.

But the first two bytes are one multi-byte char: _____. So the escaping method fails.

Much more: http://websec.wordpress.com/tag/sql-filter-bypass/