André Milani
Softblue

PHP TIPS

e-book

Sumário

Apresentação

A Softblue é uma empresa de cursos online na área de programação. Fundada em 2003 na cidade de Curitiba-PR, a empresa conta atualmente com um grande portfólio de cursos e dezenas de milhares de alunos espalhados em dezenas de países pelo mundo.

Este e-book foi criado por André Milani, sócio fundador da Softblue. André Milani é formado em Ciência da Computação pela PUC-PR, pós-graduado em Business Intelligence pela mesma instituição e possui diversas certificações na área de TI. É também autor de vários livros na área de informática, entre eles o *Programando para iPhone e iPad*, *MySQL - Guia do Programador* e *Construindo Aplicações Web com PHP & MySQL*, todos pela editora Novatec. Atua desde 2003 com desenvolvimento web e treinamentos de profissionais. Também é desenvolvedor de aplicativos para o ambiente iOS da Apple, possuindo aplicações que juntas somam mais de 50.000 downloads na AppStore.

Disclosure

1. Introdução

O objetivo deste e-book é apresentar alguns dos artigos escritos por André Milani sobre a linguagem de programação PHP, abordando algumas das dúvidas mais comuns de quem está começando a trabalhar em seus primeiros projetos com esta tecnologia.

2. Tratando "variable is not defined" no PHP

A mensagem de erro `variable is not defined` é uma antiga conhecida de programadores PHP e está relacionada diretamente ao acesso a uma variável que não tenha sido inicializada com algum valor anteriormente. Ao receber dúvidas de vários alunos, resolvi comentar os casos recebidos e apresentar soluções.

2.1. Falta de definição de valor (caso simples)

Em um dos códigos que observei, um comando `for` era montado da seguinte forma para realizar uma somatória:

```
for($i=0; $i<10; $i++)
{
        $total += $i;
}

echo $total;
```

Este código resultará no erro `variable is not defined`,pois a variável `$total` não foi inicializada antes de ser utilizada. Como a variável não foi inicializada, na primeira execução do bloco de código do comando `for` o compilador tentará atribuir para `$total` o resultado da operação `nada + 0`. Observe que para o compilador `nada` é diferente de `0`, o que acarretará na inviabilidade de realizar a soma. Para solucionar este problema, basta atribuir um valor para a variável `$total` antes do comando `for`, da seguinte forma:

```
$total = 0;

for($i=0; $i<10; $i++)
{
        $total += $i;
}

echo $total;
```

2.2. Falta de definição de valor (caso register_globals)

O PHP possui um recurso conhecido como `register globals`, uma configuração no arquivo `php.ini` que permite que algumas variáveis sejam inicializadas pelo PHP. Vale a pena citar que este recurso tornou-se obsoleto a partir do PHP 5.3.0 e seu uso não é mais encorajado.

Alguns programadores que utilizam o `register_globals` têm o costume de acessar diretamente suas variáveis, da seguinte forma:

```
if($_REQUEST["minhaVar"] == "valor")
```

```
{
    ...
}
```

Até aí, tudo certo. Contudo, quando o programador migra seu código para outro servidor PHP, ou obtém um código deste tipo na web e o seu servidor não está configurado para utilizar o `register_globals`, o erro `variable is not defined` é exibido nas situações onde o código apresentado é executado e não há a variável `minhaVar` na requisição da página. Nestes casos, como o programador pegou o código de outro servidor onde o código já estava funcionando, ele tem dificuldades em entender o motivo pelo qual o código não executa em seu servidor. O motivo é: um dos servidores está com a `register_globals` habilitada, o outro não. Mas tem solução!

2.3. Solução 1: habilitar a register_globals

A solução mais simples para a maioria dos casos é habilitar o recurso de `register_globals` no PHP. Isto é fácil de ser feito e existe bastante documentação no site do PHP e no Google sobre este procedimento. Contudo, pode não ser viável, dependendo da versão do seu PHP. Neste caso, outra solução deve ser adotada.

2.4. Solução 2: tratando o código

É possível tratar no código o acesso das variáveis para que o acesso a elas quando não definidas seja realizado. Basta realizar um comando que verifica se a variável em questão foi setada ou não. O comando `isset` faz isso: ele verifica se determinada variável existe e se pode ser acessada ou não. O mesmo código apresentado anteriormente poderia ser alterado para o seguinte formato:

```
if(isset($_REQUEST["minhaVar"]))
{
    if($_REQUEST["minhaVar"] == "valor")
    {
        ...
    }
}
```

Desta forma, somente se a variável `$_REQUEST["minhaVar"]` possuir algum valor é que o acesso à mesma será realizada dentro do bloco de código do primeiro `if`. Caso contrário, o acesso não será realizado, e a mensagem de erro não surgirá.

Vale a dica: é possível otimizar o comando `if` da seguinte forma:

```
if(isset($_REQUEST["minhaVar"]) && $_REQUEST["minhaVar"] == "valor")
{
    ...
}
```

O que ocorre nesta forma de comando `if` é que a segunda condição (a de acesso à variável) somente é realizada se a primeira condição do `if` for verdadeira. Ou seja, a variável somente será acessada se possuir algum valor.

3. PHP e arquivos em cache

Neste artigo, eu gostaria de compartilhar uma situação que aconteceu com um leitor do meu livro de PHP e MySQL sobre cache. Levamos um bom tempo para conseguir detectar que o problema não estava no código PHP, mas sim no cache do navegador. Então, para prevenir este problema, vão aí algumas dicas bem interessantes.

3.1. Forçando um refresh

Muitos sabem que a tecla F5 dos navegadores, pelo menos na maioria deles, serve para atualizar a página. Este procedimento é chamado de refresh (nome em inglês). O que nem todos sabem é que o comando F5 realiza um refresh com base em algumas configurações de data de expiração dos arquivos em questão, e que dependendo da data, somente a tela é atualizada, mas não os conteúdos dos arquivos cujas datas ainda não expiraram.

Para resolver esta situação, utilize o comando CTRL + F5. Este comando força o refresh dos arquivos, independentemente da data de expiração. Ele apaga do cache do seu navegador todos os arquivos da página em questão, e solicita todos os arquivos novamente ao servidor.

3.2. Evitando o cache em PHP

Mesmo conhecendo o comando CTRL + F5, pode ser que nem todos os visitantes de nossos sites conheçam este recurso, ou ainda, saibam que a página está em cache. Por este motivo, é interessante prevenir que as páginas utilizem cache, fazendo com que seus visitantes sempre acessem as páginas com as últimas atualizações realizadas.

Para prevenir o uso do cache em PHP, utilize o comando `header` setando duas propriedades específicas:

```
// HTTP/1.1
header("Cache-Control: no-cache, must-revalidate");

 // Data no passado
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
```

A primeira propriedade é a `Cache-Control`, que informa ao navegador, ou qualquer proxy que possa estar no caminho do arquivo, para não armazená-lo em cache.

A segunda propriedade é a `Expires`, que define a data de expiração do arquivo. É possível definir uma data já passada, para que o cache já seja entendido como expirado, ou então é possível definir a sua data de expiração. Esta propriedade deve

ser informada para arquivos que possuem alto volume de acesso, mas baixa taxa de atualização, o que otimiza recursos do servidor.

Vale a pena mencionar que, ao utilizar o comando `header`, nenhuma informação pode ser impressa antes pelo PHP, nem por meio de comando `echo`, `print` ou outro. Esta é uma particularidade do comando `header`, e maiores informações podem ser encontradas no manual do PHP, neste endereço:

http://php.net/manual/pt_BR/function.header.php

4. Protegendo senhas

Neste artigo eu pretendo debater um pouco sobre a proteção no armazenamento de senhas de clientes e usuários de nossos sites em bancos de dados. Por mais que já estejamos carecas de saber que nossos sites devem prevenir ataques de SQL Injection (inserção de códigos SQL em campos de formulários ou parâmetros de request) e de outros tipos, não deixa de ser uma medida de prevenção proteger ainda mais as senhas cadastradas por nossos usuários em nossos sites, uma vez que temos usuários que utilizam uma senha padrão em vários serviços (obviamente não recomendado), e até mesmo para evitar que outra pessoa de posse da senha do usuário em nosso site possa realizar qualquer operação não autorizada pelo titular da conta.

Os códigos apresentados neste artigo são da linguagem SQL e alguns comandos da linguagem PHP, mas podem ser convertidos para a maioria das linguagens de programação. Para exemplificar o conteúdo abordado neste artigo, vamos considerar inicialmente que temos a seguinte tabela em nosso banco de dados:

```
ID | USUARIO | SENHA
1    André     123456
2    Milani    abcdef
```

Mesmo supondo que nosso site seja seguro contra SQL Injection, todos que tiverem acesso ao banco de dados poderão ter acesso às senhas dos usuários. Isto inclui pelo menos os administradores do banco, e em alguns casos também os responsáveis pela hospedagem do site, até mesmo desenvolvedores e pessoas com acessos aos arquivos de backup do banco. Um dos métodos mais tradicionais para se proteger a senha dos usuários é por meio da geração de um código-hash.

4.1. Gerando um código-hash

Um código hash é o resultado de um algoritmo de criptografia que recebe como parâmetro um dado, que neste caso pode ser a senha do usuário. Algumas funções são conhecidas como one-way (via única), onde a partir do resultado não é possível realizar o caminho inverso e assim descobrir a senha cadastrada ou o dado passado como parâmetro. Alguns exemplos de algoritmos de hash one-way são o `MD5` e o `SHA1`.

O algoritmo `MD5` retorna uma string com 32 posições preenchidas com caracteres hexadecimais, que é formada a partir de uma fórmula própria.

```
echo md5("123456");  // e10adc3949ba59abbe56e057f20f883e
echo md5("abcdef");  // e80b5017098950fc58aad83c8c14978e
```

Já o algoritmo `SHA1` retorna uma string com 40 posições, formada por uma fórmula própria e diferente do `MD5`.

```
echo sha1("123456");  // 7c4a8d09ca3762af61e59520943dc26494f8941b
echo sha1("abcdef");  // 1f8ac10f23c5b5bc1167bda84b833e5c057a77d2
```

Tanto o `MD5` quanto o `SHA1` retornam sempre o mesmo resultado para o mesmo parâmetro informado. Isto quer dizer que sempre que a função `MD5` for chamada para o parâmetro `123456`, o resultado será sempre `e10adc3949ba59abbe56e057f20f883e`. De posse do código-hash da senha do usuário no momento de seu cadastro, basta armazenar no banco o código-hash em questão, transformando nossa tabela `USUARIOS` fictícia para o seguinte formato:

```
ID | USUARIO | SENHA
1    André     e10adc3949ba59abbe56e057f20f883e
2    Milani    e80b5017098950fc58aad83c8c14978e
```

Neste momento, mesmo com acesso (autorizado ou não-autorizado) nesta tabela, a senha dos usuários não fica tão exposta quanto anteriormente, diminuindo muito as chances de alguém que acesse esta tabela possa realizar alguma ação indevida com estes dados.

4.2. Autenticando a senha com o código-hash

Como fazer então para, no momento em que um usuário retornar ao site e desejar se autenticar, validar se a senha que ele informou no login é a mesma senha criptografada no banco? É simples: basta novamente gerar o código-hash da senha informada pelo usuário no momento da autenticação e verificar se os códigos hash são idênticos. Por exemplo:

```
$usuarioLogin = "andre";
$senhaLogin = "123456";
$senhaLoginHash = md5(senhaLogin);

$sql = "SELECT * FROM USUARIOS WHERE USUARIO = '".$usuarioLogin."' AND SENHA =
'".$senhaLoginHash."'";
```

Se a consulta SQL apresentada retornar algum resultado, é porque temos um usuário autenticando-se com sucesso. Caso contrário, ou o usuário ou a senha foram informados de forma incorreta. Observe neste código que foi utilizada a função `MD5` para abordar o exemplo. É importante salientar que é a mesma função de criptografia que deve ser utilizada no cadastro do usuário e no momento de autenticação, ou seja, se no cadastro for utilizado a função `SHA1` para criptografar a senha, é a função `SHA1` que deve ser utilizada no momento da autenticação.

4.3. Proteção contra ataques que força-bruta

Você já protegeu as senhas de seus clientes no banco de dados utilizando criptografia, porém o seu sistema ainda pode estar vulnerável contra ataques de força-bruta. Este tipo de ataque consiste em tentar milhares de combinações de senhas para um determinado usuário pré-conhecido. Por exemplo: um usuário mal-intencionado poderia, de posse do nome do usuário `andre`, criar um script que enviasse milhares de tentativas de senhas (ou de um dicionário de senhas), até obter sucesso (similar ao tipo de ataque que foi usado para descobrir a conta do Barack Obama no Twitter há um tempo atrás). Para evitar este tipo de ataque, basta você criar um contador de tentativas falhas de autenticação em sua tabela `USUARIOS`, deixando-a da seguinte forma:

```
ID | USUARIO | SENHA                            | FALHAS
1    André     e10adc3949ba59abbe56e057f20f883e   0
2    Milani    e80b5017098950fc58aad83c8c14978e   0
```

Bastaria então incrementar o valor do campo `FALHAS` a cada tentativa inválida que for realizada para o usuário em questão. Uma regra em seu sistema pode definir que, a partir de três tentativas inválidas de autenticação para um usuário, o sistema bloqueie sua conta, não permitindo novas tentativas e solicitando que o usuário entre em contato para desbloquear a sua senha. Outra regra que poderia ser realizada é: ao atingir três senhas inválidas, o sistema não permite que o usuário em questão possa realizar novas tentativas de autenticação nos próximos 15 minutos, o que tornaria um ataque força-bruta bastante inviável.

Vale a pena comentar que o contador `FALHAS` deve ser zerado toda vez que o usuário em questão consiga se autenticar com sucesso, para evitar que, caso ele esqueça a senha de vez em quando, este contador bloqueie sua senha em um momento futuro sem que ele tenha errado a senha três vezes consecutivas.