web development

security

Съдържание

Intro

Intro

Security is a major concern for web applications.

Even major organizations such as the United Nations have been hacked using very simple security flaws.

Intro

There is no such thing as a completely secure system.

Our aim when securing an application is two-fold.

- First - to make it take as long as possible for an attacker to gain access.
- Next - to minimize the value of any information they can retrieve.

Configuration

Configuration

When configuring PHP make sure that you keep up to date with the releases and use the improvements they bring.

You should have a very strong reason if you are not using the most current stable release of PHP in favor of an older version.

Errors and Warnings

Errors and Warnings

You should configure PHP to hide warnings and errors while in production.

Errors and warnings can give a person a clue about the internal workings of your code such as directory names and what libraries you are using.
This sort of information can help them exploit vulnerabilities in your stack.

Errors and Warnings

php.ini file or at runtime with the error_reporting() function

Setting	Value
display_errors	Off
log_errors	On
error_reporting	E_ALL & ~E_DEPRECATED & ~E_STRICT

Session Security

Session Hijacking

HTTP is a stateless protocol and a web server can be expected to be serving multiple different visitors at the same time.

The server needs to be able to tell clients apart and does so by assigning each client a **session identifier**.

*The session identifier can be retrieved by calling **session_id()**. It is created after the **session_start()** function is called.*

Session Hijacking

When the client makes subsequent requests to the server, they provide the session identifier, which allows the server associate the request with a session.

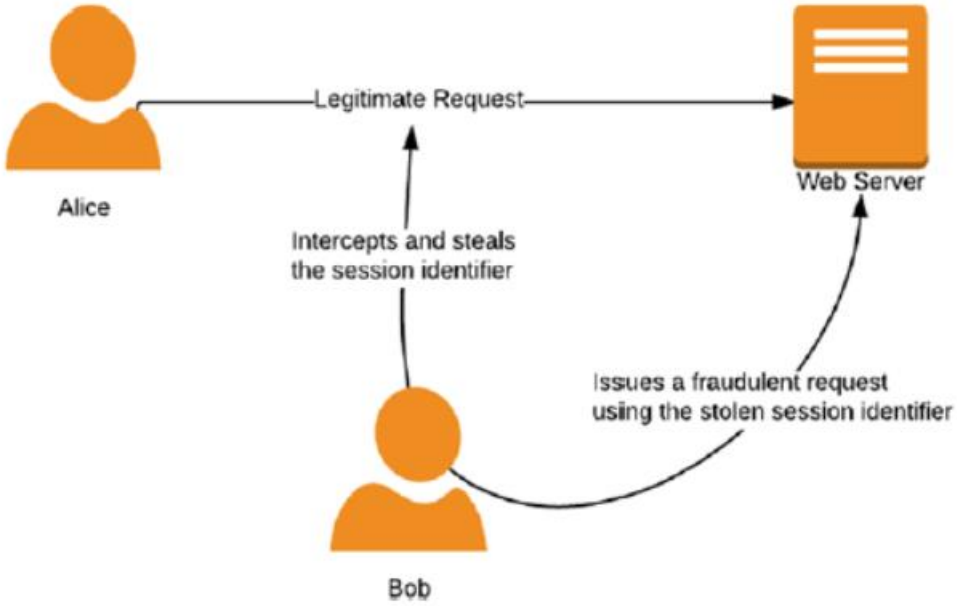
Clients can provide the session either with cookies or with a URL parameter.

Cookies are preferable but are not always available. If PHP cannot use a cookie, it will automatically and transparently use the URL.

Session Hijacking

If you are able to present somebody else's session identifier to the server, you can masquerade as that user or **hijack his session.**

Session Hijacking



Session Hijacking

Obtaining the session identifier of another user can be accomplished in several ways.

- If the session identifier follows a predictable pattern. *PHP uses a very random way to generate session identifiers, so you don't need to worry about this.*
- By inspecting network traffic between the client and the server, the attacker could read the session identifier.

You can set session.cookie_secure=On to make session cookies only available over HTTPS to mitigate this.

HTTPS will also encrypt the URL being requested, and so if the session identifier is being passed as a parameter in the request, it will be encrypted.

Session Hijacking

• Attacks made against the client, such as an XSS attack or Trojan program running on their computer, could also reveal the session identifier. ***This can be partially mitigated by setting the session.cookie_httponly directive on.***

Session Fixation

Session fixation exploits a weakness in the web application. Some applications do not generate a new session ID for a user when authenticating them. Instead they allow an *existing session ID to be used.*

The attack occurs when an opponent creates a session on the web server.
They know the session ID for this session. They then trick a user into using this session and authenticating themselves.
The attacker is then able to use the known session ID and *has the privileges of the authenticated user.*

Session Fixation

There are several ways to set the session ID and the actual method used will depend on how the application accepts the identifier.

The simplest way to do it would be to pass the session identifier in the URL, like this
http://example.org/index.php?PHPSESSID=1234.

Session Fixation

How to mitigate the risk of session fixation

- call the function *session_regenerate_id()* every time the privilege level changes, for example after logging in.
- set **session.use_strict_mode=On** in your config file. This setting will force PHP to only use session identifiers that it creates itself. It will reject a user-supplied session identifier. This will mitigate attempts to manipulate the cookie.
- *session.use_cookies=On* and *session.use_only_cookies=On* will prevent PHP from accepting the session identifier from the URL.

Improving Session Security

Use several layers of security.

Improving Session Security

None of these is particularly effective by itself but each can contribute toward improving your overall security.

Improving Session Security

In addition to the mitigation strategies, do the following:

• *Check that the IP address remains the same between calls*. This is not always feasible for mobile phones that move between towers and so change connections, so check your use-cases before you do this.

• *Use short session timeout* values to reduce the window for fixation.

• Provide a means for *users to log out that calls session_destroy()*.

None of these is particularly effective by itself but each can contribute toward improving your overall security.

Cross-Site Scripting
/XSS/

XSS

Cross-site scripting (XSS) attacks are attacks where malicious code is injected onto an otherwise benign site. Usually malicious browser-side code like JavaScript is placed onto the web site to be downloaded and run by clients.

The attack is effective because the client thinks that the code originated from the web site that it trusts. The code can access session identifiers, cookies, HTML storage data, and other information related to the site.

Example here and here and here

XSS

The only way for the attacker to run his malicious JavaScript in the victim's browser **is to inject it into one of the pages** that the victim downloads from the website. This can happen if the website **directly includes user input** in its pages, because the attacker can then insert a string that will be treated as code by the victim's browser.

XSS

The most important rule to follow is –

Never allow unescaped data to be output to the client.
Always filter data and strip out harmful tags before allowing it to be sent to the client.

XSS

Remember this mantra

“Filter input, escape output”

XSS

htmlspecialchars()
htmlentities()
strip_tags

filter_var($string, FILTER_SANITIZE_STRING)
/safest way to escape output before displaying it/

Cross-Site Request Forgeries /CSRF/

CSRF

CSRF attacks exploit the trust that a web site has in a client.

In these attacks, the opponent tricks the client into executing a command on a web site that trusts that client.

CSRF

Imagine that Alice is logged onto her bank web site that has a form that allows her to transfer money to another account.

Chuck knows the endpoint of that form and what input fields it has. He somehow manages to trick Alice’s web browser into sending a POST request to that form instructing the bank to transfer money into his account.

The bank trusts Alice’s web browser because it has a valid session and performs the request.

CSRF

To mitigate these requests, you should generate **a unique and very random token that you store in Alice's session**. When you output the form, you include this token so that when Alice submits the form, she also submits the token.

CSRF

Before you process the form, **you check** that the submitted token matches the token stored in her session.

Chuck has no way of knowing what token is in Alice's session and so won't be able to include it in his POST.

Your code will reject the request that he tricked Alice into making because it **doesn't have a valid token**.

CSRF

Actual banks often require a person to re-authenticate when performing a sensitive operation and will often require two-factor authentication as part of this process.

SQL Injection

SQL injection

SQL injection is the most common form of attack on the web, and one of the easiest to defend against.

SQL injection occurs when the attacker can insert malicious commands into a SQL statement for execution by the database.

Examples here

SQL injection

At its heart the problem with SQL injection comes from the fact that a SQL statement has a mix of data and syntax.

By allowing user-supplied data to be incorporated with function syntax, we create the possibility that malicious data can interfere with the syntax.

SQL injection

The most effective way to start to mitigate SQL injection in the PHP language is to exclusively use **prepared statements** to interact with your database.

Example here

SQL injection

A less effective way to mitigate SQL injection is to **escape special characters** before sending them to the database. This is more prone to error than using prepared statements.

If you are going to try escaping special characters, you must use the database specific function (e.g., **mysqli_real_escape_string()**) or PDO::quote() and not a generic function like addslashes().

http://php.net/manual/en/mysqli.real-escape-string.php

SQL injection

General Principles

You should also always connect to the database with a user who has the least amount of privileges that are required for the application to function.

SQL injection

General Principles

Never allow your web application to connect to the database as its root user.

If you host multiple databases on your server, use a different user for each database on your server and make sure that their passwords are unique. This will help prevent a SQL injection attack on one site from affecting the databases of other sites.

SQL injection

General Principles

Make sure that you're using an up-to-date version of MySQL and enforce the use of a character set in the client DSN. There is a very subtle way to use mismatching character sets in certain vulnerable encoding schemes to deploy a SQL injection;

see the second answer (not the accepted one) on this StackOverflow article for an exposition.

Remote Code Injection

Remote Code Injection

Remote code injection is an attack where an opponent can get the server to include and execute their code.

Email Injection

Email Injection

It is possible for users to supply hexadecimal control characters that allow them to change the message body or recipient list.

For example, if your form allows the person to enter their e-mail address as a “from” field for the e-mail, the following string will cause additional recipients to be included as cc and blind carbon copy recipients of the message:

sender@example.com%0ACc:target@email.com%0ABcc:anotherperson@emailexample.com,stranger@shouldhavefiltered.com

It is also possible for the attacker to provide their own body, and even to change the MIME type of the message being sent. This means that your form could be used by spammers to send mail.

Filter Input

Filter Input

When approaching security, it is best to plan for the worst-case scenario and assume that all input is tainted, and that all user behavior is malicious. You should only use input that you've manually confirmed to be safe.

Filter Input

PHP has a very robust filtering function, filter_var(), which can be used to perform a number of different filtering and sanitizing operations. You can find a list of the filters in the PHP Manual.

http://php.net/manual/en/function.filter-var.php

Filter Input

There are also several functions that can be used to check for individual types of strings. They are locale-aware and so will take language characters into account. The functions will return true if the string contains only characters in the filter and false otherwise.

ctype_ functions
http://php.net/manual/en/book.ctype.php

Filter Input

It is common to perform filtering on the client side, for example using JavaScript in the browser. This is not sufficient and you must filter and validate on the server side as well.

Escape output

Escape Output

Before you emit data, you must make sure that **it is safe for** the client. Recall how XSS attacks work as an example of why you need to make sure that what you send to the client is properly sanitized.

If the data you send to a client includes instructions for it to execute code, then it will do so blindly. You must make sure that you send only code you intend for the client to execute, and not code injected by an attacker.

Escape Output

The most secure way to filter output is using **filter_var()** with the FILTER_SANITIZE_STRING flag.
https://www.w3schools.com/php/func_filter_var.asp

There might be use-cases where this is too restrictive for you, in which case you will need to look at functions like **htmlspecialchars(), strip_tags(), and htmlentities()**.

Escape Output

htmlentities() will encode anything that has an HTML entity representation,
https://www.w3schools.com/html/html_entities.asp

whereas **htmlspecialchars()** will only encode characters that have special significance in HTML.

Escape Output

```
<?php
$string = '© 1982 Sinclair Research Ltd.';
echo htmlentities($string); // &copy; 1982 Sinclair Research Ltd.
echo PHP_EOL;
echo htmlspecialchars($string); // © 1982 Sinclair Research Ltd.
```

Escape Output

This table shows the characters that will be converted by htmlspecialchars().

Character	Becomes
& (Ampersand)	&amp;
" (Double quote)	&quot;
' (Single quote)	&#039;
< (Less than)	&lt;
> (Greater than)	&gt;

Escape Output

Both functions take a flag as their second parameter.

You should make sure that you know at least these three flags as they are important for escaping JavaScript you're outputting:

Flag	Description
ENT_COMPAT	Converts double quotes, not single quotes
ENT_QUOTES	Converts double quotes and single quotes
ENT_NOQUOTES	Does not convert any quotes

Escape Output

When escaping a JavaScript string, you should use the

ENT_QUOTES

flag.

Encryption and Hashing Algorithms

Encryption and Hashing Algorithms

Encryption is a two-way operation; you can encrypt and decrypt.

Hashing is a one-way operation and by design it is difficult or time-consuming to take a hash and reverse it to the original string.

Encryption and Hashing Algorithms

You should **store passwords in the database as hashes**. This way, if attackers get a copy of your database, they are still unable to obtain user passwords unless they can reverse the hash.

Typically, reversing the hash will take a significant amount of time, and hopefully you will have enough time to notice the breach of security and alert your users that they need to change their passwords.

Encryption and Hashing Algorithms

Encryption in PHP is provided by the **mcrypt module**, which needs to be installed and enabled separately. The mcrypt module makes available a wide range of encryption functions and constants.

The algorithms that are available are dependent on the operating system on which PHP is installed. You should not attempt to write your own implementation of an encryption algorithm.

Encryption and Hashing Algorithms

Older hashes like MD5 and SHA1 are very quick to calculate and so **you must not use them** in any place where security is involved.

They are still very useful in other areas of programming, but not in any place where you're relying on them being a one-way operation.

Encryption and Hashing Algorithms

password_hash() function provides a convenient way to generate secure hashes.

By default the password_hash() function uses the bcrypt algorithm to hash the password.

The bcrypt algorithm has a parameter that includes how many times it should run on the password before returning the hashed result.

This is referred to as the “cost” of the algorithm.

Encryption and Hashing Algorithms

By increasing the number of times that the algorithm must run, you can increase the length of time that it takes to calculate a hash.

This means that as computers get faster, you can increase the number of iterations in your bcrypt algorithm to keep your passwords secure from brute force attacks.

You can use the **password_info()** function to retrieve information about how a hash was calculated.

This function will tell you the name of algorithm, the cost, and the salt.

Encryption and Hashing Algorithms

The **password_needs_rehash()** function will compare a hash against the options you specify to see if it needs to be rehashed. This will let you change the algorithm used to hash your passwords, for example increasing the cost over time.