Attacks Against Websites

Tom Chothia
Computer Security, Lecture 11

Typical Web Setup

HTTP website:
<form action="http://site.com/index.jsp" method="GET">
  Email: <input type="text" name="email">
  <input type="submit" value="Submit">
</form>

http://site.com/index.jsp?email=x@y.com

PHP page reads and processes:

```php
<?php
  $email=$_GET['emailAddress'];
  mysql_query("INSERT INTO emailsTable VALUE ("$email")");
?>

<b>Your e-mail has been added</b>
```

Authenticating web users after log in

- **IP address-based**
  - NAT may cause several users to share the same IP
  - DHCP may cause same user to have different IPs

- **Certificate-based**
  - Who has a certificate and what is it, and who will sign it?

- **Cookie-based**
  - The most common

Simple authentication scheme

- A Capability List access control system.
- The Web Application:
  - Verifies the credentials, e.g., against database
  - Generates a cookie which is sent back to the user
    - Set-Cookie: auth=secret
- When browser contacts the web site again, it will include the session authenticator
  - Cookie: auth=secret
Eavesdropping

• If the connection is not encrypted, it is possible to eavesdrop, by
  – ISP,
  – anyone on the route
  – Anyone on your local network, e.g. using the same wi-fi.

• Log in should be done using SSL (https)
  • This makes it impossible to eavesdrop the password.
  • After login many websites drop to http
  • Big security flaw

Steal the Cookie

• So the attacker doesn’t need the username and password
  – Just the cookie

• If the website uses https (SSL) it’s safe

• But many websites drop back down to http after a secure login.

Countermeasures

• Use https (SSL) all the time.
  – Last year I demo-ed this on Facebook, this year it no longer works.

• Set the secure flag, cookie is sent only over secure connections:
  Cookie secureCookie =
    new Cookie("credential", c);
    secureCookie.setSecure(true);

SQL Injection Attacks

http://www.shop.com/page?do=buy&product=17453

Web server might look up "17453" in a SQL database using:

... SELECT * FROM products WHERE (code='17453')
... INSERT INTO sales VALUES (id, customer, 17453)
...

SQL Injection Attacks

http://www.eshop.co.uk?action=buy&product=X

=> SELECT * FROM products WHERE (code=‘X’)

What else could we use for X?

X= ‘} ; DROP TABLE products; --
=> SELECT * FROM products WHERE (code=");
  DROP TABLE products; -- ‘}
SQL Injection Attacks

Secret Item: dh2*%Bgo

=>

SELECT * FROM users WHERE (item = 'dh2*%Bgo')

If found, then item details are given.

SQL Injection Attacks

Secret Item: ' OR '1'='1' --

=>

SELECT * FROM users WHERE (item = '' OR '1'='1') --

1 does equal 1! Therefore I will return all item's details

Stopping SQL attacks

You can stop SQL attacks by checking the input.

– In PHP mysqli_real_escape_string

E.g.: \\\
\"\'OR \"1\"\"=\"1\"\" -- maps to
\\\n\\\n\"\"\"\"\"OR \"1\"\"=\"1\"\" --"\n
All user controlled data must be "sanitized" (this can be hard).

Not Just Websites.

Any source of data can be used for an SQL attack

• 2D bar

• RFID cards
URL hacking

- The user can type anything they want into the URL bar, or even form the request by hand.

  http://nameOfHost/filePath

- Attacker can try to guess filenames, – Guessable directory names will be found.

Path Transversal

- The user can type anything they want into the URL bar, or even form the request by hand.

  http://nameOfHost/../../../etc/shadow

- If the webserver is running with root permission this will give me the password file.

Path Transversal: Fix

- Use access control settings to stop Path Transversal.

- Best practice, make a specific user account for the webserver.

- Only give that account access to public files.

Missing Function Level Access Control

Query strings are used to tell dynamic webpages what to do

http://myWebShop.com/index.php?
  account=tpc&action=add
http://myWebShop.com/index.php?
  account=tpc&action=show

What if the attacker tries:

http://myWebShop.com/index.php?
  account=admin&action=delete

Fix

No security through obscurity

Never rely on just the URL request for authentication.

E.g. Use cookies to control access.
Cross Site Scripting (XSS)

• Web browsers are dumb:
  – they will execute anything the server sends to them.
  
• Can an attacker force a website to send you something?

Cross-site scripting (XSS)

• An input validation vulnerability.
  
• Allows an attacker to inject client-side code (JavaScript) into web pages.
  
• This is then served by a vulnerable web application to other users.

Steal cookie example

• JavaScript can access cookies and make remote connections.
  
• A XSS attack can be used to steal the cookie of anyone who looks at a page, and send the cookie to an attacker.
  
• The attacker can then use this cookie to log in as the victim.

XSS attacks: phishing

• Attacker injects script that reproduces look-and-feel of “interesting” site (e.g., paypal, login page of the site itself)
  
• Fake page asks for user’s credentials or other sensitive information
  
• The data is sent to the attacker’s site

XSS attacks: redirect

• Attacker injects script that automatically redirects victims to attacker’s site

<script>
document.location = "http://evil.com";
</script>

Cross-site scripting (XSS)

1. Attacker injects malicious code into vulnerable web server
2. Victim visits vulnerable web server
3. Malicious code is served to victim by web server
4. Malicious code executes on the victims with web server’s privileges
### XSS attacks: run exploits

- The attacker injects a script that launches a number of exploits against the user’s browser or its plugins.
- If the exploits are successful, malware is installed on the victim’s machine without any user intervention.
- Often, the victim’s machine becomes part of a botnet.

### Summary

- This lecture basic web attacks:
  - SQL injection attacks
  - Stolen cookies
  - URL hacking
  - Cross site scripting attacks (XSS)

- Next Lecture:
  - More on XSS,
  - Cross-site request forgery (CSRF)
  - OWASP top 10