SektionEins

# PHP Web Security

## Vulnerabilities & Concepts

### Vulnerability Types

**Cross Site Scripting (XSS)**
This vulnerability allows data to be injected into webpages. This data is then interpreted as code and executed by the viewer's web browser, which can effectively be seen as remote controlling a victim's browser.

**Cross Site Request Forgery (CSRF)**
CSRF refers to a type of exploits where the victim's browser is being tricked into triggering an authenticated action inside a vulnerable web application. The target website can be affected by CSRF regardless of being susceptible to XSS. How dangerous CSRF can be really depends on the kind of action triggered this way and its impact.

**SQL Injection**
SQL injection attacks lead to the manipulation of SQL queries. Vulnerable applications allow dynamically built SQL queries to contain unfiltered or improperly sanitised user input. If exploited successfully an attacker can gain access to all data in the database as well as modify data, limited only by the access level of the database user.

**Insecure Session Handling**
This category covers problems enabling attackers to access or manipulate a session token in order to control or take over a session.

**Session Fixation**
Session Fixation allows an attacker to control the session of a user. This is done by injecting a known token to be used as a valid session token.

**Information Disclosure**
As the name suggests, security related information is being divulged by the target system, which may simplify an attack. Such information can be found in various places, e.g. code comments, directory listings, error messages or even in search results of your favourite search engine.

**Header Injection**
This vulnerability allows HTTP headers to be injected into an HTTP response.

**File Inclusion**
The inclusion of local or remote files into a web application is a serious security vulnerability, which may lead to arbitrary code execution on the server.

**Insecure Configuration**
Misconfiguration of server or application software may facilitate or simplify attacks.

**Weak randomness**
This problem refers to predictable random number generation; e.g. badly chosen random seeds or algorithms using insufficient entropy are known to generate weak random numbers.

### Concepts

**Secure Input Handling**
Input filters and validators can be used to scan user input for specific patterns known to trigger unwanted side effects in web applications. User input can contain fragments of JavaScript, SQL, PHP or other code which - if unfiltered - could then lead to code execution within the context of the web application.

**Sanitising**
Sanitising functions can be used to "repair" user input, according to the application's restrictions (e.g. specific datatypes, maximum length) instead of rejecting potentially dangerous input entirely. In general, the use of sanitising functions is not encouraged, because certain kinds and combinations of sanitising filters may have security implications of their own. In addition, the automatic corrections could render the input syntactically or semantically incorrect.

**Escaping**
There are several different kinds of escaping:
- The backslash prefix "\" defines a meta character within strings. For Example: \t is a tab space, \n is a newline character, ... This can be of particular interest for functions where the newline character has a special purpose, e.g. header(). Within regular expressions the backslash is used to escape special characters, such as \. or *, which is relevant for all functions handling regular expressions.
- HTML encoding translates characters normally interpreted by the web browser as HTML into their encoded equivalents - e.g. < is < or < or < and > is > or > or >. HTML encoding should be used for output handling, where user input should be reflected in HTML without injecting code. (See also: htmlentities())
- URL encoding makes sure, that every character not allowed within URLs, according to RFC 1738, is properly encoded. E.g. space converts to + or %20 and < is %3C. This escaping is relevant for functions handling URLs, such as urlencode() and urldecode().

**White-/Blacklisting**
There are two different approaches to filtering input data - whitelisting and blacklisting. Blacklisting checks input data against a list of "bad patterns". This way, unwanted input can be discarded and all other content can be processed further. On the other hand, whitelisting checks input data against a list of known "good patterns". All unmatched input can be discarded and only input recognised as valid is accepted.

In the real world whitelisting turned out to be far more resistant to security vulnerabilities than blacklisting, since it is usually a lot easier to specify the narrow set of valid patterns for the whitelist than to exclude every invalid input with a blacklist. In particular, whitelisting should be used for input directly controlling the program flow, e.g. for include statements or eval().

## Security Related PHP Functions

### Validation and Sanitising Functions

**PHP-Core-Functions**
The PHP core provides a few functions suitable for sanitising:

- **is_numeric()**
  Checks a variable for numeric content.
- **is_array()**
  Checks if a variable is an array.
- **strlen()**
  Returns a string's length.
- **strip_tags()**
  Removes HTML and PHP tags.
  Warnings: As long as certain HTML tags remain, JavaScript can be injected along with tag attributes.

**CType Extension**
By default, PHP comes with activated CType extension. Each of the following functions checks if all characters of a string fall under the described group of characters:

- **ctype_alnum()**
  alphanumeric characters - A-Z, a-z, 0-9
- **ctype_alpha()**
  alphabetic characters - A-Z, a-z
- **ctype_cntrl()**
  control characters - e.g. tab, line feed
- **ctype_digit()**
  numerical characters - 0-9
- **ctype_graph()**
  characters creating visible output
  e.g. no whitespace
- **ctype_lower()**
  lowercase letters - a-z
- **ctype_print()**
  printable characters
- **ctype_punct()**
  punctuation characters - printable characters, but not digits, letters or whitespace,
  e.g. .!?:;*&$
- **ctype_space()**
  whitespace characters - e.g. newline, tab
- **ctype_upper()**
  uppercase characters - A-Z
- **ctype_xdigit()**
  hexadecimal digits - 0-9, a-f, A-F

```
<?php
if (!ctype_print($_GET['var'])) {
  die("User input contains ".
      "non-printable characters");
}
?>
```

**Filter Extension - ext/filter**
Starting with PHP 5.2.0 the filter extension has provided a simple API for input validation and input filtering.

- **filter_input()**
  Retrieves the value of any GET, POST, COOKIE, ENV or SERVER variable and applies the specified filter.

```
<?php
$url = filter_input(INPUT_GET,
  'url', FILTER_URL);
?>
```

- **filter_var()**
  Filters a variable with the specified filter.

```
<?php
$url = filter_var($var, FILTER_URL);
?>
```

### Escaping and Encoding Functions

- **htmlspecialchars()**
  Escapes the characters & < and > as HTML entities to protect the application against XSS. The correct character set and the mode ENT_QUOTES should be used.

```
<?php
echo "Hello " . htmlspecialchars(
  $_GET['name'], 'utf-8',
  ENT_QUOTES);
?>
```

- **htmlentities()**
  Applies HTML entity encoding to all applicable characters to protect the application against XSS. The correct character set and the mode ENT_QUOTES should be used.

```
<?php
echo "Hello " . htmlentities(
  $_GET['name'], 'utf-8',
  ENT_QUOTES);
?>
```

- **urlencode()**
  Applies URL encoding as seen in the query part of a URL.

```
<?php
$url = "http://www.example.com/" .
  "index.php?param=" .
  urlencode($_GET['pa']);
?>
```

- **addslashes()**
  Applies a simple backslash escaping. The input string is assumed to be single-byte encoded. addslashes() should not be used to protect against SQL injections, since most database systems operate with multi-byte encoded strings, such as UTF-8.

- **addcslashes()**
  Applies backslash escaping. This can be used to prepare strings for use in a JavaScript string context. However, protection against HTML tag injection is not possible with this function.

- **mysql_real_escape_string()**
  Escapes a string for use with mysql_query(). The character set of the current MySQL connection is taken into account, so it is safe to operate on multi-byte encoded strings. Applications implementing string escaping as protection against SQL injection attacks should use this function.

```
<?php
$sql = "SELECT * FROM user WHERE" .
  " login='" .
  mysql_real_escape_string(
    $_GET['login'], $db) .
  "'";
?>
```

- **preg_quote()**
  Should be used to escape user input to be inserted into regular expressions. This way the regular expression is safeguarded from semantic manipulations.

```
<?php
$repl = preg_replace('/^' .
  preg_quote($_GET['part'], '/').
  '-[0-9](1,4)/', '', $str);
?>
```

- **escapeshellarg()**
  Escapes a single argument of a shell command. In order to prevent shell code injection, single quotes in user input is being escaped and the whole string enclosed in single quotes.

```
<?php
system('resize /tmp/image.jpg' .
  escapeshellarg($_GET['w']).' '.
  escapeshellarg($_GET['h']));
?>
```

- **escapeshellcmd()**
  Escapes all meta characters of a shell command in a way that no additional shell commands can be injected. If necessary, arguments should be enclosed in quotes.

```
<?php
system(escapeshellcmd(
  'resize /tmp/image.jpg "' .
  $_GET['w'] . 'x' .
  $_GET['h']) . '"'));
?>
```

### List of Filters

**Validation Filters**

- **FILTER_VALIDATE_INT**
  Checks whether the input is an integer numeric value.
- **FILTER_VALIDATE_BOOLEAN**
  Checks whether the input is a boolean value.
- **FILTER_VALIDATE_FLOAT**
  Checks whether the input is a floating point number.
- **FILTER_VALIDATE_REGEXP**
  Checks the input against a regular expression.
- **FILTER_VALIDATE_URL**
  Checks whether the input is a URL.
- **FILTER_VALIDATE_EMAIL**
  Checks whether the input is a valid email address.
- **FILTER_VALIDATE_IP**
  Checks whether the input is a valid IPv4 or IPv6.

**Sanitising Filters**

- **FILTER_SANITIZE_STRING / FILTER_SANITIZE_STRIPPED**
  Strips and HTML-encodes characters according to flags and applies strip_tags().
- **FILTER_SANITIZE_ENCODED**
  Applies URL encoding.
- **FILTER_SANITIZE_SPECIAL_CHARS**
  Encodes " < > & \0 and optionally all characters < chr(127) into numeric HTML entities.
- **FILTER_SANITIZE_EMAIL**
  Removes all characters not commonly used in an email address.
- **FILTER_SANITIZE_URL**
  Removes all characters not allowed in URLs.
- **FILTER_SANITIZE_NUMBER_INT**
  Removes all characters except digits and + -.
- **FILTER_SANITIZE_NUMBER_FLOAT**
  Removes all characters not allowed in floating point numbers.
- **FILTER_SANITIZE_MAGIC_QUOTES**
  Applies addslashes().

**Other Filters**

- **FILTER_UNSAFE_RAW**
  Is a dummy filter.
- **FILTER_CALLBACK**
  Calls a userspace callback function defining the filter.

## Hardening the PHP Configuration

### Configuration

This chapter provides an overview of security related PHP configuration options (PHP version 5.3).

**# Information Disclosure**

**display_errors:**
- Defines if error messages should be displayed in the web browser.
- Default: 1
- Recommendation: Off

**display_startup_errors:**
- Defines if errors occurring during request initialisation should be displayed or logged.
- Default: 0
- Recommendation: Off unless for debugging.

**log_errors:**
- Defines if errors should be logged.
- Default: 0
- Recommendation: 1, in combination with error_log defining a path outside the web root.

**expose_php:**
- Defines if PHP should add version information to the response's HTTP header.
- Default: On
- Recommendation: Off, as it is advisable to divulge as little information as possible to potential attackers.

**# Resource Limits**

**max_execution_time:**
- Sets the maximum time in seconds before the PHP script is being terminated. Time is measured in CPU time, so delays e.g. for database queries are omitted.
- Default: 30
- Recommendation: This option should be set to a value high enough to recognise only unusual execution times.

**max_input_time:**
- Sets the maximum time in seconds a script is allowed process input, e.g. POST, GET, file uploads.
- Default: -1 (no restriction)
- Recommendation: It may be useful to restrict the execution time, but it should be decided on a case by case basis for each application.

**max_input_nesting_level:**
- Sets the maximum nesting level for input variables ($_REQUEST, $_POST, $_GET, ...). This prevents the interpreter from crashing while handling too deeply nested variables.
- Default: 64
- Recommendation: The default value should be sufficient for most use cases.

**memory_limit:**
- Sets the maximum amount of memory in bytes a PHP script is allowed to allocate.
- Default: 128M
- Recommendation: Before changing this value the script's maximum memory requirement should be tested empirically and increased by 20%.

**post_max_size:**
- Sets the maximum size of POST data allowed.
- Default: 8M
- Recommendation: This value should match the size actually required. File uploads have to be covered by this setting as well.

**upload_max_filesize:**
- Sets the maximum file upload size allowed.
- Default: 2M
- Recommendation: This value should match the file size actually required.

**# Features**

**allow_url_fopen:**
- Activates URL-aware fopen() wrappers, allowing URLs to be handled like files.
- Default: On (can only be changed via php.ini)
- Recommendation: Deactivate, if possible.

**allow_url_include:**
- Allows URL wrappers to be used with include(), include_once(), require() and require_once().
- Default: Off
- Recommendation: Should remain deactivated for security reasons.

**magic_quotes_gpc:**
- Activates magic_quotes for GET/POST/COOKIE (GPC). If activated, all ' (single quotes), " (double quotes), \ (backslash) and NULLs are being escaped by \ transparently. This option will not be available in PHP 6.
- Default: 1
- Recommendation: This option should be deactivated. Instead, user input should be escaped properly and handled in a secure way when building database queries.

**magic_quotes_runtime:**
- Activates backslash escaping for all user input.
- Default: 0
- Recommendation: This option should remain deactivated. Instead, user input should be escaped properly and handled in a secure way when building database queries.

**magic_quotes_sybase:**
- Switches to escaping with a single quote instead of a backslash for magic_quotes_gpc and magic_quotes_runtime.
- Default: 0
- Recommendation: This option should remain deactivated. Instead, user input should be escaped properly and handled in a secure way when building database queries.

**enable_dl:**
- Allows PHP extensions to be loaded dynamically
- Default: 1
- Recommendation: Deactivate this option to prevent arbitrary code to be loaded during runtime.

**disable_functions:**
- Disables specified functions.
- Default: ""
- Recommendation: Potentially dangerous and unused functions should be deactivated, e.g. system().

See also: List of potentially dangerous PHP functions

**disable_classes:**
- Disables specified classes.
- Default: ""
- Recommendation: Potentially dangerous and unused classes should be deactivated

See also: List of potentially dangerous PHP classes

**request_order:**
- Defines the order in which GET, POST and COOKIE is parsed into REQUEST from left to right. Later values supersede existing ones.
- Default: ""
- Recommendation: It is recommended to use GP to register GET and POST with REQUEST.

**variables_order:**
- Defines the order in which the variables ENV, GET, POST, COOKIE and SERVER are parsed. This parsing is carried out from left to right and new values overwrite old ones. If no value is set, no superglobals will be generated. If e.g. "SP" is used, $_SERVER and $_POST will be generated.
- Default: EGPCS
- Recommendation: Changing this setting is usually not necessary; however, the ENV variables are rarely used.

**auto_globals_jit:**
- Controls if the variables REQUEST, SERVER and ENV are only generated when they are used for the first time (JIT = Just In Time) or are generated at the start of the execution of the script.
- Default: On
- Recommendation: Unless access to these variables is done through variable variables this option can remain activated.

**register_globals:**
- Defines that EGPCS variables (ENV, GET, POST, COOKIE, SERVER) should additionally be registered as global variables.
- Default: Off
- Recommendation: For safety reasons this option should never be activated.

**file_uploads:**
- Controls if HTTP file uploads are permitted or not.
- Default: On
- Recommendation: If an application does not require HTTP file uploads, this setting should be deactivated.

**filter.default:**
- Selects the standard filter used for filtering $_GET, $_POST, $_COOKIE and $_REQUEST variables.
- Default: "unsafe_raw"
- Recommendation: Change only if the application is specifically designed to handle filtered values.

**# Output**

**default_mimetype:**
- Defines the default mimetype setting.
- Default: "text/html"
- Recommendation: Default value should not be changed.

**default_charset:**
- Defines the default character set.
- Default: <empty>
- Recommendation: The default character set should be set according to the application.

**List of potentially dangerous PHP functions:**

- system - Program execution
- exec - Program execution
- shell_exec - Program execution
- proc_open - Program execution
- popen - Program execution
- passthru - Program execution
- stream_socket_server – Server socket creation
- dl – Load arbitrary PHP modules
- phpinfo – Information about the server
- mail – Send email
- posix_* - Posix functions
- apache_child_terminate – Terminate apache1 process
- apache_setenv – Set apache environment variable
- virtual - Bypass safe mode

**Liste von potentiell gefährlichen PHP-Klassen:**

- SPLFileObject – Write access to files

**# safe_mode & open_basedir**

**safe_mode:**
- Activates PHP's safe mode. This will be dropped with PHP 6.
- Default: Off
- Recommendation: As required.

**safe_mode_gid:**
- Defines if safe_mode uses UID (false, 0) or GID (true, 1) for access checks.
- Default: Off
- Recommendation: As required.

**safe_mode_include_dir:**
- Defines a list of directories that are exempt from the safe_mode restrictions for the include or require statement. This allows to include a central copy of libraries. Multiple paths are separated by colons (or semicolons in Windows).
- Default: <empty>
- Recommendation: As required.

**safe_mode_exec_dir:**
- Defines a directory that is exempt from safe_mode restrictions. Programs within this directory are allowed to be executed by system() and similar functions.
- Default: <empty>
- Recommendation: As required.

**safe_mode_allowed_env_vars:**
- Defines a list of permitted prefixes for environment variables that can be used when safe_mode is activated. If this value is empty, all variables can be used.
- Default: "PHP_"
- Recommendation: As required.

**safe_mode_protected_env_vars:**
- Defines a list of environment variables, which cannot be set by the user with the putenv() function, even if they are permitted by the safe_mode_allowed_env_vars directive.
- Default: "LD_LIBRARY_PATH"
- Recommendation: As required.

**open_basedir:**
- Defines in which directories files may be opened by a script using fopen() and similar functions. The value is used as a prefix. /dir/lib would permit access to /dir/lib & /dir/library. In order to permit only one path, it must end with a slash. This option is not affected by the setting of safe_mode.
- Default: <empty> (opening of all files is permitted)
- Recommendation: This should be restricted to the document root directory.

**# Session**

**session.save_path:**
- Defines an argument that is passed to the session's storage handler. This is usually the path where the session data is stored.
- Default: <empty>
- Recommendation: Each application should store its session data in a separate location.

**session.cookie_httponly:**
- Controls if cookies are tagged with httpOnly which makes them accessible by HTTP only and not by the JavaScript. httpOnly cookies are supported by all major browser vendors and therefore can be instrumental in minimising the danger of session hijacking.
- Default: Off
- Recommendation: Should be activated.

**session.cookie_secure:**
- Controls if cookies are tagged as secure and should therefore be sent over SSL encrypted connections only.
- Default: Off
- Recommendation: Must be activated for SSL websites.

**session.cookie_lifetime:**
- Defines the maximum lifetime of a cookie in seconds.
- Default: 0 (until the browser is closed)
- Recommendation: Should be adjusted to the desired session lifetime.

## Secure programming

### Securing HTML Output

In order to prevent the execution of JavaScript code originating from user input, it is mandatory to perform a suitable string sanitisation on all dynamic data before any HTML output. The use of htmlentities() is considered sufficient within normal HTML context.

However, if data can be injected into tags or tag attributes, JavaScript can be executed by means of event handlers such as onClick or by modifying style attributes. For these cases it is recommended to apply a whitelist filter allowing only predefined tag attributes or style sheets to be inserted.

URLs within tag attributes must be checked as well. Some URI schemes, such as data: vbscript: and javascript: can be used to execute code. Therefore only specific schemes should be allowed. Of course, it is always a good idea to encode the query part of a URL appropriately as well.

Finally, data put directly into JavaScript code must be prevented from breaking out of its JavaScript context. JavaScript strings are known to be particularly prone to incorrect escaping.

### Regular Expressions

Every user input placed inside regular expressions must be prepared using preg_quote(). Otherwise an injection into the expression's logic can easily lead to incorrect application behaviour, buffer overflows, denial of service or application crashes.

```
<?php
$res = preg_match('/(foo|' .
  preg_quote($_GET['i'], '/') .
  ')/i', $input);
?>
```

### HTTP Header Output

HTTP headers can be set using the header() function. User input should always be checked before being passed to header(), otherwise a number of security issues become relevant.

Newline characters should never be used with header() in order to prevent HTTP header injections. Injected headers can be used for XSS and HTTP response splitting attacks, too. In general, user input should be handled in a contet-sensitive manner.

Dynamic content within parameters to Location or Set-Cookie headers should be escaped by urlencode().

```
<?php
if (strpbrk($_GET['x'], "\r\n"))
  die('line break in x');
header("Location: " .
  "http://www.example.com/?p=".
    urlencode($_GET['x']));
header("Set-Cookie: mycookie=".
    urlencode($_GET['x']) .
    "; path=/;");
?>
```

For other HTTP header parameters, unintended context changes must be prevented as well; e.g. a semicolon separates several parameters within Content-Type.

```
<?php
  if (strpbrk($_GET['type'],
      ";/\r\n"))
    die('invalid characters');
  header("Content-Type: text/" .
    $_GET['type'] .
    "; charset=utf-8;");
?>
```

Applications should not allow arbitrary HTTP Location redirects, since these can be used for phishing attacks. In addition, open redirects can have a negative impact on the cross domain policy infrastructure of Adobe's Flash Player.

### Executing Shell Commands

Shell commands can be executed by various functions in PHP, e.g. system(), passthru(), exec(), shell_exec(), proc_open(), popen(). Commands built from user input must be properly escaped in order to prevent shell command injections. PHP provides two functions for this purpose – escapeshellarg() and escapeshellcmd().

```
<?php
$param = 'helloworld';
  '/usr/bin/id';
system('echo ' .
  escapeshellarg($param));
?>
```

### Secure File Handling

- Detect and replace NULL bytes:

```
<?php
if (strpos($_GET["f"], "\0") ===
    true) {
  $file = str_replace("\0", "",
    $_GET["f"]);
}
?>
```

- Prevent remote file inclusion (path prefix) and directory traversal (basename):

```
<?php
$file = "./".basename($_GET["f"]).
  ".php";
?>
```

- Include only whitelisted files:

```
<?php
if (in_array($_GET['action'],
    array('index', 'logout'))) {
  include './'.$_GET['action'] .
    '.php';
} else
  die('action not permitted');
?>
```

### Securing SQL Queries

SQL query strings should always be protected against SQL injections. There are two possible strategies. On the one hand, each dynamic part of the query string can be escaped appropriately.

On the other hand, the whole query string can be set up as a prepared statement, where dynamic parts are represented by placeholders to be replaced right before the actual query.

While both methods have pros and cons, prepared statements are most suitable for preventing SQL injections. Dynamic content and the prepared static part of a query are conceptually separated and therefore less prone to programming errors.

In PHP prepared statements are supported by several different database APIs. The following example shows the execution of a prepared statement using the common PDO API. Its syntax remains unchanged for different database management systems (DBMS), even with DBMS not supporting prepared statements, in which case it can be emulated by PDO.

The complete SQL query looks like this:

```
<?php
$sth = $dbh->prepare(
  'SELECT id, gid FROM _users ' .
  'WHERE login = ?' .
  'AND passhash = ?');
$sth->execute(
  array('admin', md5('myPass')));
$res = $sth->fetchAll();
?>
```

## Server Protection with Suhosin

Suhosin is a protective system for PHP installations which protects servers from known and also unknown weaknesses in PHP applications and the PHP core. It consists of two parts that can be used individually or together. The first part is the Suhosin patch which adds resistance against buffer overflow and other memory corruption exploits to the PHP core. The second part is the Suhosin extension which, being a loadable PHP module, implements various security defences for PHP scripts. The level of protection is set within the PHP configuration.

**# Logging Configuration**

**suhosin.log.syslog:**
- Defines which classes of security alerts should be logged to syslog. The class S_MEMORY is always logged into syslog because damage to the heap could result in all other logging-mechanisms ceasing to function. For an overview of available settings, please refer to the list of logging constants.
- Default: S_ALL & ~S_SQL
- Recommendation: Logging into syslog should be used primarily.

**List of Logging Constants**

| | |
|---|---|
| S_MEMORY | Value: 1<br>Logs memory access violations. |
| S_MISC | Value: 2<br>Logs all errors that do not fall into another category. |
| S_VARS | Value: 4<br>Logs all errors that were caused due to request variables. |
| S_FILES | Value: 8<br>Logs all errors that were caused due to uploaded files. |
| S_INCLUDE | Value: 16<br>Logs attempted injection attacks against include statements. |
| S_SQL | Value: 32<br>Logs attempted injection attacks against SQL queries. |
| S_EXECUTOR | Value: 64<br>Logs problems that are caused by the executor protection. |
| S_MAIL | Value: 128<br>Logs warnings from the mail() function protection. |
| S_SESSION | Value: 256<br>Logs warnings of the transparent session encryption. |
| S_ALL | Value: 511<br>Combines all classes. |

**suhosin.log.syslog.facility:**
- Sets the syslog facility that should be used for logging to syslog when warnings are issued. Available facilities depend on the system used and can be checked in the documentation of the system's syslog-daemon.
- Default: LOG_USER
- Recommendation: Adjust as suitable.

**suhosin.log.syslog.priority:**
- Sets the priority level used when logging to syslog. Supported priority levels depend on the system used and can be checked in the documentation of the system's syslog daemon.
- Default: LOG_ALERT
- Recommendation: If required.

**suhosin.log.sapi:**
- Defines which classes of security alerts are logged into the SAPI errorlog. Please refer to the list of logger constants for an overview of available settings.
- Default: S_ALL & ~S_SQL
- Recommendation: If required.

**suhosin.log.script:**
- Defines which classes of security alerts should be logged by a shell script. Errors of class S_MEMORY cannot be logged by a shell script because the current process might not be in a stable state after a memory access violation. Please refer to the list of logging constants for an overview of available settings.
- Default: 0
- Recommendation: If required.

**suhosin.log.phpscript:**
- Defines which classes of security alerts should be logged by a PHP script. Errors of the class S_MEMORY cannot be logged by a PHP script because the current process might not be in a stable state after a memory access violation. Please refer to the list of logging constants for an overview of available settings.
- Default: 0
- Recommendation: Should only be used in exceptional cases for classes of errors that could occur during script execution.

**suhosin.log.script.name:**
- Sets the full path name of a logging shell script. This script is called with two parameters; the first represents the error class as a string of characters and the second parameter contains the alert message.
- Default: Off
- Recommendation: If required.

**suhosin.log.phpscript.name:**
- Sets the full path name of a logging PHP script. When this script is executed the two variables SUHOSIN_ERRORCLASS and SUHOSIN_ERROR are created in the current variable scope. The logging-script can use them to extract the error type and message.
- Default: <empty>
- Recommendation: If required.

**suhosin.log.use-x-forwarded-for:**
- Defines if Suhosin should extract the IP address of an attacker from the X-Forwarded-For HTTP header when constructing error messages. This is required if the server is behind a reverse proxy.
- Default: off
- Recommendation: If required.

**# Executor Options**

**suhosin.executor.max_depth:**
- Defines the maximum stack depth that is permitted during the execution of PHP scripts. If the stack depth is exceeded, the script will be terminated.
- Default: 0 (off)
- Recommendation: This setting should be set to a value that does not interfere with the application, but at the same time does not allow to crash the PHP interpreter, e.g. 500.

**suhosin.executor.include.max_traversal:**
- Defines how often "../" may occur in filenames for include-statements before it is considered to be an attack. A value of zero deactivates the feature.
- Default: 0
- Recommendation: Most PHP-applications do not require a value greater than 5.

**suhosin.executor.include.whitelist:**
- Defines a list of URI schemes that are permitted to be used in include statements. If no whitelist is defined, then a potentially defined blacklist is evaluated. This option does only work if URL inclusions are not completely disabled in the PHP configuration.
- Default: <empty>
- Recommendation: URL inclusions should be deactivated unless absolutely necessary.

**suhosin.executor.include.blacklist:**
- Defines a list of URI schemes that must not be used in include statements. Other URI schemes remain permitted. In case a whitelist is defined, the blacklist will be ignored. This option works only if URL inclusions are not completely disabled in the PHP configuration.
- Default: <empty>
- Recommendation: URL inclusions should be deactivated unless absolutely necessary.

**suhosin.executor.func.whitelist:**
- Defines a list of PHP functions that may be used by PHP scripts. The use of any other function is prohibited and will lead to a termination of the script. If the whitelist is empty, a potentially defined black list will be evaluated.
- Default: <empty>
- Recommendation: Should be used if a PHP script requires only very few functions.

**suhosin.executor.func.blacklist:**
- Defines a list of PHP functions that must not be used by PHP scripts. Executing these functions will lead to a termination of the script.
- Default: <empty>
- Recommendation: Should at least contain all functions that are deactivated by disable_functions.

**suhosin.executor.eval.whitelist:**
- Defines a list of PHP functions that may be used by runtime evaluated PHP code. All other functions are prohibited and their use will lead to a termination of the script. If the white list is empty, a potentially defined blacklist will be evaluated.
- Default: <empty>
- Recommendation: Should be used if evaluated PHP code requires only very few functions.

**suhosin.executor.eval.blacklist:**
- Defines a list of PHP functions that must not be used by runtime evaluated PHP code. Executing these functions will lead to a termination of the script.
- Default: <empty>
- Recommendation: Should be used to disallow dangerous functions that are not absolutely necessary.

**suhosin.executor.disable_eval:**
- Controls if the eval() statement is enabled or not. If disabled it is no longer possible to dynamically evaluate PHP code.
- Default: Off
- Recommendation: Activating this switch will break a lot of scripts because many rely on the availability of eval().

**suhosin.executor.disable_emodifier:**
- When set the /e modifier cannot be used within preg_replace(). Its use is deprecated because dynamically evaluating PHP code is error-prone which can result in arbitrary PHP code execution. The same functionality as the /e modifier can be achieved by using preg_replace_callback().
- Default: Off
- Recommendation: In addition to activating this switch, it is recommended to check used PHP applications for the usage of the /e modifier and rewrite them to use preg_replace_callback().

**# Other Options**

**suhosin.simulation:**
- Activates or deactivates the simulation mode of Suhosin. If activated violations will only be logged and not blocked.
- Default: Off
- Recommendation: During initial deployment of Suhosin, this flag should be switched on to ensure that the application continues to work under the new configuration.

**suhosin.multiheader:**
- Allows to disable the protection against HTTP Response splitting.
- Default: Off
- Recommendation: This should only be allowed if the application requires this hack and cannot be changed.

**suhosin.mail.protect:**
- Defines how strict the mail() protection should operate. A value of 0 deactivates the protection. A value of 1 prohibits line breaks in headers and double line breaks in extended headers. A value of 2 prohibits all addressing through extended headers.
- Default: 0
- Recommendation: Should at least be set to 1 to activate the basic protection.

**suhosin.memory_limit:**
- If safe_mode is not set any script can arbitrarily change the memory_limit. Suhosin allows to configure a hard memory_limit which cannot be bypassed by the script.
- Default: 0
- Recommendation: If the desired memory_limit is not supposed to be altered, it must be set here.

**# Configuration of Transparent Encryption**

**suhosin.session.encrypt:**
- Activates transparent encryption of session data.
- Default: On
- Recommendation: Transparent encryption of session data should always be activated.

**suhosin.session.cryptkey:**
- Defines an application specific key that is merged into the encryption key for the transparent session encryption.
- Default: <empty>
- Recommendation: Each application should set its own secret key with ini_set().

**suhosin.session.cryptua:**
- Defines if the content of the User-Agent header is included in the encryption key for the session encryption.
- Default: Off
- Recommendation: This option is known to cause problems with IE.

**suhosin.session.cryptdocroot:**
- Defines that the current document root directory of the web server is merged into the encryption key for the session encryption.
- Default: On
- Recommendation: Should only be used if the visitor´s IP address is not expected to change during the session. Users of large proxy farms very often change their IP address.

**suhosin.session.cryptraddr:**
- Defines how many octets of REMOTE_ADDR (0..4) are included in the encryption key for the session encryption.
- Default: 0
- Recommendation: Should only be used if visitors of a site will not have changes of their IP addresses. Users of large proxy farms very often change their IP addresses during a session.

**suhosin.session.checkraddr:**
- Defines how many octets of REMOTE_ADDR (0..4) are checked after decryption. This option allows, for example, to have strict controls on administrative backends. The difference to the previous configuration is that it is possible to use the same session for both strict and non-strict areas.
- Default: 0
- Recommendation: This should always be used for the protection of administrative areas.

**suhosin.cookie.encrypt:**
- Activates transparent encryption of cookie data.
- Default: On
- Recommendation: The transparent encryption of cookie data should always be activated.

**suhosin.cookie.cryptkey:**
- Defines an application specific key that is merged into the encryption key for transparent cookie encryption.
- Default: <empty>
- Recommendation: Every application should set its own secret key with ini_set().

**suhosin.cookie.cryptua:**
- Defines if the content of the User-Agent header is included in the encryption key for the cookie encryption.
- Default: Off
- Recommendation: This option is known to cause problems with IE.

**suhosin.cookie.cryptdocroot:**
- Defines that the current document root directory of the web server is merged into the encryption key for cookie encryption.
- Default: On
- Recommendation: Should remain activated. Problems will arise if the session is started from different root directories.

**suhosin.cookie.cryptraddr:**
- Defines how many octets of REMOTE_ADDR (0..4) are included in the encryption key for the cookie encryption.
- Default: 0
- Recommendation: Should only be used if the visitor´s IP address is not expected to change during the session. Users of large proxy farms very often change their IP address.

**suhosin.cookie.checkraddr:**
- Defines how many octets of REMOTE_ADDR (0..4) are verified after decryption. This option allows, for example, to have strict controls on administrative backends. The difference to the previous configuration is that it is possible to use the same cookies for the strict and the non-strict area.
- Default: 0
- Recommendation: This should always be used for the protection of administrative areas.

**suhosin.cookie.cryptlist:**
- Specifies a list of cookies to be encrypted. No other cookies are encrypted.
- Default: <empty>
- Recommendation: Should be used if the majority of cookies are required to be readable on the client side (e.g. JavaScript).

**suhosin.cookie.plainlist:**
- Specifies a list of cookies that should not be encrypted. All other cookies are encrypted.
- Default: <empty>
- Recommendation: Should be used if the majority of cookies are not required to be readable on the client side (e.g. JavaScript).

**# Filter Configuration**

**suhosin.filter.action:**
- Defines Suhosin's reaction to variable filter violations. Different configuration options can be found in the Suhosin documentation.
- Default: <empty>
- Recommendation: It is recommended that only the registration of violating variables is blocked. Anything else could lead to DoS vulnerabilities of the application.

**suhosin.[source].max_array_depth:**
- Defines the maximum depth for arrays of this variable source. Supported variable sources are get/post/cookie and request. Setting a value for the request source overrides individual settings of the other three sources.
- Default: Depends on the source
- Recommendation: Default settings should be checked. The chosen settings may be too low for popular PHP applications such as phpMyAdmin.

**suhosin.[source].max_array_index_length:**
- Defines the maximum length of array indexes. Supported variable sources are get/post/cookie and request. Setting a value for the request source overrides individual settings of the other three sources.
- Default: Depends on the source
- Recommendation: It is usually not necessary to change the default setting.

**suhosin.[source].max_name_length:**
- Defines the maximum length of variable names. Supported variable sources are get/post/cookie and request. Setting a value for the request source overrides individual settings of the other three sources.
- Default: Depends on the source
- Recommendation: It is usually not necessary to change the default setting.

**suhosin.[source].max_totalname_length:**
- Defines the maximum length of the complete name of a variable including the indexes. Supported variable sources are get/post/cookie and request. Setting a value for the request three3 sources.
- Default: Depends on the source
- Recommendation: It is usually not necessary to change the default setting.

**suhosin.[source].max_value_length:**
- Defines the maximum length of variable values. Supported variable sources are get/post/cookie and request. Setting a value for the source request overrides individual settings of the other three sources.
- Default: Depends on the source
- Recommendation: Default settings should be checked, as they may be too low for popular PHP applications such as CMS systems.

**suhosin.[source].max_vars:**
- Defines the maximum number of variables that can be registered. Supported variable sources are get/post/cookie and request. Setting a value for the source request overrides individual settings of the other three sources.
- Default: Depends on the source
- Recommendation: Default settings should be checked, as they may be too low for some popular PHP applications.

**suhosin.[source].disallow_nul:**
- Defines if variable values may contain ASCII NUL bytes or not.
- Default: On
- Recommendation: Should be activated because ASCII NUL bytes are hardly ever used for anything other than attacks.

**suhosin.upload.max_uploads:**
- Defines how many files can be uploaded in one request at the same time.
- Default: 25
- Recommendation: The default setting usually does not need to be changed.

**suhosin.upload.verification_script:**
- Specifies a shell script that is used to verify uploaded files. The script is called with the temporary file name as parameter. This can be used e.g. for virus scans. The upload is only permitted if the script echoes "1", otherwise the file will be deleted.
- Default: <empty>
- Recommendation: This feature should be used to check if uploaded files contain viruses or malware.

**suhosin.session.max_id_length:**
- Specifies the maximum length of session identifiers. Longer session identifiers are ignored and replaced with new ones.
- Default: 128
- Recommendation: This setting should not be changed.

**Security Audits**
Security Audits for web applications assure the server and client side production readiness of applications. Result of such audits is a security report structured by OWASP guidelines that contain all risks found and recommendations to mitigate them. If the software has been written in-house, an additional workshop with your developers should be conducted to build security knowhow and awareness.

**Simple Audits**
A black box audit examines the web application from an attackers point of view. The test is accomplished manually with additional tool support, without any knowledge of the application's internals. A black box audit can significantly increase your application's security by decreasing the probability of successful hacking attacks by detecting and removing the "low hanging fruits".

**Extended Audit**
The extended audit adds source code inspection to the black box audit. The source code is tested manually supported by automated static code analysis tools; input flow and critical function calls are checked. A complete source code audit provides a good reliability and prevents even hard to find security issues.

**Consulting**
A network and application architecture audit helps assuring the intended level of security as early as during the design phase of the software. Security relevant ports and paths of the software are revealed and solutions are provided. That allows the development team to focus on their core competences while maintaining a state of the art security.

If you are interested in an audit or consulting by SektionEins, please contact us at

SektionEins GmbH
Eupener Strasse 150
50933 Cologne
Germany
Tel: +49 221 29282933

http://www.sektioneins.de/



Concept: Stefan Esser, fukami, Ben Fuhrmannek

As of: October 2009