SUMÁRIO

1. COMMAND INJECTION

Esta classe de ataques, muito comum em ambientes web, consiste na inserção de comandos nos dados passados como entrada para a aplicação de forma que estes comandos sejam executados pela ou através da aplicação.

Existem duas subclasses mais comuns, de ataques de Command Injection, que são **SQL injection** e **Shell Command Injection**.

1.1. SHELL COMMAND INJECTION

Este tipo de ataque pode ocorrer quando a aplicação executa funcionalidades no próprio sistema, isto é, opera com funções e comandos de sistema operacional.

Para tornar a questão clara, existem algumas do próprio PHP para realizar operações de sistema operacional, mas serão citadas três dentre as existentes: **system( )**, **exec( )** e **passthru( )**.

1.1.1. SOLUÇÃO

Imagine uma aplicação PHP hipotética que executa comandos de exploração de diretórios passados por um formulário HTML, via método POST, e utilize a função system( ) para executa-los. Assim, os comandos, como valores, seriam repassados para a variável $_POST[“cmd”], e após para a variável $cmd. Portanto, $cmd = $_POST[“cmd].

Neste caso, além de executar o comando passado como valor no Input, pelo usuário comum, a aplicação também poderia receber um outro adicionado pelo atacante. **Para resolver isto**, e que de fato é o caso mais comum para este ataque, para a variável final $cmd, ao invés de $_POST[“cmd”], se atribuiria este como parâmetro da **função escapeshellcmd()**, que anula quaisquer outros valores além dos que já haviam sido capturados. Assim seria, portanto, **$cmd = escapeshellcmd($_POST[“cmd]).**

1.2. SQL INJECTION

Injeção de SQL é um tipo de ameaça de segurança que se aproveita de falhas em sistemas que interagem com bases de dados através de comandos SQL. De modo geral, é análogo ao tipo de ataque citado no tópico anterior, mas, neste caso, o atacante insere um comando SQL para manipular um bando de dados.

1.2.1. SOLUÇÃO

Imagine uma aplicação de Login, em que, como qualquer outra, para logar o usuário precise inserir um “nome de usuário” e uma “senha” que existam em um mesmo registro de uma tabela do banco de dados.

Para isso, o sistema irá recuperar os dados passados no Input do formulário de Login, e utiliza-los em um comando SELECT direcionado ao banco de dados. Este comando SELECT irá procurar por registros cujos valores de “nome de usuário” e “senha” sejam iguais aos informados pelo sujeito que tenta acessar o sistema. Seria, portanto:

$sql = "SELECT * FROM Usuario WHERE nome_usuario = $nome AND Senha = $senha

Deste modo conseguimos verificar o email e a senha para um usuário comum, cujas intenções sejam éticas e apenas aquelas previstas para a aplicação.

Mas, um usuário malicioso, com conhecimento, pode tentar passar códigos SQL nos inputs do form. Por exemplo, pode passar um nome de usuário qualquer, como “Fulano” mas no campo senha um comando SQL, como ' or id = '1.

Assim, como o comando SQL, no código PHP, atribuído a uma variável, é escrito entre aspas duplas, e os comandos entre aspas simples, a senha passada pelo atacante se tornaria ‘’ or id = ‘1’. Desta forma, o comando SELECT estaria selecionando tudo da tabela onde o nome de usuário é igual a "Fulano" e senha é “vazia OU o id igual a 1”.

Existem diversas formas de lidar com isso, mas a mais comum é utilizar, o método **prepared statement** da classe PDOStatment. Ou seja, para operações em banco de dados, é crucial a utilização da classe PDO.

Também é interessante realizar validações, como não espaços em branco em um valor passado, exigir caracteres especiais, verificar o tipo do valor inserido e verificar o método da requisição – deve ser sempre POST.

2. PERMISSÕES DE PASTA

Como diz na documentação do PHP, é permitido o controle de quais arquivos no sistema podem ser lidos e por quem, e ressalta que de fato é preciso ter cuidado com quaisquer arquivos que possam ser manipulados, em uma aplicação. Isto significa definir o nível de acesso de funcionalidades que envolvem a manipulação de diretórios.

Muito importante ressaltar que o que será apresentado em seguida é **funcional no sistema operacional Unix**.

2.1. SOLUÇÃO

Imagine que uma aplicação PHP crie diretórios, se não existem, e os manipule. Para isso uma condicional poderia ser escrita, como:

$pasta = "./pastaa"

if(!is_dir($pasta)){ mkdir($pasta), }

Ou seja, "se não existir uma pasta $pasta (sendo o valor da variável um endereço de pasta), crie uma pasta $pasta". O valor da variável, ./pastaa, significa uma pasta, chamada "pastaa" no mesmo diretório do script PHP. Ou seja, se não existir essa pasta, o código irá cria-la.

Mas, a função mkdir( ) aceita também como segundo parâmetro um "modo", isto é, neste caso o nível de permissão da pasta criada, ou, em outras palavras, o que é ou não permitido realizar na própria pasta.

Ver em: https://www.php.net/manual/pt_BR/function.mkdir.php

3. RECAPTCHA PARA IMPEDIR ROBÔS

O reCAPTCHA é uma tecnologia do Google que se faz presente em sites e aplicativos comuns na vida dos usuários, sendo utilizado para fins de segurança. O objetivo é limitar o acesso e tráfego criado por bots e outros tipos de robôs.

Uma curiosidade: para o próprio Google, tem uma utilidade diferente, que é o reconhecimento de elementos em imagens, que o próprio OCR não conseguiu identificar. As respostas médias são armazenadas e servem para desenvolver e solucionar os erros desta tecnologia.

Na documentação do Google, sobre o reCaptcha, há um guia para implementar o sistema em um site. Veja em: https://www.google.com/recaptcha/about/

4. CROSS SITE SCRIPTING (XXS)

Muito similar a uma injeção de comando SQL, este tipo de ataque consiste na inserção de tags HTML ou Javascript via estruturas de entradas de dados, como Inputs de formulários.

4.1. SOLUÇÃO

Para impedir ataques deste tipo, de injeção de tags, a variável de recuperação de valores via HTTP, pode e deve ser filtrada.

Além das validações comuns, e do uso da função “filter_input()”, com os parâmetros de filtragem adequados para a situação, ao invés de $_GET, e $_POST, deve-se utilizar uma outra função, “strip_tags()” para filtrar tags, se existentes. Veja abaixo:

```php
<?php

    //Recuperação do valor com filter_input(), e filtragem com o modo FILTER_SANITIZE_STRING
    $valor_input = filter_input(INPUT_POST, "nome_valor", FILTER_SANITIZE_STRING);

    if(!empty($valor_input)){

        //Irá retornar o valor filtrado pela função
        $valor_filtrado = strip_tags($valor_input);

        //Para verificar o valor
        echo $valor_filtrado;
    }


?>
```

Fonte: autor

Em primeiro lugar, o que está sublinhado em verde é a recuperação dos dados com a função filter_input(), sendo seus parâmetros (i) o método de requisição HTTP, GET ou POST, (ii) o nome referencial do valor, definido no campo name, do formulário HTML, e (iii) o filtro utilizado – existem diversos modos de filtragem.

Em segundo lugar, o que está sublinhado em vermelho é a função de filtragem de tags, utilizada para evitar ataques XXS. **Portanto**, primeiro o valor é filtrado com a função filter_input() e atribuído a uma variável, e esta por sua vez, é filtrada, após a condicional, novamente pela função strip_tags();

5. DEFESA EM CAMADAS

Defense in Depth (DiD) é uma abordagem à cibersegurança em que uma série de mecanismos defensivos são dispostos em camadas para proteger dados e informações valiosas. Se um mecanismo falhar, outro intervém imediatamente para impedir um ataque.

Este conceito, chamado originalmente de “Defense in Depth”, **se divide em duas classes**: segurança física e segurança do ambiente de programação.

5.1. SEGURANÇA DA ARQUITETURA FÍSICA

Na defesa da arquitetura física é envolvida a segurança do servidor web, onde estão hospedados os arquivos e dados do site. Neste sentido, a segurança envolve diversos aspectos, desde a possibilidade de escalabilidade da aplicação, até a garantia de acesso restrito aos documentos hospedados.

Neste caso é fortemente recomendado, se a arquitetura física for terceirizada, que se utilizem os serviços de empresas reconhecidas por manterem boas políticas de segurança. E que serviços seriam esses? Diferentemente do passado, em que se utilizavam sites de hospedagem, **hoje se tornou comum a utilização de máquinas virtuais, isto é, servidores na nuvem**.

A exemplo de serviços de locação de máquinas virtuais, que podem ser utilizadas para hospedar sites com segurança, existe a AWS (Amazon Web Services), Microsoft Azure, Digital Ocean, Google Cloud, entre outros.

5.2. SEGURANÇA DA APLICAÇÃO EM SI

Neste sentido, para deixar claro, e em termos simplistas, uma aplicação não pode depender de uma única verificação, ou de uma única validação, mas sim de diferentes mecanismos que se complementam e fortalecem.

Ver mais em: https://en.wikipedia.org/wiki/Defense_in_depth_(computing)

Ver mais em: https://phpsecurity.readthedocs.io/en/latest/Introduction.html

6. CRIPTOGRAFIA EM PHP

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 diferentes métodos para criptografar dados, e além de se diferirem na maneira que são criptografados, ou seja, no script que criptografa, se diferem também nos aspetos referentes à possibilidade de retornar a variável criptografada para a variável real.

Importante citar que a partir do entendimento do primeiro método, MD5, apresentado a seguir, o entendimento dos outros, e de suas comparações, se torna muito mais fácil.

6.1. MÉTODO MD5

A função MD5( ) calcula o "hash MD5" de uma string, que é um número hexadecimal de 32 caracteres. Além disso, no próprio PHP, não há função para descriptografa-lo, sendo, portanto, uma criptografia de mão única.

O que não significa, porém, que não pode ser descriptografado. Na verdade, apesar de bastante utilizado, o método md5() não é considerado tão seguro, se comparado com outras formas criptografia, como o bcrypt, utilizado na função password_hash.

6.1.1. SOBRE SALT

O MD5() não inclui um salt. E o que seria isso? **O salt consiste** na adição de caracteres, palavras, termos ou mesmo números que dão aleatoriedade ao resultado da criptografia e ajuda a torná-lo mais complexo.

O princípio fundamental de um salt é a sua utilidade para gerar hashes únicos; com uma mesma palavra, se utilizada criptografia que utilize salt, serão sempre gerados hashes diferentes.

Pois bem: isto explica, em parte, o porquê do método MD5 não ser seguro se comparado a outros, que incluem um salt, pois um atacante poderia, a partir do valor criptográfico registrado no banco de dados, descobrir o exato valor correspondente.

6.1.2. EXEMPLO DE USO

Imagine um sistema de Login em que, para logar, o usuário precise inserir um nome de usuário e senha existentes em um mesmo registro da tabela de usuários, do banco de dados.

No próprio banco de dados, a senha não será salva no seu formato original, mas no próprio cadastro o valor escolhido pelo usuário, como senha, será criptografado com a função md5(). Portanto, na tabela de usuários, será registrado, em uma linha, o nome do usuário e a sua senha em formato de hash de 32 caracteres.

Sempre que o usuário tentar logar na sua conta, irá informar um nome de usuário e senha, mas esta última será criptografada, com a mesma função md5(), no próprio script de validação – que irá validar os dados inseridos pelo usuário.

Desta forma, quando realizado o SELECT na tabela de usuários do banco, os valores procurados, em cada registro, serão o nome de usuário e a senha em formato de hash. Assim, se existir um registro com o nome de usuário informado, no campo de nome, e o valor de

hash, existente no campo “senha” -deste registro da tabela-, for igual ao do valor digitado no Input de senha -do login-, será uma confirmação da existência da combinação user-password informados, e o acesso ao sistema será concedido.

6.2. MÉTODO BASE64

Apesar de ser utilizado, e ensinado com recorrência, o método Base64 não é criptografia, e sim um mecanismo de codificação facilmente reversível.

No PHP, a função “base64_encode()” gera, a partir do valor passado como parâmetro, um código de 64 bits, e a função “base64_decode()” retorna o código gerado para o valor original.

6.3. MÉTODO PASSWORD_HASH

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.

Assim, diferentemente do MD5, este método utiliza um salt que, ou pode ser definido pelo desenvolvedor, ou, se não for, é gerado automaticamente. Por via de regra, não é preciso definir um salt, pois o gerado pela própria função é bastante seguro.

Além de tudo, com a função password_verify(), cujo primeiro parâmetro é um valor, e o segundo o hash, é possível verificar a compatibilidade entre ambos.

Ver mais em: https://www.php.net/manual/pt_BR/function.password-hash.php

6.4. MÉTODO CRYPT

Este método unidirecional é, em certo sentido, o anterior ao password_hash(). Usa salt como parâmetro opcional, o que resulta em criptografias mais fracas, e em versões mais recentes do PHP retorna um E_NOTICE caso não lhe seja fornecido um salt.

Diferente do password_hash() que utiliza apenas o algoritmo bcrypt, o método crypt() utiliza o algoritmo DES como padrão, mas suporta diversos outros algoritmos de encriptação.

Não é, portanto, em si, inferior ao password_hash(), e sim mais flexível, pois se difere deste apenas quando no modo de aplicação. Talvez, devido a isto, no que tange a

eficiência, o password_hash() seja preferível, pois já define, por si mesmo, configurações bastante adequadas para criptografar valores.

7. SEQUESTRO DE SESSÃO E SSL

Este ataque, que consiste no roubo de sessão, se dá quando um invasor intercepta e assume uma sessão legitimamente estabelecida entre um usuário e um host.

A finalidade de uma sessão é armazenar dados para usuários individuais usando um ID de sessão único. IDs de sessão normalmente são enviados ao navegador através de cookies de sessão, e o ID é usado para recuperar dados da sessão existente.

Por exemplo, um usuário, por meio de uma sessão criada para si, com um ID único, em um acesso a um site de compras, poderia gravar seus produtos escolhidos e continuar suas compras, ou buscas, em um acesso posterior – não precisaria, a cada acesso, ter que encher seu carrinho de compras novamente.

Se um sequestro de sessão é realizado com êxito, é possível que o atacante tenha acesso a todas as transações e ações do usuário no site, podendo, portanto, roubar e corromper dados, pois o servidor irá compartilhar todas as informações relacionados ao ID de sessão informado.

7.1. SOLUÇÃO

Existem duas classes de medidas principais para evitar roubos de sessão, sendo a primeira delas relacionada à codificação, isto é, o algoritmo da aplicação PHP, e a segunda com o próprio tráfego dos dados, o que refere ao uso de SSL.

7.1.1. SOLUÇÃO SSL - HTTPS

SSL, ou Security Socket Layer, é uma camada que permite que aplicativos cliente/servidor possam trocar informações em total segurança, protegendo a integridade e a veracidade do conteúdo que trafega na Internet.

Este aspecto da segurança, que é implementado no lado do servidor, é realizado pela equipe responsável pela infraestrutura, não sendo, portanto, uma implementação de código em si.

7.1.2. SOLUÇÃO DE CÓDIGO

Uma solução cabível, além do uso do SSL, é a utilização da função **session_regenerate_id()**, após a destruição da sessão com **session_destroy()**.

Esta solução se resume a destruir o ID de uma sessão, com a função de destruir sessões, e com a primeira função, de regenerar, gerar um novo, mas mantendo as informações antigas, pois, como diz na documentação do PHP, a função “salva os dados da sessão antiga antes de a encerrar”.