A3 – INSECURE REMOTE FILE INCLUDE

Insecure remote file include vulnerabilities are found in a great deal of code. Developers will often concatenate or directly use potentially hostile input with file or stream functions. On many platforms, frameworks allow the use of external object references, such as URLs or file system references. When the data is insufficiently checked, this can lead to arbitrary remote and hostile content being included, processed or invoked by the web server.

This allow attackers to perform:

- Remote code execution
- Remote root kit installation and complete system compromise
- On Windows, internal system compromise may be possible through the use of PHP's SMB file wrappers

This attack is particularly prevalent on PHP, and extreme care must be taken with any stream or file function to ensure that user supplied input does not influence file names.

ENVIRONMENTS AFFECTED

All web application frameworks that allow uploaded files to be executed are vulnerable to remote file include. By default, PHP 4.0.4 and later and 5.x are vulnerable to remote file inclusion. Other environments are susceptible if they allow file upload into web directories.

VULNERABILITY

A common vulnerable construct is:

```
include $_REQUEST['filename'];
```

Not only does this allow evaluation of remote hostile scripts, it can be used to access local file servers (if PHP is hosted upon Windows) due to SMB support in PHP's file system wrappers.

Other methods of attack include:

- Hostile data being uploaded to session files, log data, and via image uploads (typical of forum software)
- Using compression or audio streams, such as zlib:// or ogg:// which do not inspect the internal PHP URL flag and thus allow access to remote resources even if allow_url_fopen or allow_url_include is disabled
- Using PHP wrappers, such as php://input and others to take input from the request POST data rather than a file
- Using PHP's data: wrapper, such as `data:;base64,PD9waHAgcGhwaW5mbygpOz8+`

As this list is extensive (and periodically changes), it is vital to use a properly designed security architecture and robust design when dealing with user supplied inputs influencing the choice of server side filenames and access.

Although PHP examples have been given, this attack is also applicable in different ways to .NET and J2EE. Applications written in those frameworks need pay particular attention to code access security mechanisms to ensure that filenames supplied by or influenced by the user do not allow security controls to be obviated.

VERIFYING SECURITY

Automated approaches: Vulnerability scanning tools will have difficulty identifying the parameters that are used in a file include or the syntax for making them work. Static analysis tools can search for the use of dangerous APIs, but cannot verify that appropriate validation or encoding might be in place to protect against the vulnerability.

Manual approaches: A code review can search for code that might allow a file to be included in the application, but there are many possible mistakes to recognize. Testing can also detect these vulnerabilities, but identifying the particular parameters and the right syntax can be difficult.

PROTECTION

Preventing remote file include flaws takes some careful planning at the architectural and design phases, through to thorough testing. In general, a well-written application will not use user-supplied input in any filename for any server-based resource (such as images, XML and XSL transform documents, or script inclusions), and will have firewall rules in place preventing new outbound connections to the Internet or internally back to any other server. However, many legacy applications will continue to have a need to take user supplied input.

Among the most important considerations are:

- Consider a variable naming scheme to assist with taint checking:

```
$hostile = &$_POST; // refer to POST variables, not $_REQUEST

$safe['filename']= validate_file_name($hostile['unsafe_filename']); // make it safe
```

Therefore any operation based upon hostile input is immediately obvious:

☒ `require_once($_POST['unsafe_filename'] . 'inc.php');`

☑ `require_once($safe['filename'] . 'inc.php');`

- Strongly validate user input using "accept known good" as a strategy
- Hide server-side filenames from the user. For example, instead of including $language . ".lang.php", use an array index like this:

```
<select name="language"><option value="1">Français</option></select>

…

$language = intval($_POST['language']);

if ($language > 0) {

    require_once($lang[$language]); // lang is array of strings eg "fr.lang.php"

}
```

- Disable allow_url_fopen and allow_url_include in php.ini and consider building PHP locally to not include this functionality
- Add firewall rules to prevent web servers making new connections to external web sites and internal systems. For high value systems, isolate the web server in its own VLAN or private subnet.
- Ensure that file and streams functions (stream_*) are carefully vetted. Ensure that the user input is not supplied any function which takes a filename argument, including:

```
include() include_once() require() require_once() fopen() imagecreatefromXXX() file()
file_get_contents() copy() delete() unlink() upload_tmp_dir() $_FILES move_uploaded_file()
```

- Be extremely cautious if data is passed to system() eval() passthru() or ` (the backtick operator)
- Check that any files taken from the user for legitimate purposes cannot be otherwise obviated, such as including user supplied data in the session object, avatars and images, PDF reports, temporary files, and so on.

SAMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5220
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5205
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5188

REFERENCES

- OWASP Guide, http://www.owasp.org/index.php/File_System#Includes_and_Remote_files
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP PHP Top 5, http://www.owasp.org/index.php/PHP_Top_5#P1:_Remote_Code_Execution
- Stefan Esser, http://blog.php-security.org/archives/45-PHP-5.2.0-and-allow_url_include.html