Web Security: Vulnerabilities & Attacks

Slide credit: John Mitchell
Security User Interface
Safe to type your password?
Safe to type your password?
Safe to type your password?

thevest.com/

thevest.com/
Safe to type your password?
Mixed Content: HTTP and HTTPS
Mixed content and network attacks

• banks: after login all content over HTTPS
  – Developer error: Somewhere on bank site write
    `<script src="http://www.site.com/script.js"></script>`
  – Active network attacker can now hijack any session

• Better way to include content:
  `<script src="/www.site.com/script.js"> </script>`
  – served over the same protocol as embedding page
The Status Bar

- Trivially spoofable

```html
<a href="http://www.paypal.com/">
  onclick="this.href = 'http://www.evil.com/';">
  PayPal
</a>
```
Command Injection
Background

Client Browser

URI

foo.php

Web Page

Web Server

UID: www

PHP -> WEB PAGE
Quick Background on PHP

display.php: `<? echo system("cat ".$_GET['file']); ?>`

**IN THIS EXAMPLE**

- `echo expr`: evaluates expr and embeds in doc.
- `system(call, args)`: performs a system call in the working directory.
- " .....", ‘ ..... ’: String literal. Double-quotes has more possible escaped characters.
- `. `: (dot). Concatenates strings.
- `$_GET['key']`: returns value corresponding to the key/value pair sent as extra data in the HTTP GET request.

**LATER IN THIS LECTURE**

- `preg_match(Regex, Stiring)`: Performs a regular expression match.
- `proc_open`: Executes a command and opens file pointers for input/output.
- `escapeshellarg()`: Adds single quotes around a string and quotes/escapes any existing single quotes.
- `file_get_contents(file)`: Retrieves the contents of file.
display.php: `<? echo system("cat ".$_GET['file']); ?>`
Today we are learning about Web Security.

Content of notes.txt

display.php: `<? echo system("cat ".$_GET['file']); ?>`
Q: Assuming the script we’ve been dealing with (reproduced above) for http://www.example.net/display.php. Which one of the following URIs is an attack URI?

Hint: Search for a URI Decoder to figure out values seen by the PHP code.

- %3B -> “;”
- %20 -> “ “
- %2F -> “/”

 c. http://www.example.net/display.php?file=notes.txt%3B%20rm%20-rf%20%2F%3B%0A%0A
Command Injection

display.php: <? echo system("cat ".$_GET['file']); ?>

Q: Assuming the script we’ve been dealing with (reproduced above) for http://www.example.net/display.php. Which one of the following URIs is an attack URI? Hint: Search for a URI Decoder to figure out values seen by the PHP code.

(URIs decoded)

c. http://www.example.net/display.php?file=notes.txt; rm -rf /;
Q: Assuming the script we’ve been dealing with (reproduced above) for http://www.example.net/display.php. Which one of the following URIs is an attack URI? Hint: Search for a URI Decoder to figure out values seen by the PHP code.

(Resulting php)

a. `<? echo system("cat rm"); ?>`
b. `<? echo system("cat rm -rf /;"); ?>`
c. `<? echo system("cat notes.txt; rm -rf /;"); ?>`
d. `<? echo system("cat   "); ?>`
Injection

• Injection is a general problem:
  – Typically, caused when data and code share the same channel.
  – For example, the code is “cat” and the filename the data.
    • But ‘;’ allows attacker to start a new command.
Input Validation

• Two forms:
  – Blacklisting: Block known attack values
  – Whitelisting: Only allow known-good values

• Blacklists are easily bypassed
  – Set of ‘attack’ inputs is potentially infinite
  – The set can change after you deploy your code
  – Only rely on blacklists as a part of a defense in depth strategy
# Blacklist Bypass

## Table

<table>
<thead>
<tr>
<th>Blacklist</th>
<th>Bypass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disallow pipes and semi colons</td>
<td>Use the backtick operator to call commands in the arguments</td>
</tr>
<tr>
<td>Disallow pipes, semi colons and backticks</td>
<td>Use the $ operator which works similar to backtick</td>
</tr>
<tr>
<td>Disallow <code>rm</code></td>
<td>Use <code>unlink</code></td>
</tr>
<tr>
<td>Disallow <code>rm</code>, <code>unlink</code></td>
<td>Use <code>cat</code> overwrite existing files</td>
</tr>
</tbody>
</table>

- *Ad infinitum*
- *Tomorrow, newer tricks might be discovered*
Input Validation: Whitelisting

display.php:
```php
<?
if (!preg_match("/^[a-z0-9A-Z.]*$/", $_GET['file'])) {
    echo "The file should be alphanumeric.";
    return;
}
echo system("cat ".$_GET['file']);
?>
```

<table>
<thead>
<tr>
<th>INPUT</th>
<th>Passes?</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes.txt</td>
<td>Yes</td>
</tr>
<tr>
<td>notes.txt; rm -rf;</td>
<td>No</td>
</tr>
<tr>
<td>security notes.txt</td>
<td>No</td>
</tr>
</tbody>
</table>
Input Escaping

display.php:

```php
<?
echo system("cat ".escapeshellarg($_GET['file']));
?>
```

`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>notes.txt</td>
<td>cat 'notes.txt'</td>
</tr>
<tr>
<td>notes.txt; rm -rf /;</td>
<td>cat 'notes.txt rm -rf /;'</td>
</tr>
<tr>
<td>mary o'donnel</td>
<td>cat 'mary o''donnel'</td>
</tr>
</tbody>
</table>
Use less powerful API

- The system command is too powerful
  - Executes the string argument in a new shell
  - If only need to read a file and output it, use simpler API

  ```php
  display.php: <? echo file_get_contents($_GET['file']); ?>
  ```

- Similarly, the `proc_open` (executes commands and opens files for I/O) API
  - Can only execute one command at a time.
Recap

• Command Injection: a case of injection, a general vulnerability
• Defenses against injection include input validation, input escaping and use of a less powerful API
• Next, we will discuss other examples of injection and apply similar defenses
SQL Injection
Background

• SQL: A query language for database
  – E.g., SELECT statement, WHERE clauses

• More info
  – E.g., http://en.wikipedia.org/wiki/SQL
Running Example

Consider a web page that logs in a user by seeing if a user exists with the given username and password.

```php
$result = pg_query("SELECT * from users WHERE uid = " . $_GET['user'] . " AND pwd = " . $_GET['pwd'] . ";");
if (pg_query_num($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.
Client Browser

```
login.php?user=pikachu&pwd=password123
```

URI

Web Server

```php
login.php
connect to database using dbuser login.
Execute query with
$_GET['user']
$_GET['pwd']
```
Background

Client Browser

Web Server

URI

login.php?user=pikachu&pwd=password123

login.php
connect to database using dbuser login.
Execute query with
$_GET['user']
$_GET['pwd']

SELECT * from users WHERE uid='pikachu' AND pwd = 'password123';
**Background**

**Client Browser**
- login.php?user=pikachu&pdr=password123

**Web Server**
- login.php
  - connect to database using dbuser login
  - Execute query with 
    - $_GET['user']
    - $_GET['pwd']
  - SELECT * from users WHERE uid='pikachu' AND pwd = 'password123';

**Results:**
- 25 | pikachu | password123 | electric
Background

Client Browser

URI

login.php?user=pikachu&pwd=password123

Web Server

login.php
connect to database using dbuser login. Execute query with
$_GET["user"]
$_GET["pwd"]

Results:

| 25 | pikachu | password123 | electric |

Success and redirect to user control panel.

Query

SELECT * from users WHERE uid='pikachu' AND pwd='password123';

DB Server
Q: Which one of the following queries will log you in as admin?

Hints: The SQL language supports comments via ' ' characters.

b. http://www.example.net/login.php?user=admin--&pwd=foo  
c. http://www.example.net/login.php?user=admin'--&pwd=f
SQL Injection

login.php:
```
$result = pg_query("SELECT * from users WHERE
    uid = '".$_GET['user']."' AND
    pwd = '".$_GET['pwd']."';");
if (pg_query_num($result) > 0) {
    echo "Success";
    user_control_panel_redirect();
}
```

Q: Which one of the following queries will log you in as admin?
Hints: The SQL language supports comments via '---' characters.

SQL Injection

```
URI: http://www.example.net/login.php?user=admin'--&pwd=f

pg_query("SELECT * from users WHERE uid = 'admin'--' AND pwd = 'f';");
pg_query("SELECT * from users WHERE uid = 'admin';");
```

```
SQL Injection

Q: Under the same premise as before, which URI can delete the users table in the database?

a. www.example.net/login.php?user=;DROP TABLE users;--
b. www.example.net/login.php?user=admin%27%3B%20DROP%20TABLE%20users%20%20--%3B&pwd=f
c. www.example.net/login.php?user=admin;%20DROP%20TABLE%20users;%20--&pwd=f
d. It is not possible. (None of the above)
SQL Injection

Q: Under the same premise as before, which URI can delete the users table in the database?

a. www.example.net/login.php?user=;DROP TABLE users;--
b. www.example.net/login.php?user=admin'; DROP TABLE users;--&pwd=f
c. www.example.net/login.php?user=admin; DROP TABLE users; --&pwd=f
d. It is not possible. (None of the above)

pg_query("SELECT * from users WHERE
    uid = 'admin'; DROP TABLE users;--' AND
    pwd = 'f';");

pg_query("SELECT * from users WHERE uid = 'admin';
    DROP TABLE users;");
SQL Injection

• One of the most exploited vulnerabilities on the web

• Cause of massive data theft
  – 24% of all data stolen in 2010
  – 89% of all data stolen in 2009

• Like command injection, caused when attacker controlled data interpreted as a (SQL) command.

Data Source: Verizon DBIR 2011
Injection Defenses

• Defenses:
  – Input validation
    • Whitelists untrusted inputs to a safe list.
  – Input escaping
    • Escape untrusted input so it will not be treated as a command.
  – Use less powerful API
    • Use an API that only does what you want
    • Prefer this over all other options.
Input Validation for SQL

`login.php:`

```php
<?
if (!preg_match("/^[a-z0-9A-Z.]*$/", $_GET['user'])) {
    echo "Username should be alphanumeric."
    return;
}
// Continue to do login query
?>
```

<table>
<thead>
<tr>
<th>GET INPUT</th>
<th>PASSES?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pikachu</td>
<td>Yes</td>
</tr>
<tr>
<td>Pikachu'; DROP TABLE users--</td>
<td>No</td>
</tr>
<tr>
<td>O’Donnel</td>
<td>No</td>
</tr>
</tbody>
</table>
Given that our web application employs the input validation mechanism for usernames, which of the following URIs would still allow you to login as admin?

```php
pg_query("SELECT * from users WHERE
    uid = '" . $_GET['user'] . "' AND
    pwd = '" . $_GET['pwd'] . "';");
```

b. http://www.example.net/login.php?user=admin&pwd='%20OR%201%3D1;--
c. http://www.example.net/login.php?user=admin'--&pwd=f
d. http://www.example.net/login.php?user=admin&pwd='--
Input Validation for SQL

Given that our web application employs the input validation mechanism for usernames, which of the following URIs would still allow you to login as admin? (%3D -> “=“)

```php
pg_query("SELECT * from users WHERE
    uid = ".\$_GET['user']."" AND
    pwd = ".\$_GET['pwd']."";");
```

b. http://www.example.net/login.php?user=admin&pwd='%20OR%201%3D1--;'
c. http://www.example.net/login.php?user=admin'--&pwd=f
d. http://www.example.net/login.php?user=admin&pwd='--
Input Validation for SQL

Given that our web application employs the input validation mechanism for usernames, which of the following URIs would still allow you to login as admin?

a. `pg_query("SELECT * from users WHERE uid = ".$_GET['user']."' AND pwd = ".$_GET['pwd']."';")`;

b. `http://www.example.net/login.php?user=admin&pwa=' OR 1=1;-- pg_query("SELECT * from users WHERE uid = 'admin' AND pwd = ' OR 1 = 1;--'");`
Input Validation for SQL

Given that our web application employs the input validation mechanism for usernames, which of the following URIs would still allow you to login as admin?

```php
pg_query("SELECT * from users WHERE
    uid = ".$_GET['user']."' AND
    pwd = ".$_GET['pwd']."'";");
```

```php
pg_query("SELECT * from users WHERE
    (uid = 'admin' AND pwd = '') OR
    1 = 1;--';";");
```

1=1 is true everywhere. This returns all the rows in the table, and thus number of results is greater than zero.
Input Escaping

```php
$_GET['user'] = pg_escape_string($_GET['user']);
$_GET['pwd'] = pg_escape_string($_GET['pwd']);
```

`pg_escape_string()` escapes a string for querying the PostgreSQL database. It returns an escaped literal in the PostgreSQL format.

<table>
<thead>
<tr>
<th>GET</th>
<th>INPUT</th>
<th>Escaped Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td></td>
<td>Bob</td>
</tr>
<tr>
<td>Bob'; DROP TABLE users; --</td>
<td></td>
<td>Bob'; DROP TABLE users; --</td>
</tr>
<tr>
<td>Bob' OR '1'='1</td>
<td></td>
<td>Bob'' OR ''1''='''1</td>
</tr>
</tbody>
</table>
Use less powerful API:
Prepared Statements

• Create a template for SQL Query, in which data values are substituted.
• The database ensures untrusted value isn’t interpreted as command.
• Always prefer over all other techniques.
• Less powerful:
  – Only allows queries set in templates.
Use less powerful API:
Prepared Statements

```php
<?
# The $1 and $2 are a ‘hole’ or place holder for what will be filled by the data
$result = pg_query_params('SELECT * FROM users WHERE
    uid = $1 AND
    pwd = $2', array($_GET['user'], $_GET['pwd']));

# Compare to
$result = pg_query("SELECT * FROM users WHERE
    uid ="."$_GET["user"].("" AND
    pwd ="."$_GET["pwd"].("";");

?>
```
Recap

• SQL Injection: a case of injection, in database queries.
• Extremely common, and pervasively exploited.
• Use prepared statements to prevent SQL injection
  – **DO NOT** use escaping, despite what xkcd says.
• Next, injection in the browser.