CSE 127: Computer Security

SQL Injection

Vector Li

November 14, 2017
<table>
<thead>
<tr>
<th>When</th>
<th>Who</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-11-24 18:07:55.925434</td>
<td>idfoster</td>
<td>omg i luv chattr!</td>
</tr>
</tbody>
</table>
Chattr

When

2014-11-24 18:07:55.925434
<table>
<thead>
<tr>
<th>When</th>
<th>Who</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-11-24 18:08:19.99257</td>
<td>jmaskiew</td>
<td>lol hack the planet</td>
</tr>
</tbody>
</table>
A Magic Trick

- The functional specification only allowed seeing one user’s posts at a time
  - Current user’s posts on `view.php` without URL arguments
  - Any user’s posts with `view.php?user=USERNAME`
<table>
<thead>
<tr>
<th>When</th>
<th>Who</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-11-24 18:07:55.925434</td>
<td>idfoster</td>
<td>omg i luv chattr!</td>
</tr>
<tr>
<td>2014-11-24 18:08:19.99257</td>
<td>jmaskiew</td>
<td>lol hack the planet</td>
</tr>
</tbody>
</table>

Wow! very hack
http://127.0.0.1:8080/view.php?user=%27%20or%20%27%27%20=%20%27encodes

http://127.0.0.1:8080/view.php?user=user1' or ' ' = '
What is going on?
From Someone’s view.php:

```php
<?php

if (isset($_GET["user"])) {
    $user = ($_GET["user"]);
}

$exists = true;

$dbconn = pg_connect("host=localhost dbname=chattr user=student password=hacktheplanet");

$query = "SELECT * FROM messages WHERE name = '$user';
$result = pg_query($query);

... ...
?>
```
<?php

if (isset($_GET["user"])) {
    $user = ($_GET["user"]);
}

$exists = true;

$dbconn = pg_connect("host=localhost dbname=chattr user=student password=hacktheplanet");

$query = "SELECT * FROM messages WHERE name = '$user';";
$result = pg_query($query);

...

?>
http://127.0.0.1:8080/view.php?
user=%27%20or%20%27%27%27%20=%20%27

encodes

http://127.0.0.1:8080/view.php?user=' or ' = ' 

results in query

select * from posts where name = '' or '' = ''; 

always true
From Someone’s `view.php`:

```php
<?php

if (isset($_GET['user'])) {
    $user = ($_GET['user']);
}

$exists = true;

$dbconn = pg_connect("host=localhost dbname=chattr user=student password=hacktheplanet");

$query = "SELECT * FROM messages WHERE name = '$user'";
$result = pg_query($query);

... ... 
?>
```

untrusted user input inserted directly into query that is sent to the database
From Someone's `login.php`:

```php
<?php
...

else {
    $query = "SELECT username FROM chattrdb.users WHERE username='".$username.' AND password='".$password.'";
    $result = pg_query($conn, $query);
    if (!$row = pg_fetch_row($result))
    {
        session_unset();
    } else {
        $_SESSION['username'] = $username;
        header("Location: view.php?user=".$username);
    }
?>

...

<?php
```
SQL Injection

- **SQL Injection**: Inserting SQL fragment into query sent by an application to an SQL database
- Application assumes user input is **data**
- Databases parses user input as **code**
New Sony Hack Claims Over a Million User Passwords

By Doug Aamoth | June 02, 2011 | Add a Comment

Another of Sony’s websites has reportedly been hacked—this time around, the victim is SonyPictures.com. The group claiming responsibility for the breach, “LulzSec,” is the same group behind the recent PBS website hack.
SQL Injection Possibilities

- Dump the entire database (violate secrecy)
- Drop the entire database (deny availability)
- Modify database data (violate integrity)
<?PHP

... ...

$username = $_SESSION['username'];
$timeStamp = date("Y-m-d H:i:s");
$userMessage = $_POST['TEXT'];

//Insert into database
$insertQuery = "INSERT INTO messages (username, time, message)
VALUES ('$username', '$timeStamp', '$userMessage')";

$result = pg_query($insertQuery);

//Navigate to view the posts
header('Location: view.php?user=' . $username)

?>

Delete user’s posts through SQL injection
Constraints

❖ Easiest case (for attacker): known application code and database schema, direct access to query results
  • HW2: view.php?user=

❖ Hardest case (for attacker): unknown code and schema, one bit of output per query
  • 1 bit output: success or failure
  • HW2: login.php
View users’ passwords through SQL injection
SQL Injection

- **SQL Injection**: Inserting SQL fragment into query sent by an application to an SQL database

- Application assumes user input is data

- Databases parses user input as code

- Attacker gains ability to submit SQL directly to backend database on behalf of *application database* user
Web Application Architecture

User system
*Client side*

Application server
*Server side*

DBMS server
*Database*
User Domains

❖ Operating system
  • **HW2**: root and student

❖ Database
  • **HW2**: postgres, student, chattr

❖ Application
  • **HW2**: idfoster, jmaskiew (in examples)
SQL Injection Privileges

❖ Server-side application process connects to database as a particular database user (application database user)

```
<?php
$db_host = 'localhost';
$db_user = 'student';
$db_pass = 'hacktheplanet';
$db_name = 'chattr';

$conn = pg_connect (
    "host=$db_host dbname=$db_name user=$db_user password=$db_pass")
```

❖ Attacker gains direct access to database with application database user privilege
Mitigation

❖ Sanitize user input
  • Escape SQL delimiters to input treated as quote

❖ Use prepared statements
  • Complete separation of control and data
  • Preferred way
Sanitizing User Input

❖ Escape special characters
  • E.g. change ' to '' (' ' treated as single ' inside quote)

❖ Easy to get wrong
  • With above rule \ in input becomes " which closes quote
  • Each database has its own special quoting rules

❖ Use DB-specific string escape function instead
  • PHP & PostgreSQL: pg_escape_literal
  • PHP & MySQL: mysqli_real_escape_string
Prepared Statements

❖ Separate control and data

❖ Prepare: define statement with parameters

    pg_prepare($conn, "get_posts",
               'select * from posts where name = $1');

❖ Execute: execute query with given parameters

    $rows = pg_execute($conn, "get_posts",
                      array($user));
Prepared Statements vs Sanitizing Input

❖ Economy of Mechanism argues against sanitizing

- String parsing implemented in database code
  - Tempting to add additional escape mechanisms as a “feature”
- String escaping implemented in database connector
  - Database connector usually maintained by third party (not database vendor)
- Two mechanisms must be exactly in sync

❖ Prepared statement escaping (if any) is at lower level

- Handled by common library maintained by DB vendor
- Invisible to user
Escaping Problems

❖ Robustness principle (Postel’s law): “Be conservative in what you do, be liberal in what you accept from others.” (RFC 793)

❖ Historically considered good protocol design philosophy

❖ Security problems can occur when assumptions at interfaces of two systems differ
The Bigger Problem

User system
Client side

Application server
Server side

DBMS server
Database
The Bigger Problem

- Application server accesses with database on behalf of application user
- In most cases application users have distinct privileges
- Application uses database as single database user
- This application database user must have *union* of all user’s privileges in order to implement functionality
<?php
$db_host = 'localhost';
$db_user = 'student';
$db_pass = 'hacktheplanet';
$db_name = 'chattr';

$conn = pg_connect (  
    "host=$db_host dbname=$db_name
    user=$db_user password=$db_pass"
  )
The Bigger Problem

- Compromising application gives attacker access to database as application database user

- **Best case:** attacker gains union of all application user’s access privileges to data

- **Worst case:** application database user is DB superuser
DB User = App User?

- Database user domain managed by database admin
- Application user domain managed by application code
- No easy way to map application user to database user
The Bigger Problem

User system  
*Client side*

Application server  
*Server side*

DBMS server  
*Database*
The Bigger Problem

- Application similar to setuid executable in Unix
- Application code must ensure all interaction consistent with security policy
- Application code part of TCB
- Is there a better way?