Special Topics in Security
ECE 5698

Engin Kirda
ek@ccs.neu.edu

Northeastern University

Admin News and Stuff

- I'll have Quiz 1 corrected by this week
- Today after class, Challenge 3 goes online
  - Should be easy for those who have some experience with SQL injections

SQL Injection Solutions

SQL Injection Solutions

- Let us use pressRelease.jsp as an example. Here our code:

```
String query = “SELECT title, description from pressReleases
    WHERE id= “+ request.getParameter(“id”);
Statement stat = dbConnection.createStatement();
ResultSet rs = stat.executeQuery(query);
```

- The first step to secure the code is to take the SQL statements out of the web application and into DB

```
CREATE PROCEDURE getPressRelease @id integer
AS
SELECT title, description FROM pressReleases WHERE
Id = @id
```

SQL Injection Solutions

- Now, in the application, instead of string-building SQL, call stored procedure:

```
CallableStatements cs = dbConnection.prepareCall(“{call
    getPressRelease(?)}”);
cs.setInt(1,Integer.parseInt(request.getParameter(“id”)));
ResultSet rs = cs.executeQuery();
```

- In ASP.NET, there is a similar mechanism

Simple Parameter Injection Example

- Perl script that lists (embeds in HTML) the directory contents by calling the shell *ls* command:

```
…
$query = new CGI;
$directory = $query->param(“directory”);
#Call the ls command in the shell using back ticks
$directory_contents = `ls $directory`;
print “
<html><body>
$directory_contents
</body></html>“;
```

Unvalidated input!

Simple Parameter Injection Example

- If the user enters a ; cat /etc/passwd as the directory, she can gain access to the contents of the passwd file as well!
  - The shell command in the script becomes ls ; cat /etc/passwd
- How can such a simple attack be prevented?
  - Do not use shell commands directly in Web scripts
  - Filter out characters such as | ; * > < etc. that have a special meaning for the shell
  - Can you think of other special characters?

Blind SQL Injection

- A typical countermeasure is to prohibit the display of error messages. But, is this enough?
  - No, your application may still be vulnerable to blind SQL injection
- Let’s look at an example. Suppose there is a news site
  - Press releases are accessed with pressRelease.jsp?id=5
  - An SQL query is created and sent to the database:

    select title, description FROM pressReleases where id=5;
  - Any error messages are smartly filtered by the application

Blind SQL Injection

- How can we inject statements into the application and exploit it?
  - We do not receive feedback from the application so we can use a trial-and-error approach
  - First, we try to inject pressRelease.jsp?id=5 AND 1=1
  - The SQL query is created and sent to the database:

  select title, description FROM pressReleases where id=5 AND 1=1
  - If there is an SQL injection vulnerability, the *same* press release should be returned
  - If input is validated, id=5 AND 1=1 should be treated as value

Blind SQL Injection

- When testing for vulnerability, we know 1=1 is always true
  - However, when we inject other statements, we do not have any information
  - What we know: If the same record is returned, the statement must have been true
  - For example, we can ask server if the current user is “h4x0r”:

pressRelease.jsp?id=5 AND user_name()=‘h4x0r’

  - By combining subqueries and functions, we can ask more complex questions (e.g., extract the name of a database character by character)

Second Order SQL Injection

- SQL is injected into an application, but the SQL statement is invoked at a later point in time
  - e.g., Guestbook, statistics page, etc.
- Even if application escapes single quotes, second order SQL injection might be possible
  - Attacker sets user name to: john’--, application safely escapes value to john’’– (-- is used for expressing comments in SQL Server)
  - At a later point, attacker changes password (and “sets” a new password for victim ***john)***:
    - update users set password= … where *database_handle*(“username”)=‘john’--‘

So what can you do to protect yourself?

- A common solution are so-called web application firewalls
  - Software that sits between application and user
  - Application that blocks malicious requests
  - An example is *mod_security* for Apache
- Web application firewalls are often expensive, can sometimes be evaded, and can sometimes be fingerprinted

Web Application Firewalls

- Fingerprinting: Sending malicious requests and seeing how application reacts
  - E.g., send ' “ < ? # - | ^ *
  - Why are these characters “malicious”?
  - Look for server header entry, status code
  - E.g., 501 Method Not Implemented (*mod_security*)
  - E.g., 999 No Hacking (*WebKnight*)

Web Application Firewalls

```
> curl -i www.microsoft.com/en/us/default.aspx
HTTP/1.1 200 OK
Cache-Control: public
Content-Type: text/html; charset=utf-8
Expires: Tue, 09 Nov 2010 11:02:47 GMT
Last-Modified: Mon, 08 Nov 2010 14:40:18 GMT
ETag: 634247952180000000
Server: Microsoft-IIS/7.5
X-AspNet-Version: 4.0.30319
VTag: 279956042600000000
P3P: CP="ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo
      OUR SAMo CNT COM INT NAV ONL PHY PRE PUR UNI"
X-Powered-By: ASP.NET
Date: Tue, 09 Nov 2010 10:52:46 GMT
Content-Length: 74444
...
```

Web Application Firewall

```
> curl -i www.microsoft.com/../../WINNT/system32/cmd.exe?d

HTTP/1.1 403 Forbidden

Content-Type: text/html; charset=us-ascii

Server: Microsoft-HTTPAPI/2.0

Date: Mon, 08 Nov 2010 16:40:08 GMT

Cneonction: close

Content-Length: 312


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
      4.01//EN""http://www.w3.org/TR/html4/strict.dtd">

<HTML><HEAD><TITLE>Forbidden</TITLE>

<META HTTP-EQUIV="Content-Type" Content="text/html; charset=us-ascii”>
```

Web Application Firewalls

- Automated WAF detection
  - Waffit (WAFW00F)
  - http://code.google.com/p/waffit/

Web Application Firewalls

```
> python wafw00f.py http://www.microsoft.com/austria
                                 ^     ^
        _   __  _   ____ _   __  _    _   ____
       ///7/ /.' \ / __////7/ /,' \ ,' \ / __/
      | V V // o // _/ | V V // 0 // 0 // _/
      |_n_,'/_n_//_/   |_n_,' \_,' \_,'/_/
                                <
                                 ...'

    WAFW00F - Web Application Firewall Detection Tool

    By Sandro Gauci && Wendel G. Henrique

Checking http://www.microsoft.com/austria
The site http://www.microsoft.com/austria is behind a Citrix NetScaler
Number of requests: 5
```

Web Application Firewall

```
> python wafw00f.py http://www.tuwien.ac.at

                                 ^     ^

        _   __  _   ____ _   __  _    _   ____
       ///7/ /.' \ / __////7/ /,' \ ,' \ / __/
      | V V // o // _/ | V V // 0 // 0 // _/
      |_n_,'/_n_//_/   |_n_,' \_,' \_,'/_/
                                <
                                 ...'

    WAFW00F - Web Application Firewall Detection Tool

    By Sandro Gauci && Wendel G. Henrique

Checking http://www.tuwien.ac.at
The site http://www.tuwien.ac.at is behind a Imperva
Number of requests: 8
```

Web Application Firewall Evasion

- So are WAFs “unbreakable”?
  - It may not be easy to by pass them, but an attacker can be creative
  - For example, different encodings can be used (as we will later see in other vulnerabilities)
  - <script>alert(‘This is an attack’)</script> may become:
    - `%u003c%uff53%uff43%uff52%uff49%uff50%uff54%u003e%uff41%uff4c%uff45%uff52%uff54%uff08%u02b9%uff38%uff33%uff33%u02b9%uff09%u003c%u2215%uff53%uff43%uff52%uff49%uff50%uff54%u003e`

Web Application Firewall Evasion

- Encodings can be mixed (making it more difficult for firewall to filter)
- Random garbage can be inserted into comments
- Example string:
  - `page.asp?id=2%20or%202%20in%20(/*IDS*/%73/*and*/%65/*WAF*/%6C/*evasion*/%65/*is*/%63/*pretty*/%74/*easy*/%20%75/*if*/%73/*you*/%65/*do*/%72/*it*/)/*right*/%2D%2D`
- That is: `page.asp?id=2 or 2 in (select user)--`

Discovering “clues“ in HTML code

- Developers are notorious for leaving statements like FIXME's, Code Broken, Hack, etc... inside the source code. Always review the source code for any comments denoting passwords, backdoors, or that something doesn't work right.

- Hidden fields (<input type=“hidden“…>) are sometimes used to store temporary values in Web pages. These can be changed with ease (Hidden Field Tampering!)

Session Management

- HTTP is a stateless protocol:
  it does not “remember” previous requests
- web applications must create and manage sessions themselves
- session data is
  - stored at the server
  - associated with a unique Session ID
- after session creation, the client is informed about the session ID
- the client attaches the session ID to each request

Session Management and Authentication

- authentication: strongly connected to session management
- the authentication state is stored as session data
- this makes the session ID a popular target for attackers:
  - stealing the ID of an active, authenticated session allows impersonation of the victim
- protect the session ID!

Session ID Transmission

- three possibilities for transporting session IDs
- encoding it into the URL as GET parameter; has the following drawbacks
  - stored in referrer logs of other sites
  - caching
  - visible in browser location bar (bad for internet cafes...)
- hidden form fields: only works for POST requests
- cookies: preferable, but can be rejected by the client

Cookies

- token (“name=value“) that is stored on client machine
- set by server
- uses a single domain attribute
  - cookies are only sent back to servers whose domain attribute matches

Cookies

- Non-persistent cookies
  - are only stored in memory during browser session
  - good for sessions
- Secure cookies
  - cookies that are only sent over encrypted (SSL) connections
- Only encrypting the cookie over insecure connection is useless
  - attackers can simply replay a stolen, encrypted cookie
- Cookies that include the IP address
  - makes cookie stealing harder
  - breaks session if IP address of client changes during that session

Session Attacks

- targeted at stealing the session ID
- Interception:
  - intercept request or response and extract session ID
- Prediction:
  - predict (or make a few good guesses about) the session ID
- Brute Force:
  - make many guesses about the session ID
- Fixation:
  - make the victim use a certain session ID
- the first three attacks can be grouped into “Session Hijacking“ attacks

Session Attacks

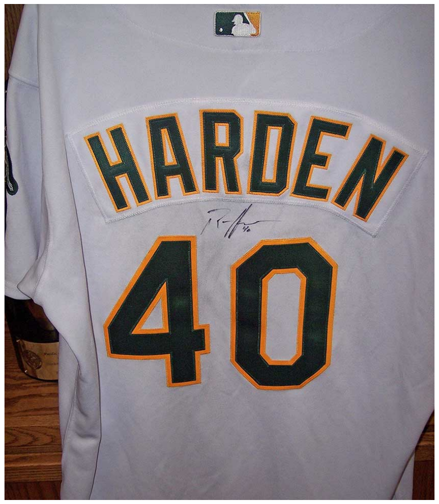
- preventing Interception:
  - use SSL for each request/response that transports a session ID
  - not only for login!
- Prediction:
  - possible if session ID is not a random number...

Prediction Example

- Suppose you are ordering something online. You are registered as user *john*. In the URL, you notice:
  - www.somecompany.com/order?s=john05011978
  - What is *s*? It is probably the session ID…
  - In this case, it is possible to deduce how the session ID is made up...
- Session ID is made up of user name and (probably) the user‘s birthday
  - Hence, by knowing a user ID and a birthday (e.g., a friend of yours), you could hijack someone‘s session ID and order something

Harden Session Identifiers

- Although by definition unique values, session identifiers must be more than just unique to be secure
  - They must be resistant to brute force attacks where random, sequential, or algorithm-based forged identifiers are submitted
  - By hashing the session ID and encrypting the hash with a secret key, you create a random session token and a signature (!)
  - Session identifiers that are truly random (hardware generator) for high-security applications



Another Session Attack

- additional attacks can be made possible by flawed credential management functions
  - e.g., weak “remember my password“ question
- rule of thumb:
  - use existing solutions for authentication and session management
  - never underestimate the complexity of authentication and session management...

Session Fixation Attacks

- As we discussed, web security is mainly focused on preventing the attacker from obtaining session credentials
  - This approach, however, ignores the possibility of the attacker “issuing” a session ID to the user’s browser
  - The browser then uses a “chosen” session
  - In a session fixation attack, the attacker fixes the user’s session ID before the user even logs into the target server
  - What does this mean? The session ID does not have to be stolen

Session Fixation Attacks

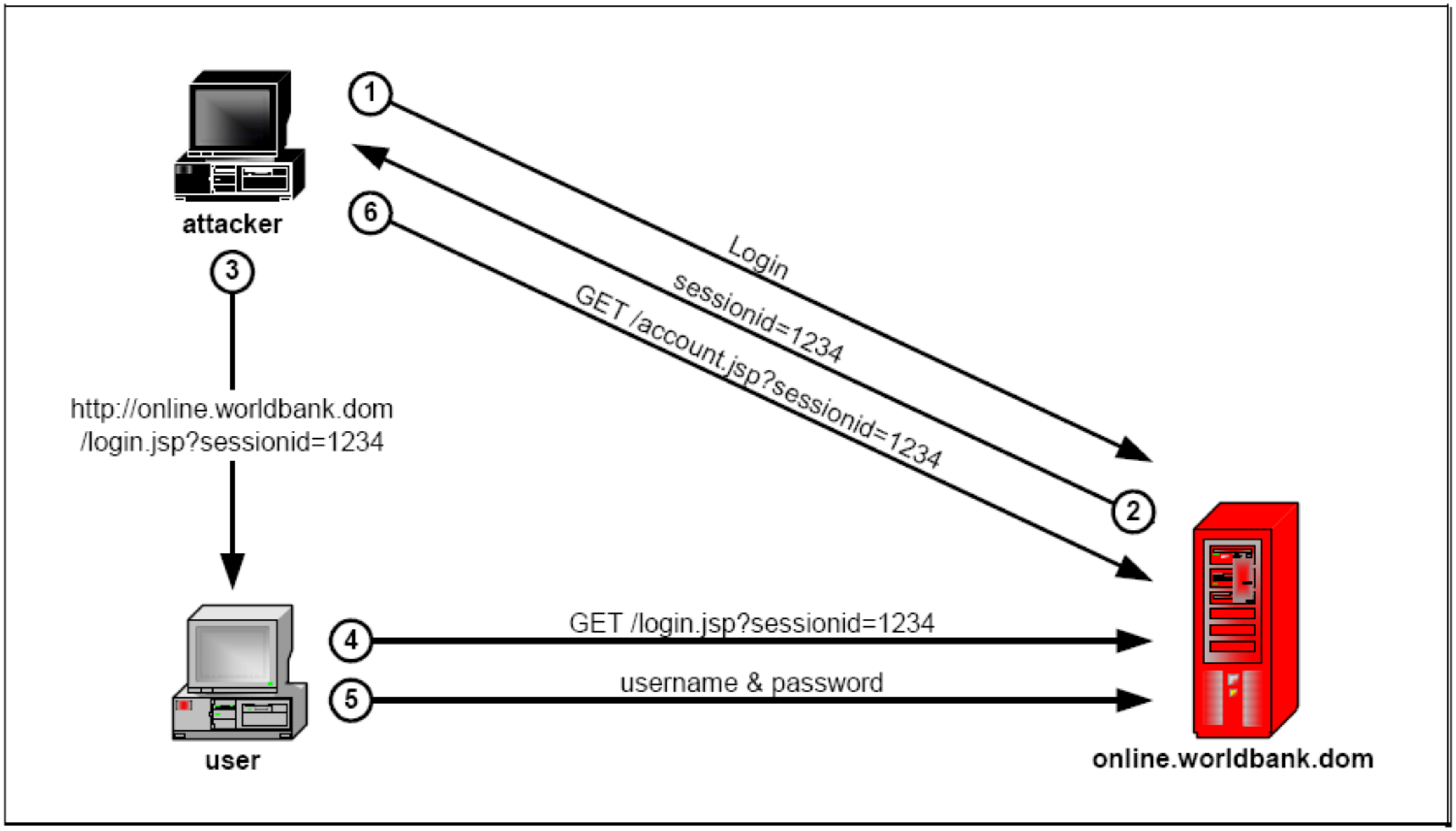
- Suppose we have a bank *online.worldbank.com*
  - When the web site is accessed, a session ID is transported via URL parameter *sessionid*

  1) First, attacker logs in and is issued a sessionid=1234
  2) Then, attacker sends *sessionid* to victim and tricks him into clicking on it

  http://online.worldbank.com/login.jsp?sessionid=1234

  3) The user clicks on the link and is taken to the banking application login page
  4) The web application sees that a session has been assigned and does not issue a new one. Session is bound to new user
  5) Attacker can access victim's account

Session Fixation Attacks

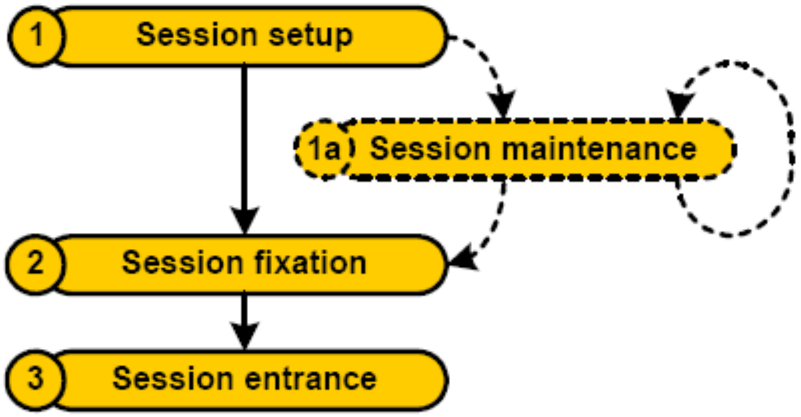

attacker
1
Login
6
sessionid=1234
GET /account.jsp?sessionid=1234
2
3
http://online.worldbank.dom
/login.jsp?sessionid=1234
4
GET /login.jsp?sessionid=1234
5
username & password
user
online.worldbank.dom

Session Fixation Attacks

- The example we gave is simple and has shortcomings for attacker
  - Attacker needs to be a legitimate user on the server
  - The victim needs to be tricked into clicking on link (before session times out)
- Generally, a session fixation attack is a three-step process
  1) Session setup
  2) Session fixation
  3) Session entrance

Session Fixation Attacks

- In session startup, attacker sets up a so-called “trap session”. Session needs to be kept “alive” by sending requests
- Next, attacker needs to introduce her session ID to the victim’s browser
- Attacker needs to wait until the victim logs in and then enter the victim’s session.



Session Fixation Attacks

- STEP 1: Session setup
  - Session management mechanisms can be classified as: *permissive* and *strict*
  - Permissive are those that accept arbitrary session IDs from browser (e.g., Macromedia JRun Server, PHP)
  - Strict are those that only accept sessions that they have created (e.g., MS IIS)
  - Permissive systems are easy to attack because the attacker can construct her session ID and “propose” it to the target server; requires no trap session maintenance

Session Fixation Attacks

- STEP 2: Session fixation
  - Attacker can use several methods to transport the trap session to the victim’s browser
  - One way is to trick user into clicking a URL, for example:
    http://online.worldbank.com/login.jsp?sessionid=1234
  - Another possibility is to prepare a login page (e.g., phishing page) where session ID is embedded as a hidden form field
  - additional method for session fixation attack: cookies

Session Fixation Attacks

- The attacker needs to install the trap session ID cookie on the victim's browser
  - However, browser will only accept cookie assigned either to the issuing server or the issuing server's domain. *attacker.com* cannot set cookie for *online.worldbank.com*
  - Hence, the attacker can

  1) Use a client-side script to set a cookie

     e.g., JavaScript: document.cookie="sessionid=1234"

  2) Use HTML<META> tag with *Set-Cookie* attribute

  3) Use *Set-Cookie* HTTP response header (advanced, requires hacking DNS)

Session Fixation Attack Solutions

- Preventing session fixation is the responsibility of the web application, not the web server
  - Only the web application can implement effective protection. The web server (e.g., Tomcat) only needs to make sure that session IDs cannot be brute-forced or guessed
  - A common-denominator for session fixation attacks is that victim needs to login. Hence, there should be a *forceful prevention of logging* into a chosen session
  - If possible, web application should issue session IDs *only* after successful authentication
  - alternatively, regenerate the session ID after login

Session Fixation Attack Solutions

- Session ID usage should be restricted
  - Bind to IP (network address): usual problems
  - Session ID should be bound to SSL client certificate for highly-critical applications (was the session ID established using certificate?)
  - Session destruction (timeout or logging out)
  - User must have option to log out (destroy current session and any other session)
  - There should be *absolute session timeouts* in order to prevent attacker from repeatedly sending requests to maintain session (can become inconvenient for the user)

Session Fixation VS Hijacking

- Timing
  - Session Fixation: Victim’s browser is attacked *before* she logs into the target server
  - Session Hijacking: Victim’s browser is attacked *after* she logs into the target server