Web security: server-side attacks

Myrto Arapinis
School of Informatics
University of Edinburgh

November 14, 2017
Injection attack

**OWASP definition**
Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

We are going to look at:
- command injection attacks
- SQL injection attacks
Command injection: a simple example

- Service that prints the result back from the linux program whois

- Invoked via URL like (a form or Javascript constructs this URL):

  http://www.example.com/content.php?domain=google.com

- Possible implementation of content.php

```php
<?php
    if ($_GET['domain']) {
        <? echo system('whois '.$_GET['domain']); ?>
    }
?>
```
This script is subject to a **command injection attack**! We could invoke it with the argument `www.example.com; rm * http://www.example.com/content.php?domain=www.google.com; rm *`

Resulting in the following PHP

```php
<? echo system('whois www.google.com; rm *'); ?>
```
Defense: input escaping

```php
<? echo system('whois'.escapeshellarg($_GET['domain'])); ?>
```

`escapeshellarg()` adds single quotes around a string and quotes/escapes any existing single quotes allowing you to pass a string directly to a shell function and having it be treated as a single safe argument

<table>
<thead>
<tr>
<th>GET INPUT</th>
<th>Command executed</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.google.com">www.google.com</a></td>
<td>whois 'www.google.com'</td>
</tr>
<tr>
<td><a href="http://www.google.com">www.google.com</a>; rm *</td>
<td>whois 'www.google.com rm *'</td>
</tr>
</tbody>
</table>
Command injection recap

- Injection is generally caused when data and code share the same channel:
  - "\texttt{whois}\" is the code and the filename the data
  - \textbf{But ‘;}’ allows attacker to include new command

- \textbf{Defenses} include input validation, input escaping and use of a less powerful API
Web applications

Client
(HTML, JavaScript)

HTTP

Server
(PHP)

Database
(SQL)
Databases

<table>
<thead>
<tr>
<th>username</th>
<th>password</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>01234</td>
</tr>
<tr>
<td>bob</td>
<td>56789</td>
</tr>
<tr>
<td>charlie</td>
<td>43210</td>
</tr>
</tbody>
</table>

**User accounts**

- Web server connects to DB server:
  - Web server sends *queries* or *commands* according to incoming HTTP requests
  - DB server returns associated values
  - DB server can *modify/update* records
- SQL: commonly used database query language
SQL SELECT

Retrieve a set of records from DB:

```
SELECT field FROM table WHERE condition # SQL comment
```

returns the value(s) of the given field in the specified table, for all records where condition is true

Example:

<table>
<thead>
<tr>
<th>username</th>
<th>password</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>01234</td>
</tr>
<tr>
<td>bob</td>
<td>56789</td>
</tr>
<tr>
<td>charlie</td>
<td>43210</td>
</tr>
</tbody>
</table>

```python
SELECT password FROM user_accounts WHERE username='alice
```
returns the value 01234
SQL INSERT

Retrieve a set of records from DB:

```
INSERT INTO table VALUES record # SQL comment
```

adds the value(s) a new record in the specified table

Example:

```
username  password
alice     01234
bob       56789
charlie   43210
```

→

```
username  password
alice     01234
bob       56789
charlie   43210
eve       98765
```

```
INSERT INTO user_accounts VALUES ('eve', 98765)
```
Other SQL commands

- **DROP TABLE** `table`: deletes entire specified table

- Semicolons separate commands:

  **Example:**

  ```sql
  INSERT INTO user_accounts VALUES ('eve', 98765);
  SELECT password FROM user_accounts
    WHERE username='eve'
  ```

  returns 98765
SQL injection: a simple example

The web server logs in a user if the user exists with the given username and password.

login.php:

```php
$conn = pg_pconnect("dbname=user_accounts");
$result = pg_query($conn,
    "SELECT * from user_accounts
    WHERE username = " .$_GET['user'] ."'
    AND password = " .$_GET['pwd'] .";");
if(pg_query_num($result) > 0) {
    echo "Success";
    user_control_panel_redirect();
}
```

It sees if results exist and if so logs the user in and redirects them to their user control panel.
SQL injection: a simple example

Login as admin:
SQL injection: a simple example

Login as admin:

http://www.example.com/login.php?user=admin’#&pwd=f

pg_query(conn,
    "SELECT * from user_accounts
    WHERE username = 'admin' # ' AND password = 'f';");
SQL injection: a simple example

Login as admin:
http://www.example.com/login.php?user=admin’#&pwd=f

pg_query(conn,
    "SELECT * from user_accounts
    WHERE username = 'admin' # ' AND password = 'f';";)

Drop user_accounts table:
SQL injection: a simple example

Login as admin:
http://www.example.com/login.php?user=admin’#&pwd=f

```
pg_query(conn,
    "SELECT * from user_accounts
    WHERE username = 'admin' # ' AND password = 'f';"
);
```

Drop user_accounts table:
http://www.example.com/login.php?user=admin’;

```
DROP TABLE user_accounts #&pwd=f
pg_query(conn,
    "SELECT * from user_accounts;
    WHERE user = 'admin'; DROP TABLE user_accounts;
    # ' AND password = 'f';"
);
```
Defense: prepared statements

- Creates a template of the SQL query, in which data values are substituted
- Ensures that the untrusted value is not interpreted as a command

```php
$result = pg_query_params(
    conn,
    SELECT * from user_accounts WHERE username = $1 AND password = $2,
    array($_GET['user'], $_GET['pwd']));
```