## SUMÁRIO

# 1. SEGURANÇA PHP: ESCOPO DO CÓDIGO

Nesse documento serão tratados as práticas e métodos de segurança a nível do código. Isso envolve filtragem e validação de dados de origem externa, como de formulários, e interna, como do banco de dados, design da construção das classes, encapsulamento de atributos, entre outras coisas.

O escopo do código, como é chamado aqui, termina onde começa o que podemos classificar como escopo dos diretórios e arquivos, mas não se trata de uma divisão abrupta dos escopos, porque é impossível que esses diferentes contextos não se mesclem em alguns momentos.

**Haverá três documentos nessa seção sobre segurança**. O primeiro é este, o segundo será sobre a segurança dos diretórios, que tratará dos locais que não podem ser acessados, e dos arquivos sigilosos, e o terceiro sobre os serviços externos passíveis de serem utilizados para complementar a segurança desse conjunto de fatores.

# 2. PHP SEGURO: TRATAMENTO DE DADOS

## 2.1. TIPAGEM DE DADOS

Embora o PHP seja uma linguagem fracamente tipada, o recurso de tipagem existe na linguagem e pode ser utilizado.

Não são todas as coisas que lidam com dados que podem ter o seu tipo primitivo definido no PHP, mas mesmo nos poucos casos em que é possível, essa prática pode ser entendida como um mecanismo de "afunilamento" dos fluxos de dados, permitindo que o desenvolvedor reduza, de antemão, possibilidades de ataques e até mesmo erros do sistema relacionados a presença de dados indesejados em diferentes fluxos.

**Sendo breve, as coisas que podem ser tipadas em PHP são**: parâmetros de funções, retorno de funções e atributos de classes.

## 2.2. TRATAMENTO DE DADOS EXTERNOS E INPUTS

Uma prática universal de segurança é o tratamento dos dados que são inseridos em uma aplicação, isto é, que são provenientes de fontes externas ao escopo do próprio código em si, e o PHP dispõe de várias funções nativas para realizar esse tipo de procedimento.

Essas fontes externas de dados podem ser formulários, como os utilizados para login e registro, webservices, que retornam dados via HTTP, dispositivos físicos externos etc.

### 2.2.1. EVITANDO ATAQUES XSS

Cross-site scripting (XSS) é um tipo de vulnerabilidade do sistema de segurança de um computador, e que é encontrado normalmente em aplicações web. Na prática esse ataque consiste em injetar scrips no cliente-side da aplicação.

Para evitar XSS existe uma função no PHP chamada de strip_stags. Ela recebe como argumento um dado string, e retorna esse mesmo dado filtrado.

### 2.2.2. EVITANDO SQL INJECTION – SANITIZE E VALIDATE

Injeção de SQL é um tipo de vulnerabilidade normalmente explorada em formulários cujos dados não são devidamente tratados. O ataque consiste em injetar instruções SQL, ao invés dos dados esperados no input, para manipular o banco de dados.

Para evitar esse tipo de ataque, **existe um grupo de funções** que trabalham de maneira semelhante: filter_var, filter_input e array_filter. Eles se diferenciam apenas na forma como realizam a mesma tarefa com dados diferentes. A primeira função trabalha com variáveis quaisquer, a segunda com super globais, como $_GET e $_POST, e a terceira com arrays.

**Seu principal fator é** modo da operação. Ambos possuem um fator que chamamos de "modo", ou "tipo de filtro", e ele possui algumas variações, mas duas podem ser consideradas como principais nesse contexto: **o modo "**sanitize**"**, que filtra o dado segundo uma especificação, retornando-o após a filtragem, e **o modo "**validate**"**, que valida o dado também segundo uma especificação, retornando true ou false.

### 2.2.3. O MÉTODO PREPARE DA CLASSE PDO

A classe PDO, para conexão com o banco de dados, e realização de operações CRUD, é recomendada por uma série de motivos, e um deles é a segurança.

O método prepare, acessível por um objeto PDO, e que tem como função "preparar" uma query SQL, retornando um PDOStatment, também realiza, por debaixo dos panos, um tratamento dos dados que pode evitar ataques de injeção de dados maliciosos.

### 2.2.4. EVITANDO SHELL COMMAND INJECTION

Esse tipo de ataque pode ocorrer quando a aplicação executa funcionalidades no próprio sistema, isto é, opera com funções e comandos de sistema operacional. De fato existem algumas

funções nativas do PHP para realizar operações de sistema operacional, como “system”, “exec” e “passthru”.

Para entender melhor e saber como evitar esse tipo de vulnerabilidade, clique aqui.

### 2.2.5. TRATAMENTO DE UPLOADS

Primeiro a definição breve: upload é o termo usado para ação de envio de dados localmente existentes para um servidor remoto, e normalmente por meio da internet.

Pois bem: quando um formulário HTML tem a funcionalidade de envio de arquivos, o que ocorre é que o arquivo enviado, para ser tratado em um script PHP, é na verdade persistido em uma pasta de arquivos temporários – seu nome é **upload_tmp_dir**.

Para acessar os valores desse arquivo persistido na pasta de arquivos temporários é utilizada o array super global $_FILES, e existem ainda funções nativas para lidar com os valores desse array. Veja aqui as funções para sistemas de arquivos.

Existem diversas tratativas que podem ser realizadas com arquivos. As principais e mais básicas incluem checagem da extensão e do seu tamanho máximo, mas nem uma dessas duas podem, por exemplo, prevenir o upload de arquivos com scripts maliciosos.

Veja esse artigo da Owasp sobre isso.

## 2.3. CRIPTOGRAFIA DOS DADOS

A criptografia é uma técnica que pode ser utilizada para manter as informações dos bancos de dados em sigilo, protegidas. Por meio dela, é possível evitar que pessoas não autorizadas tenham acesso aos dados armazenados, pois somente aqueles que possuem a devida chave de criptografia serão capazes de visualizá-los.

**Existem criptografias irreversíveis**, isto é, que uma vez criadas não podem ser revertidas para o valor original, **e outras que requerem uma chave** para realizar o processo de descriptografia.

**Um último conceito importante é o salt**, que consiste na adição de caracteres, palavras, termos ou mesmo números que dão aleatoriedade ao resultado da criptografia e ajuda a torná-lo mais complexo, e as possibilidades de hashes muito mais diversas.

### 2.3.1. FUNÇÃO PASSWORD_HASH PARA CRIPTOGRAFAR DADOS

A função “password_hash” cria uma senha hash usando o algorítimo padrão bcrypt e, como diz na documentação do PHP, **é fortemente recomendado** o seu uso, pois “usa um hash

forte, gera um salt forte e aplica rodadas apropriadas automaticamente". Seu primeiro parâmetro é o valor a ser criptografado, e o segundo é o modo de criptografia.

**Outras funções seriam** a "**md5**", que não possui um salt, e é considerada uma criptografia ultrapassada, e "**crypt**" que tem o salt como um parâmetro opcional, e um método criptográfico padrão inferior, mas que pode ser alterado.

Teoricamente, a função crypt pode ser tão eficaz quanto a password_hash, mas não é por padrão. Para torná-la são necessárias as especificações necessárias, o que torna a função password_hash a mais recomendada para desenvolvedores no geral.

### 2.3.2. COMO LIDAR COM SENHAS

Senhas para acesso a sistemas são, presumivelmente, armazenadas no banco de dados em sua forma criptográfica. Para chegar a esse resultado, o que pode ser feito é a utilização da função password_hash antes que seja inserida no banco de dados, e, para verificar sua compatibilidade com a informada no campo senha do formulário de login, a função password_verify.

Mas, o que talvez nem todos saibam, é que é uma péssima prática persistir o valor enviado para o backend em uma variável, na sua forma de texto puro, antes do processo de criptografia. Na verdade, **uma senha deve ser criptografada antes de ser persistida em uma variável**.

## 2.4. ENCAPSULANDO ATRIBUTOS SENSÍVEIS DAS CLASSES

O uso das funcionalidades de encapsulamento de atributos, como private e protected, é uma prática imprescindível para preservar a privacidade dos valores de um contexto que, nas regras de negócio, não podem ser de acesso público.

## 2.5. MÉTODO SERIALIZE

O método "serialize" do PHP é uma ótima forma de transitar dados entre diferentes contextos da aplicação respeitando a privacidade original dos dados – coisa que o JSON não faz.

Por exemplo, se o atributo "$senha" for definida como "private" em uma classe X, se enviada para uma outra, digamos a classe Y, um print_r($senha) não irá revelar o seu valor verdadeiro; como valor, serão mostrados apenas alguns asteriscos, dando a ideia de que é um valor oculto.

### 2.5.1. ATAQUES ENVOLVENDO DADOS SERIALIZADOS

Como diz nesse artigo, é possível manipular uma estrutura serializada, para que, quando descontruída, crie fluxos indesejados e maliciosos dentro do sistema.

Para evitar isso é fortemente recomendado que os dados sejam filtrados e validados após um processo de “unserialize()”.

## 3. PHP SEGURO: SESSIONS E COOKIES

**Primeiro vamos começar pela conceituação**: navegadores não preservam dados por padrão, porque o protocolo HTTP é stateless, isto é, não possui estado persistido. O que ocorre, na prática, é que recursos são requisitados a um servidor por um cliente, e esse último quando os recebe utiliza o navegador como meio para interagir com o que recebe.

**As requisições são independentes** e possuem um tempo de vida (conexão, envio de mensagem, resposta, encerramento da conexão). **O servidor web não é capaz de identificar se duas requisições vieram de um mesmo navegador**, e ele não faz nenhum gerenciamento em memória para que mensagens sejam compartilhadas entre requisições.

É para suprir esta necessidade que entram os cookies e sessões.

### 3.1. A SOLUÇÃO DOS COOKIES

Tecnicamente falando, um cookie é uma pequena quantidade de **informação persistida** temporariamente pelo navegador (em uma pasta), isto é, no **lado do cliente**. Os navegadores normalmente limitam o tamanho dos cookies em até 4KB, e apagam cookies com a data de “validade vencida”.

**Eles são gerados no lado do servidor,** enviados para o lado do cliente, e persistidos em uma pasta do navegador por um tempo determinado. **A cada nova requisição para o mesmo domínio** os dados persistidos no lado do cliente são novamente enviados para o servidor, juntamente com a própria requisição dos recursos para montar a página.

Através de cookies o servidor web é capaz de trocar informações de estado com o navegador do usuário. Desse modo, somos capazes de adicionar produtos a um **carrinho de compras**, sem perder estas informações ao mudar de página, sair do website ou até mesmo fechar o navegador.

**Mas atenção**: os cookies não são mais utilizados para esses fins. Ao invés deles, existem soluções mais modernas, como o LocalStorage, que é manipulado pelo Javascript e suporta até 5MB de dados.

**Nos tempos atuais**, ao invés de serem utilizados para persistir informações variadas, **eles são utilizados para armazenar ID de sessõe**s. Esse é o próximo tópico.

## 3.2. A SOLUÇÃO DAS SESSIONS...OU "COOKIES DE SESSÃO"

As sessões têm um princípio similar aos cookies, só que o armazenamento do estado, ou dos dados da variável, é feito pelo servidor web, normalmente por meio de um banco de dados, e não pelo navegador.

Na realidade, apesar de existirem esses dois nomes, as sessões são cookies, mas que possuem uma forma de operacionalidade diferente, e por isso são as vezes chamadas de "cookies de sessão".

**Quando iniciamos uma sessão**, é criado um arquivo temporário, em uma pasta do PHP, com o nome sess_[ID], que conterá os valores acessados pela super global $_SESSION, e ao mesmo tempo é enviado um cookie para o navegador cujo nome é PHPSESSID, com um valor único que corresponde ao [ID] da sessão inicializada.

**Ou seja**, com o comando "session_start()" é gerada uma sessão com um ID aleatório, que chamamos de "ID de sessão", que é utilizado no nome do arquivo que persistirá os dados acessados com $_SESSION, o sess_[ID], e, por último, persistido em Cookie, o PHPSESSID.

**Por exemplo**, se o ID da sessão inicializada for "ABC", hipoteticamente, o nome do arquivo temporário de sessão será sess_ABC, e o cookie enviado para o navegador, cujo nome por padrão é PHPSESSID, terá o valor ABC.

**Ok, mas e aí? Onde os dados são armazenados?** Pois bem: esse Cookie de vida limitada, e nome PHPSESSID, é sim persistido no navegador como qualquer outro, mas seu valor será apenas o ID da sessão inicializada, enquanto os dados propriamente ditos serão armazenados pelo servidor, em um banco de dados, junto com o ID da sessão, que será o parâmetro da procura.

**O resultado é esse**: a cada nova requisição o navegador informará ao servidor que ele possui um cookie chamado PHPSESSID. A partir daí o PHP irá recuperar o valor do Cookie, que é o ID da sessão, e utilizar ele para procurar os dados da sessão no banco de dados.

Encontrando o registro que possui o ID da sessão, ele irá capturar os dados a esse valor relacionados de alguma forma, e irá atribuí-los ao array super global $_SESSION.

As sessões, ou cookies de sessões, são normalmente utilizados para persistir logins, e informações gerais ao longo das páginas das aplicações. Mas, **é recomendado que seu uso seja moderado** porque, dado o seu funcionamento, sempre haverá o processamento adicional de procura no banco de dados pelos valores relacionados ao ID de sessão.

## 3.3. SEGURANÇA DOS COOKIES E DAS SESSÕES

### 3.3.1. COOKIES NÃO PODEM TER DADOS SENSÍVEIS

Os cookies de sessão são normalmente utilizados aliados a bancos de dados, porque dados sensíveis não podem existir em qualquer lugar.

### 3.3.2. O NOME DO COOKIE DEVE SER ALTERADO PARA CADA USUÁRIO

O nome padrão utilizado pelo PHP é PHPSESSID, mas para que sejam evitados roubos de sessão é sempre bom definir nomes diferentes.

Um bom método para realizar esse procedimento é criar um hash criptográfico como nome, e utilizando dados relacionados ao ambiente do cliente. Por exemplo:

```
session_name(password_hash($_SERVER[“HTTP_USER_AGENT”]));
```

### 3.3.3. LIMITE DE VIDA PARA AS SESSÕES

Mesmo que um cracker consiga acessar o seu sistema roubando uma sessão, ele terá pouquíssimo tempo para agir se você limitar o tempo de vida das sessões.

```
session_cache_expire(10); // Irá expirar em 10 minutos
```

### 3.3.4. SEMPRE VALIDAR A ORIGEM DA SESSÃO

Utilizando um banco de dados para armazenar os dados da sessão, como também seu próprio ID, pode ser criada uma tabela só para isso, e que também tenha diversos outros dados sobre o cliente original.

Por exemplo, essa tabela pode ter como chave primária o ID da sessão, os dados dela, e o IP do cliente, bem como outros dados que possam ser relacionados a ele e servir em validações adicionais.