

1 ANO de edições

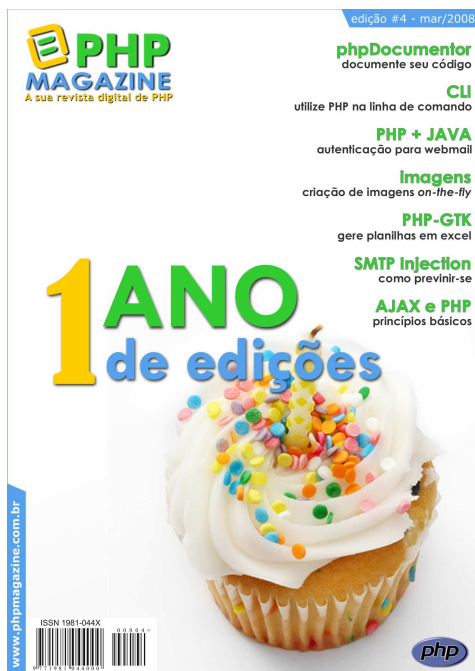


ISSN 1981-044X



9 771981 044000





PHP + JAVA

PHP-GTK

SMTP Injection

AJAX

editorial

Apresentação

Equipe

Mensagens de leitores

Chamadas

artigos

phpDocumentor

por Maykel dos Santos Braz

Desenvolvendo em PHP para linha de comando - CLI

por Helton Eduardo Ritter

Ataque de sobrecarga utilizando SMTP Injection. O que é, como é e como se prevenir

por Ricardo Striquer Soares

Gerando Planilhas Excel com PHP-GTK

por Pablo Dall'Oglio

AJAX e PHP I: Conhecendo AJAX

por Rafael Dohms

Webmail com Java + PHP

por Guilherme Gall, Pedro Lara, Fabio Borges

Criação de imagens on-the-fly

por Leandro Schwarz

Caro leitor,

Um ano após o primeiro exemplar, apresentamos a quarta edição da PHP Magazine, a única publicação nacional sobre tecnologia PHP. Alguns imprevistos prejudicaram o lançamento desta edição ainda em 2007, como era do interesse da equipe. Reconhecemos nossa falha perante a nossa comunidade. Pedimos desculpas, e para compensar, preparamos uma revista repleta de assuntos interessantes.

Nesta edição, selecionamos 7 artigos para leitores de todos os níveis. O artigo sobre *phpDocumentor* demonstra uma alternativa para organizar a documentação de seus projetos. Em "*Desenvolvendo em PHP para linha de comando – CLI*", você irá acompanhar valiosas noções sobre a utilização do PHP através da interface de linha de comando. Discutiremos também sobre políticas de segurança em um artigo sobre *SMTP Injection*. Pablo Dall'Oglio participa desta vez com um excelente texto que aborda a *geração de planilhas Excel com PHP-GTK*. Na sequência dois artigos discutem o tema *Ájax e autenticação de webmail através de Java e PHP*. Encerrando a edição, Leandro Schwarz, único autor que publicou em todas as edições da revista, explora a *criação de imagens on-the-fly*.

Também já estamos com muitos planos para o futuro e você não pode ficar fora dessa! Foram muitas conquistas em 2007. Ganhamos credibilidade e repercussão na comunidade de desenvolvedores Web. Mesmo assim, ainda acreditamos que não estamos explorando o potencial máximo da PHP Magazine. Em 2008 iremos focar nossa caminhada na tentativa de obter mais destaque, oferecendo conteúdo de qualidade, apoiando eventos e, como novidade, disponibilizando alguns serviços em nosso portal. Neste ano, com a união e comprometimento de nossa equipe, iremos atuar intensamente no site com o intuito de incluir novas seções e serviços até o lançamento da quinta edição. Aguardem as novidades que estão por vir...

Deixamos aqui o nosso agradecimento a todos os autores que contribuíram nesta edição e nas anteriores. Estejam certos de que vocês fazem parte da história da revista ao longo deste primeiro ano de publicações. E, como sempre, fica aqui nossa convocação para mais um ciclo de atividades. Teremos o imenso prazer em analisar o seu trabalho e disponibilizá-lo em nossa revista. Envie o seu artigo até o dia 25 de abril e participe do próximo exemplar.

A administração da PHP Magazine agradece o apoio de todos os leitores, principalmente daqueles que ficaram preocupados e enviaram mensagens questionando o nosso atraso e perguntando sobre a continuidade do projeto. Aproveitamos para registrar que o projeto continua firme e esperamos atingir o marco dos 10.000 usuários até o lançamento da quinta edição. Para mais esta conquista, contamos com a participação de todos.

Uma boa leitura e até a próxima.

Equipe PHP Magazine

Editores

Flávio Zacharias Fagundes, zach@phpmagazine.com.br
Ricardo Aragão, ricardoaragao@phpmagazine.com.br

Administração

Flávio Zacharias Fagundes, zach@phpmagazine.com.br
Norberto Augusto, augusto@phpmagazine.com.br
Ricardo Aragão, ricardoaragao@phpmagazine.com.br

Comercial

Norberto Augusto

Projeto gráfico

Ricardo Aragão
Flávio Zacharias Fagundes

Revisão técnica

Ricardo Aragão da Silva
Flávio Zacharias Fagundes

Revisão - Língua Portuguesa

Camilla Carvalho, camilla@phpmagazine.com.br



www.phpmagazine.com.br

Comercial

comercial@phpmagazine.com.br

Contato/Suporte

contato@phpmagazine.com.br

Marketing

mkt@phpmagazine.com.br



Fim da PHP Magazine?

Percebi que desde a última edição vocês não lançaram mais nenhum exemplar. Está ocorrendo algum problema? Por favor, se for o caso, comunique não só a mim, mas a todos os leitores, pois até agora as três últimas edições foram legais.

Vagner Virgilio dos Santos :: Guarulhos - SP

Fiquem tranquilos, pois a revista continuará. E em breve teremos novidades em nosso portal.

Novos colaboradores

Acompanho o trabalho de vocês, pois sou a favor da ideologia da linguagem. Ofereço sempre cursos gratuitos de PHP para comunidades, geralmente ao público universitário. Gostaria de saber se existem vagas para a equipe e como faço para me inscrever.

Thiago Silva :: Contagem - MG

Sou graduado em Ciência da Computação e desenvolvo projetos em PHP desde 2005. Quero colocar-me à disposição para colaborar na revista. Tenho habilidades em gestão de projetos, desenvolvimento e análise de sistemas. Sou fluente em inglês e possuo habilidades didáticas.

Gabriel Rambaldi :: Três Corações - MG

No momento, precisamos de colaboradores que incentivem a submissão de artigos e participem do processo de seleção, captando autores.

Parabéns

Parabéns pela iniciativa e viva a comunidade *free!* Uma sugestão: separem uma seção da revista somente para iniciantes. É difícil encontrar um material organizado didaticamente.

Rogério Menta Monici :: Santa Rosa de Viterbo - SP

Sugestão anotada.

Assinaturas

Tenho 18 anos e trabalho com PHP há 2 anos. Achei muito bacana a iniciativa de vocês e gostaria de saber onde encontro a revista na versão impressa e se vocês trabalham com assinaturas.

Bruno Scherer :: Camaquã - RS

Ainda não dispomos de versão impressa, portanto não é necessária nenhuma assinatura da revista, apenas o cadastro no portal para o download das edições.

ENVIE E-MAIL PARA

contato@phpmagazine.com.br

Só serão publicadas mensagens com os seguintes dados do remetente: nome completo, cidade e estado.

Devido às limitações de espaço, a redação se reserva o direito de selecionar e adaptar as mensagens sem alterar o conteúdo.



Mais uma vez será realizado em Porto Alegre/RS, no Centro de Eventos da PUCRS, o 9º Fórum Internacional de Software Livre para debates técnicos e estratégicos sobre o desenvolvimento e o uso do Software Livre. Com atividades para três dias, a programação do evento propõe em sua agenda: Palestras, encontros com Grupos de Usuários, Arena de Programação, Homenageados, Cultura Livre e o Workshop de Software Livre.

As palestras do fisl9.0 serão divididas em doze trilhas: **Desenvolvimento** (Banco de Dados, Ferramentas e Metodologias, Java, Perl, Python, PHP e Ruby), **Kernel, Admin** (Segurança, Redes e Telecom), **Hardware e Sistemas Embarcados, Ecossistema** do Software Livre (Comunidade, Filosofia, Aspectos Sociais e Cultura Livre), Governo e Software Público, **Educação e Inclusão Digital, Jogos** e Multimídia, **Desktop, Casos/Soluções, Negócios** (Produtos e Serviços), e **Tópicos Emergentes**.

Até este momento, estes são os palestrantes confirmados:

- **Randal Schwartz** - Administrador de Sistemas e escritor de diversos livros sobre Perl, Randal é co-fundador da comunidade Perl Mongers e uma das figuras mais respeitadas da comunidade Perl mundial. Atualmente, ele trabalha como consultor para desenvolvimento de software, e é co-host do podcast FLOSS Weekly.
- **Bram Moolenaar** - Engenheiro de Software atualmente trabalhando para o Google, em Zurich, Bram é o criador do editor de textos **VIM**, um dos editores mais utilizados por programadores e administradores de sistemas Unix no mundo inteiro.
- **Ken Coar** - Ken é um dos principais desenvolvedores do Apache, servidor http mais utilizado na internet. Atualmente trabalha para o Linux Technology Center, da IBM.
- **Zaheda Bhorat** - Zaheda ficou conhecida por seu trabalho na comunidade OpenOffice.org, e hoje em dia trabalha como Open Source Program Manager para o Google, tratando de assuntos como Open Standards.
- **Rishab Ghosh** - Diretor da Open Source Initiative. Entre outras coisas, ele é fundador e gerente de edição do jornal First Monday, e Programme Leader para Software Livre na UNU-MERIT. Ele trabalha em diversos projetos de pesquisas sobre Software Livre.
- **Danese Cooper** - Open Source Diva, Intel e Open Source Initiative. Responsável pelo apoio a iniciativas de software livre na Intel.
- **Arnaldo Carvalho de Melo** - Desenvolvedor do kernel Linux com foco em protocolos de rede e um dos fundadores da Conectiva.
- **David Fetter** - Desenvolvedor do projeto PostgreSQL.
- **Jon Maddog Hall** - Diretor-presidente da Linux International.
- **Georgy Berdyshev** - Líder do projeto Hardened Linux.
- **Michael Hanselmann** - Michael Hanselmann trabalha para o Google Switzerland como Administrador de Sistemas, especificamente no desenvolvimento do projeto Ganeti. Ele mantém um projeto de hosting há 5 anos, participou no desenvolvimento do Gentoo por 3 anos e meio, e contribuiu para diferentes projetos de Software Livre, incluindo o Linux Kernel.
- **Alberto Barrionuevo** - Vice-presidente da Foundation for a Free Information Infrastructure (FFII).

PARTICIPE

Está disponível a seção para [inscrições](#), garanta já sua participação.

Você pode apoiar o evento tornando-se [Página Amiga](#), divulgando o evento em seu site e obtendo um pequeno espaço para sua logomarca no site do portal [liberta](#).

Você também poderá obter material para divulgação, saiba mais clicando [aqui](#).

Eventos

A PHP Magazine apóia a iniciativa de eventos como cursos gratuitos, palestras e fóruns. Para divulgar na revista, envie informações sobre o evento para contato@phpmagazine.com.br, que teremos prazer em contribuir com a divulgação.

phpDocumentor

O objetivo deste trabalho é apresentar a ferramenta phpDocumentor e exemplificar algumas das funcionalidade utilizadas para documentação de código PHP. Além dos exemplos de utilização, também será demonstrado o processo de execução do phpDocumentor, o que possibilitará ao leitor gerar sua própria documentação.

À medida que um projeto cresce ou sua equipe aumenta torna-se difícil memorizar todos seus componentes (funções, métodos, classes e até arquivos) e suas funcionalidades, então se faz necessário à documentação do código. Neste ponto, o *phpDocumentor* possibilita ao desenvolvedor e sua equipe acesso à documentação do código de uma forma amigável e acessível.

Analisando o código em busca de palavras chaves e determinados tipos de comentários o *phpDocumentor* transforma estes comentários em páginas HTML ou arquivos PDF entre outros formatos.

1. História

O *phpDocumentor* é uma alternativa para documentação de códigos PHP muito similar ao Javadoc, voltado à linguagem Java. Seu desenvolvimento foi iniciado em meados de 2000 por Joshua Eichorn. Mais tarde, Greg Beaver juntou-se ao projeto. Hoje Joshua e Greg são os principais mantenedores do projeto, embora outros também tenham contribuído. Uma lista completa dos envolvidos no projeto pode ser encontrada em [Autores *phpDocumentor*].

Desde sua criação até hoje o *phpDocumentor* evoluiu muito e atualmente é uma das principais ferramentas utilizadas para a documentação de código escrito em PHP, e que, na sua atual versão (1.4.0), apresenta suporte para PHP5.

2. Estrutura da documentação

Para que o *phpDocumentor* processe os comentários de seu código, estes devem possuir um formato específico. Estes blocos de comentários, também conhecidos como *docBlocks*, tem o seguinte formato:

```
/**
 * Texto ou tag
 */
```

Formato de um docBlock válido.

Um docBlock deverá ser definido imediatamente antes do bloco a ser documentado. A única exceção a esta regra é o docBlock de nível de página. Um docBlock é

definido em três partes que são:

- Descrição;
- Descrição estendida;
- Tags.

A primeira descrição deve ser feita em apenas uma única linha e deve ser utilizada como uma forma resumida e direta de definir o bloco. Esta descrição pode ser terminada com a utilização de uma linha em branco e se for muito grande será utilizada apenas a primeira. Algumas tags HTML são permitidas.

A partir da segunda linha do docBlock é iniciada a descrição estendida do elemento. Nesta parte, o desenvolvedor poderá introduzir quaisquer anotações relevantes ao elemento sem limite de conteúdo. Nesta parte do docBlock são permitidas algumas tags HTML para formatação, alguns exemplos:

- Texto em itálico e/ou negrito;
- Definição de parágrafos e quebras de linha;
- Criação de listas ordenadas;
- Inserção de trechos de código.

No final do docBlock são incluídas as tags que ajudam o phpDocumentor no processamento da documentação. Estas tags e suas funções serão discutidas no próximo tópico.

2.1 Níveis de um docBlock

Existem dois níveis de docBlock, o nível de página e o nível de classe. O nível de página é utilizado para documentar arquivos e geralmente é associado ao paradigma de programação estruturada. Já o de nível de classe é utilizado para documentação de classes e é associado ao paradigma de orientação a objetos.

Na verdade, um docBlock de página é a única forma de agrupar os elementos estruturados em uma unidade lógica de documentação, em pacotes. Mas nada impede a utilização de um docBlock de página em conjunto com um docBlock de classe e, na verdade, em alguns casos se faz necessário utilizar os dois em conjunto.

3. Tags

As *tags* são diretivas inseridas na terceira parte de um *docBlock*. Com exceção das tags inline, que podem ser utilizadas na descrição completa - e são sempre iniciadas pelo caractere "@". Pode-se dividir as *tags* em categorias para um melhor entendimento:

Classificação(C): Agrupam arquivos ou classes. Exemplo: @package

Informação(I): Adicionam informações à documentação. Exemplo: @author

Descrição(D): Descrevem alguma particularidade do bloco documentado. Exemplo: @abstract

Referência(R): Fazem referência a informações que ajudam a compreender um trecho documentado. Exemplo: @see

Além das *tags* padrões o *phpDocumentor* define mais algumas *tags* que são permitidas nos blocos de documentação dentro dos trechos de descrição estendida, estas são chamadas *tags inline*. Estas *tags* permitem incluir mais informações à documentação com um menor esforço por parte do desenvolvedor. As *tags inline* não serão abordadas neste documento. Para mais informações sobre elas acesse "phpDocumentor Inline tags" em [Manual *phpDocumentor*].

3.1 Tags padrão

Na sequência, serão apresentadas algumas das tags padrão mais utilizadas no processo de documentação.

@package: Esta *tag* é utilizada para definir "pacotes" lógicos, onde podem ser incluídos arquivos ou classes. Deve ser utilizada apenas em *docBlocks* de classe e de página. É uma *tag* da categoria C.

@subpackage: Utilizada apenas em conjunto com @package, permite subdividir um agrupamento em sub-pacotes. Assim como @package é um *tag* da categoria C.

Estas *tags* podem ser utilizadas em *docBlocks* de classe e de página.

```
<?php
/**
 * Este é um <i>docBlock<i> de página.
 * A instrução criará o sub-pacote
 * "Tags" no pacote "Exemplo";
 * @package Exemplo
 * @subpackage Tags
 */
?>
```

Exemplo de @package e @subpackage

```
* @author Maykel dos Santos
<maykelsb@yahoo.com.br>
```

Definindo autor de um bloco de código com @author

@copyright: Adiciona informações de *copyright* à documentação. A *tag* é da categoria I.

```
* @copyright Copyright (c) 2007, UpZone
```

Adicionando Copyright

@filesource: Ao encontrar esta *tag* no *docBlock* de página, o *phpDocumentor* adiciona na documentação um link para o código fonte. Pode ser classificado como pertencente à categoria R.

```
* @filesource
```

Criando um link entre a documentação e o código fonte

@deprecated: Usado para indicar que o elemento documentado. É obsoleto e não deve mais ser utilizado. Adicionalmente é possível incluir um comentário após a *tag*. É classificada como categoria I.

```
* @deprecated Utilize f1NaoObsoleto.php;
```

Definindo um elemento como obsoleto

@see: Cria uma referência para outro elemento do código. Podem ser funções, classes, arquivos entre outros. Observe que o link só será criado se o elemento referenciado existir. Outra observação é utilizar referências que ajudem no reconhecimento de um elemento, ou seja, **()** para funções e métodos, **\$** para variáveis, etc. Esta ação melhorará a performance de interpretação desta *tag* da categoria R.

```
* @see f1NaoObsoleto.php
```

Referenciando outros elementos

```
* @todo Apagar este arquivo;
```

Adicionando itens a *ToDo List*

@todo: Esta *tag* cria uma lista de futuras alterações e pode ser utilizada com quase todos os elementos do PHP. Além de listado junto ao elemento documentado, é gerada uma lista global, com todas as ocorrências da *tag*. Isto permite uma consulta geral sobre pendências no sistema. Esta é uma *tag* da categoria I.

Na figura 1 pode-se visualizar a saída gerada pelo *phpDocumentor* com as tags apresentadas até o momento aplicadas a um *docBlock* de página. Observe no menu do lado esquerdo da imagem o link para a *ToDo List*.

@global: Esta *tag* é utilizada para documentação de variáveis globais e de sua utilização através do código.

Pode ser utilizada em conjunto com *tag* @name. Esta é uma *tag* da categoria D.



Figura 1 – Saída do *phpDocumentor*

@name: Ao utilizar esta *tag* para atribuir um apelido a um bloco procedural de código, o *phpDocumentor* criará um *link* de referência a este bloco sempre que ele for documentado futuramente. Esta é uma *tag* da categoria R.

```
/**
 * Esta é uma variável global;
 * @global string $GLOBALS['sGlobal']
 * @name $sGlobal
 */
$GLOBALS['sGlobal'] = 'Variável global';

/**
 * @global string É reinicializada;
 */
function pLimpaGlobal(){
    global $sGlobal;
    $sGlobal = null;
}
```

Utilizando @name e @global

Na figura 2 é mostrado o resultado da documentação do trecho de código acima. Observe o *link* que foi criado na documentação da função para a documentação da variável global. Este *link* é o resultado da utilização da *tag* **@name**.

@param: Definem o tipo e uma descrição para os parâmetros de uma função ou método. É utilizada no seguinte formato: **@param** tipo \$nome descrição. Quando o parâmetro pode ser de mais de um tipo existem duas formas de declaração. A primeira utilizando "|" para separar os tipos ou simplesmente utilizando o valor "mixed". Caso os parâmetros não sejam documentados, o *phpDocumentor* irá processá-los apenas adicionando-os à lista junto à declaração da função ou método. Esta *tag* é da categoria D.

@return: É utilizada para definir o tipo de retorno de uma função ou método, adicionalmente, é possível definir uma descrição para o retorno. Sua sintaxe é: **@return** tipo descrição. Assim como **@param** é possível definir mais de um tipo de retorno para **@return**. Esta *tag* é classificada como D.

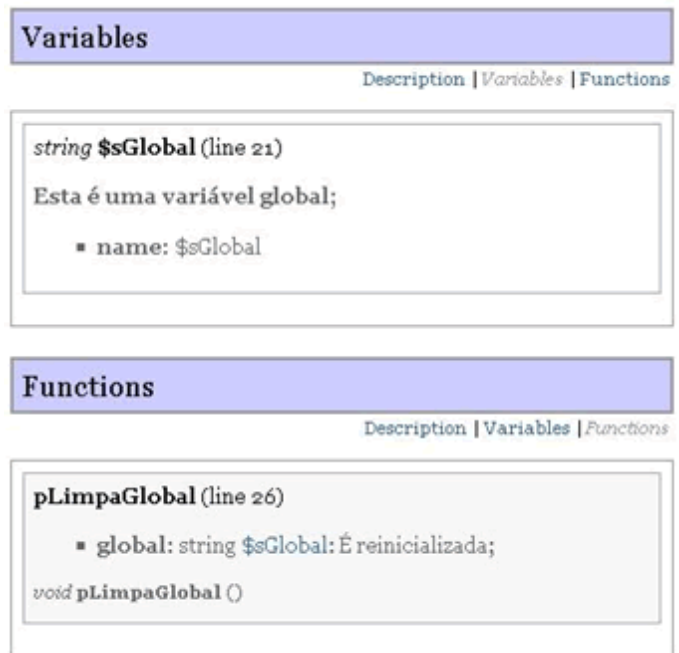


Figura 2 – Documentação gerada por **@global** e **@name**

```
/**
 * Imprime tipo e conteúdo do parâmetro;
 * @param string|int|boolean $iParam1
 * Descrição;
 * @param mixed $iParam2 Descrição;
 * @return string Descrição;
 */
function fImprime($iParam1, $iParam2){
    var_dump($iParam1);
    var_dump($iParam2);
    return (string)$iParam1
        . (string)$iParam2;
}
```

Documentando parâmetros e retorno de uma função ou método

@var: A documentação dos atributos de uma classe é feita através desta *tag*. A sintaxe utilizada é: **@var** tipo descrição. Adicionalmente, pode-se utilizar as descrições curtas e completas junto a *tag* **@var**. Esta *tag* é classificada como categoria D.

@abstract: Classifica uma classe, método ou atributo como abstrato. Ao processar um bloco de código PHP com a diretiva *abstract* o *phpDocumentor* já o classifica como abstrato, o que dispensa a utilização desta *tag*. Pertence à categoria D.

@final: Documenta um método final, ou seja, um método que não pode ser sobrescrito por uma classe filha. Também é uma *tag* que é processada automaticamente caso o código já possua a diretiva *final*. Esta é uma *tag* da categoria D.

@static: Define métodos e atributos como estáticos em classes. Da mesma forma que **@abstract** e **@final** esta *tag* também é opcional, pois uma vez encontrada a palavra chave, o elemento será documentado como *static*. Também é uma *tag* da categoria D.


```

/**
 * Uma classe abstrata;
 * @package Exemplo
 * @subpackage Tags
 * @abstract
 */
abstract class cExemplo{
    /**
     * Documentação de atributos;
     * @var string $sNome Nome do exemplo;
     * @static
     * @final
     */
    final static public $sNome;
}

```

Documentação de classes e atributos com @abstract, @var, @static e @final

Estas não são todas as tags padrão do *phpDocumentor*. Para uma lista completa e mais detalhes, visite a referência [Manual phpDocumentor].

4. Instalação e execução

A última versão do *phpDocumentor* está disponível em [Download phpDocumentor] e existem versões disponíveis para Windows e Linux. Para executar o *phpDocumentor* é necessário a existência de uma versão do PHP maior que 4.1.0 e, para usar a *interface web* um servidor rodando o PHP. Este artigo aborda apenas a execução via linha de comando.

4.1 Execução via linha de comando

Esta é a maneira mais simples de execução e a que necessita de menos recursos. Considerando que o php esteja no seu PATH, para executar o *phpDocumentor* com a configuração padrão basta executar o seguinte comando:

```
php "path/to/phpdoc" -t "saida" -o
HTML:default:default -d "a/processar"
```

phpDocumentor via linha de comando

Por padrão, *phpdoc.bat* está na raiz do arquivo compactado.

O parâmetro *-o* define a saída da documentação gerada e está dividida da seguinte forma: Tipo de saída:Nome do conversor:Nome do template.

Para tipo de saída existem os seguintes valores

- HTML
- XML
- PDF
- CHM¹

Entre os diversos tipos de conversores para HTML estão:

- default;
- earthli;
- l0l33t
- phpdoc.de;

Considerações finais

Analisando a documentação gerada pelo *phpDocumentor* é possível ver a diferença entre ler a documentação direto no código fonte e ler a documentação apresentada com uma saída amigável e com *hyperlinks* para fácil acesso a outros itens documentados.

Além da saída amigável e em diversos formatos, que satisfazem diversas situações, o *phpDocumentor* apresenta um conjunto de *tags* muito simples, o que permite um rápido aprendizado a qualquer desenvolvedor.

Outra vantagem deste tipo de documentação é o fato da documentação ser realizada junto ao elemento e pode ser atualizada no mesmo momento em que este é atualizado. Esta facilidade evita a necessidade do desenvolvedor procurar por outros arquivos para atualizar a documentação, o que não seria muito produtivo.

Não se esqueça que o objetivo deste artigo é apenas apresentar ao leitor esta ferramenta de documentação. Com essa informações o leitor poderá aproveitar ainda mais suas diversas funcionalidades, muitas das quais não foram apresentadas aqui.

Referências e links sugeridos

[*phpDocumentor*] – <http://www.phpdoc.org>

[Manual *phpDocumentor*] – <http://manual.phpdoc.org>

[Download *phpDocumentor*] – http://sourceforge.net/project/showfiles.php?group_id=11194

[Autores *phpDocumentor*] – <http://www.phpdoc.org/notes/Authors>

Os arquivos gerados são apenas a base do CHM.

Maykel dos Santos Braz maykelsb@yahoo.com.br
Bacharel em Engenharia de computação pela
Universidade do Estado de Minas Gerais (UEMG).



Desenvolvendo em PHP para linha de comando – CLI

O PHP foi projetado para trabalhar em conjunto com um servidor HTTP, porém sua facilidade de uso e sua gama de recursos foram vistos como aplicáveis à *interface* da linha de comando. Assim, a partir da versão 4.3.0, o PHP CLI passou a fazer parte da instalação *default* do interpretador. Com o CLI podemos fazer aplicativos para serem executados no terminal, tais como Shell Script ou Perl. A possibilidade de executar PHP na linha de comando potencializa a integração com as aplicações Web.

O PHP tem seu nome reconhecido pela sua eficiência quando aplicado para desenvolvimento de soluções Web. Entretanto, assim como Perl pode ser usado em aplicações na linha de comando.

A vantagem do uso do PHP ao invés de outra forma de execução no Shell está na disponibilidade de documentação e na comodidade de não precisar aprender uma outra linguagem para executar tarefas relativamente complexas fora do ambiente Web.

Usando-se desta característica pouco difundida, o programador pode criar uma série de utilitários com funcionalidades que não eram possíveis ou viáveis usando PHP em conjunto com o Apache ou outro servidor HTTP.

Este artigo expõe algumas diferenças de executar PHP na linha de comando, bem como mostra exemplos básicos da entrada de dados a uma aplicação. Tendo isso claro, o programador já tem tudo que precisa para desenvolver as mais variadas soluções.

1. Diferenças básicas do PHP CLI

Programar para CLI não difere muito de programar para Web. Mesmo porque a versão CLI foi baseada na CGI, aquela quando o PHP é executado através da requisição feita pelo Apache. Pode-se usar de todos os recursos que estamos acostumados a usar como classes de terceiros, módulos do PHP, entre outros.

O que não temos disponível – e não faz sentido termos – é a manipulação de cabeçalhos HTML, logo, não conseguimos redirecionar, pegar o valor de \$_POST ou \$_GET, \$_COOKIE ou se imprimirmos algum HTML na tela, ele será mostrado como texto puro. Terminal não é um *browser*. Por isso que, quando programamos para terminal, os erros são exibidos em texto puro. O CLI não envia os cabeçalhos padrões que o PHP envia ao *browser*.

A característica que mais precisa estar clara é que

agora o PHP é independente do servidor Web, ou seja, o processo é executado sobre o usuário que iniciou a execução. Cada execução tem seu próprio PID, e não há tempo limite para execução do *script*.

Observe que na figura 1 um *script* que dispara *ping* contra uma máquina é executado. Este *script*, enquanto não interrompido manualmente (ctrl+c), continuará executando e não acusará erro por exceder o tempo limite.

```
ss:/phpshell # php5 ping.php
```

Figura 1 - Executando o *script* de ping

Na figura 2, observe que o PID 3776 está sendo executado pelo “root”, mostrando que cada *script* executado tem seu próprio PID, o que não ocorre quando o PHP é executado para Web, onde todas as execuções compartilham o mesmo processo.

```
helton@ss:/phpshell> ps -fa |grep php
root      3776  3605  0 17:54 pts/1    00:00:00 php5 ping.php
helton    3789  3292  0 17:55 pts/0    00:00:00 grep php
helton@ss:/phpshell>
```

Figura 2 - Processo da execução do *script* de ping

Na Web seria inviável. Imagine um site com 800 usuários *on-line*. Já executando um *script* pela linha de comando, essa característica muda. Exclusivamente para CLI, o PHP possui um módulo, o “pcntl”, justamente para controle de processos.

Em tese, a execução de PHP para linha de comando também tende a ser mais rápida, pois a entrada do usuário vai diretamente para o interpretador PHP e é devolvida. Enquanto na Web a entrada do usuário é para o servidor, este para o PHP, que devolve para o servidor (de HTTP), e este para o usuário.

2. Como executar *scripts* pela CLI

Há três formas básicas de se fazer isso em ambiente Linux.

```
[helton@localhost phpshell]$ php -r 'echo(date("d/m/Y")."\n");'  
24/11/2007  
[helton@localhost phpshell]$ █
```

Figura 3 - Executando o código de uma linha

Assim como exibido na figura 3, chamamos o executável do PHP informando que deve executar o que está entre aspas simples. Isso é feito com o parâmetro "-r". Note que neste caso não é necessário as *tags* de abertura e fechamento.

Diferentes distribuições possuem nomes diferentes para o executável do PHP. Na figura 3, usávamos Mandriva Linux 2008, e PHP5 e o nome do executável era apenas "php". Nas figuras seguintes uso PHP5, executo em Open Suse 10.1. Verifique qual se aplica para sua máquina.

A outra maneira é informar ao interpretador PHP que deve executar o que há num arquivo. Este arquivo segue as mesmas regras do PHP para Web, ou seja, contém as *tags* de abertura e fechamento de PHP, assim como ";" ao final de cada instrução. Na figura 4 tem-se o exemplo mais básico de PHP para linha de commando.

```
<?php  
print("meu primeiro script\n");  
?>
```

Figura 4 - Primeiro *script* PHP para CLI

Executando o código da figura 4, obtemos o resultado da figura 5.

```
helton@ss:/phpshell> php5 01.php  
meu primeiro script  
helton@ss:/phpshell> █
```

Figura 5 - Executando o primeiro *script* para CLI

A maneira mais prática, entretanto, para executar um *script*, é descobrir onde está o executável do PHP (normalmente em /usr/bin/php5) e na primeira linha do *script* adicionar "#!/usr/bin/php5", em seguida tornar este arquivo PHP um executável (chmod +x script.php), e executá-lo simplesmente (./script.php).

Em ambiente Windows, também é possível executar seus *scripts*, abra o *prompt* vá até o diretório onde está seu interpretador PHP, e digite "php caminho_script". Para facilitar, podemos fazer um "bat". Infelizmente, não há como dizer que um arquivo texto é um executável, e simplesmente executá-lo como em Linux.

3. Entrada de dados

A entrada de dados pode se dar de duas formas: através de parâmetros ou da leitura do "STDIN".

Assim como C, o PHP CLI possui as variáveis "\$argc" e "\$argv". A primeira tem um número inteiro indicativo

de quantos parâmetros foram passados para o arquivo. \$argv contém o valor desses parâmetros, é um *array*, que tem no índice 0 o nome do próprio arquivo que está sendo executado. Os demais parâmetros são os passados pelo usuário.

```
<?php  
echo($argc."\n");  
print_r($argv);  
?>
```

Figura 6 - Exibindo quantos e quais parâmetros

```
helton@ss:/phpshell> php5 02.php localhost root SenhaSecreta  
4  
Array  
(  
    [0] => 02.php  
    [1] => localhost  
    [2] => root  
    [3] => SenhaSecreta  
)  
helton@ss:/phpshell> █
```

Figura 7 - Passando parâmetros para o *script*

O resultado da execução do *script* da figura 6 pode ser visto na figura 7.

Sabendo desta possibilidade de entrada de dados, alguns já ficam pensando nas muitas aplicabilidades, uma vez que se têm todos os recursos do PHP disponíveis.

Passaremos agora para um meio de entrada de dados mais interativa, como estamos acostumados nos programas escritos em C ou outra linguagem para o ambiente texto. Vamos ler a entrada de teclado, do arquivo STDIN (standard input).

```
#!/usr/bin/php5  
<?php  
echo("Digite seu nome: ");  
$nome=trim(fgets(STDIN));  
echo($nome."\n");  
?>
```

Figura 8 - lendo o STDIN

Como visto, trata-se de um arquivo PHP em que na primeira linha foi incluído o caminho do executável, em seguida uma variável recebeu, sem espaços ou *enter*, a leitura do arquivo STDIN. A função "trim" é usada, pois na entrada de texto, o *enter* é lido como último caractere. O último caractere é uma quebra de linha.

Executando o *script* acima (que é um executável – chmod +x 04.php) tem-se o que é apresentado na figura 9.

```
ss:/phpshell # ./04.php  
Digite seu nome: Helton Eduardo Ritter  
Helton Eduardo Ritter  
ss:/phpshell # █
```

Figura 9 - Lendo e exibindo a entrada de teclado

Até então, acredito que temos os recursos básicos

para desenvolver qualquer *script* para linha de comando. Não precisamos nos preocupar com estouro de memória em virtude da entrada de uma *string* muito grande, mas as nossas aplicações devem estar preparadas para um tamanho fixo e máximo dessa *string*.

Podemos limitar quanto queremos ler do STDIN, já na leitura através do "fgets", informando um parâmetro opcional. Neste caso estamos lendo apenas os 10 primeiros caracteres.

```
#!/usr/bin/php5
<?php
echo("Digite seu nome: ");
$nome=trim(fgets(STDIN,10));
echo($nome."\n");
?>
```

Figura 10 - Lendo 10 caracteres de STDIN

Considerações finais

O PHP foi projetado para Web, e é nesta função que apresenta os resultados mais interessantes. Entretanto, podemos usar mais esta linguagem para resolver alguns problemas fora da Web. Simplesmente há situações em que um aplicativo Web não satisfaz. Prova disso é o PHP

-GTK que busca maximizar as soluções em que podemos aplicar PHP.

A linha de comando, por tempos sufocada e dita como "antiga", hoje é um recurso "novo" no Windows Server 2008, através do "Power Shell". Em ambiente Linux, não há porque discutir a aplicabilidade da *interface* de linha de comando.

O PHP mostra-se uma solução de fácil aplicação para automatizar e facilitar uma série de tarefas e tudo isso com o mesmo modo de programar que já estamos acostumados.

Programador PHP, o que acha de conversar com o administrador de redes da sua empresa? Tenho certeza que surgirão soluções muito boas!

Referências e links sugeridos

GUTMANS, Andi; BAKKEN, Stig Saether; RETHANS, Derick. **PHP 5 Power Programming**. Indianapolis, Prentice Hall: 2005.

<http://www.vivaolinux.com.br/artigos/verArtigo.php?codigo=347>

Helton Eduardo Ritter - heltonritter@gmail.com

Bacharelando em Sistemas de Informação pela Sociedade Educacional Três de Maio – SETREM, pela qual também é Técnico em informática formado em 2006 e funcionário desde julho do mesmo ano.

Moderador PHP do fórum ScriptBrasil, Helton é também colunista do Imasters e faz trabalhos como free-lancer.



Amplie os horizontes da sua empresa

Divulgue sua empresa e produtos para a comunidade PHP do Brasil.
Conheça os planos para anunciar no portal e na revista.
Solicite um orçamento através de comercial@phpmagazine.com.br.

Ataque de sobrecarga utilizando SMTP Injection. O que é, como é e como se prevenir

Este artigo apresenta de forma objetiva o funcionamento do SMTP *Injection* e como ele pode ser utilizado para causar um ataque de sobrecarga, como isto pode vir a afetar o servidor, como um programador deve codificar e como um administrador deve agir para evitar problemas.

Quando fui atacado pela primeira vez, pensei que a responsabilidade de resolver o problema era do técnico que gerenciava a hospedagem, mas com o passar do tempo compreendi melhor como funciona esta parte do PHP e concluí que somos todos responsáveis, tanto nós programadores quanto os administradores de sistema. Hoje sei que no caso do *Injection* a responsabilidade é muito mais do programador.

Este texto se destina àqueles que já sofreram ataque de *spammers* e nunca tiveram tempo para entender por completo como tudo aconteceu. Aqui tentamos explicar, tanto para programadores avançados quanto para iniciantes, como e por que acontece e o que fazer para evitar um novo ataque.

1. O que é ataque de carga?

Servidores estão disponíveis exatamente para isto, nós servir. Sempre que um computador fica muito ocupado atendendo a diversas pessoas ao mesmo tempo dizemos que ele está carregado de trabalho. É comum e é importante frisar que tal carga de trabalho deve ser prevista por um técnico que irá dimensionar as necessidades de *hardware* de acordo com o *software* e a demanda de serviço que a máquina irá atender, vez por outra, como um humano que tem muito trabalho, ele está sobre-carregado, o que é perfeitamente normal, um bom técnico prevê este tipo de situação. O problema é que quando ficamos sobrecarregados de trabalho começamos a ficar estressados e vez por outra cometemos erros em algumas situações, assim também é a operação de uma máquina sobrecarregada, especialmente por um período muito prolongado de prazo.

Ataque de sobrecarga é um tipo de ataque de DoS (Denial Of Service) que é uma técnica de contenção, pois eles fazem com que o servidor negue comunicação

com clientes, digamos que eu queira atrasar o recebimento de um email, ou a leitura de uma notícia, eu posso me utilizar de um ataque de DoS para conter o servidor de distribuição daquela informação, atrasando assim o cliente de obtê-la. Em verdade a maior parte dos ataques de DoS são ataques de carga.

Na maior parte dos casos um ataque de carga geralmente conta com certas vantagens em relação a outros, nem sempre temos um bom técnico dimensionando o sistema ou gerenciando o mesmo e nem sempre temos um projeto bem implementado. Em sua maioria um ataque de carga retira o sistema do ar por um período curto de tempo, é muito utilizado em situações em que temos uma concorrência desleal. Quem mais sofre por não corrigir este tipo de ataque são pequenas empresas de hospedagem que geralmente não tem uma política de defesa sobre estas situações.

2. O que é SMTP e como ele funciona?

SMTP é uma convenção descrita no RFC 2881. É um protocolo de comunicação entre um cliente que envia e-mail e um servidor que recebe e-mail. Um protocolo basicamente é um mecanismo de troca de mensagens. É como se nós padronizássemos nossa conversa e sempre que eu falo "Olá" você deve responder "Olá", pois assim eu vou saber que estou iniciando uma conversa com você, não finalizando. Dizemos, então, que temos um protocolo de comunicação, uma "linguagem" de comunicação em que você entende o que eu estou falando e eu sei que você está me escutando de maneira adequada. O protocolo SMTP descreve como um servidor de e-mail deve receber uma mensagem de e-mail. Abaixo (Figura 1) temos um exemplo de comunicação entre um cliente e um servidor. O protocolo está a partir da linha 9. Linhas como 11, 13, 21 e 38 são mensagens suplementares do programa que efetuou o envio. O SMTP está em

linhas como a linha 29, que envia o comando MAIL TO para o servidor, e na linha 31 em que o servidor responde, informando que esta de acordo em receber uma mensagem para o destinatário informado.

01 Domínio destino: tario.com	34 W: DATA
02 Identificando MTAs disponíveis	35 S: 4 bytes
03 Servidor selecionado: mx.tario.com	36 R: 354 go ahead
04 Conexão socket estabelecida	37 S: 12 bytes
05 R: 220 mx01.tario.com ESMTP	38 T: 0.082557916641235 sec
06 S: 35 bytes	39
07 T: 0.41055202484131 sec	40 W: MIME-Version: 1.0
08	41 X-SenderEngineVersion: 1.0
09 W: EHLO tario.com	42 X-SenderEngineName: Ideias Pontual - iMail
10 S: 21 bytes	43 Return-path: NomeFrom <script@site.com>
11 R: 250-mx01.tario.com	44 Content-Type: text/plain;
12 S: 29 bytes	45 charset=ISO-8859-1
13 R: 250-PIPELINING	46 Reply-to: NomeFrom <script@site.com>
14 S: 14 bytes	47 Date: Sat, 21 Jul 2007 13:53:15 -0300
15 R: 250-8BITMIME	48 From: NomeFrom <script@site.com>
16 S: 12 bytes	49 To: NomeTo <destin@tario.com>
17 R: 250-SIZE 30000000	50 Subject: Ataque de spammers
18 S: 17 bytes	51
19 R: 250 AUTH LOGIN PLAIN CRAM-MD5	52 Atacam nosso site!
20 S: 29 bytes	53 .
21 T: 0.053061962127686 sec	54 S: 379 bytes
22	55 R: 250 ok 1185036797 qp 31968
23 W: MAIL FROM: <script@site.com>	56 S: 26 bytes
24 S: 28 bytes	57 T: 0.20024585723877 sec
25 R: 250 ok	58
26 S: 6 bytes	59 W: QUIT
27 T: 0.81833100318909 sec	60 S: 4 bytes
28	61 R: 221 mx01.tario.com
29 W: RCPT TO: <destin@tario.com>	62 S: 29 bytes
30 S: 36 bytes	63 T: 0.044206857681274 sec
31 R: 250 ok	64
32 S: 6 bytes	65 Tempo total de envio: 1.6655600070953 sec
33 T: 0.043802976608276 sec	

W=comando; R=resposta; S=Tamanho do texto; T=Milisegundos entre a escrita e a leitura

Figura 1: Execução de envio de um e-mail

Dentre estes comandos temos o comando DATA, que recebe a parte principal da mensagem, seu corpo. O comando DATA é separado em diversas áreas podendo ter uma forma diferente de acordo com as necessidades da mensagem. Na Figura 1 temos um exemplo do comando DATA em uma mensagem de texto proveniente do email destin@tario.com.br sendo encaminhada a partir de script@site.com.br. Entre as linhas 40 e 50 temos o cabeçalho da mensagem. A linha 51 é uma linha em branco que separa o cabeçalho do e-mail e a mensagem em si. Já na linha 52 temos a mensagem, que poderia se estender por diversas linhas e na linha 53 informamos que a mensagem foi finalizada.

Veja que temos no cabeçalho diversos comandos conhecidos também como *Headers* da mensagem. Tais *headers* possibilitam que os softwares de envio e recepção de e-mail sejam mais práticos de serem construídos. Eles não fazem parte do protocolo SMTP, porém ampliam sua funcionalidade. Estes cabeçalhos implementam instruções do RFC 2882 e com tais *headers* o servidor pode identificar se a mensagem está sendo recebida mais de uma vez, comparando tanto sua data de criação quanto seu criador, bem como saber que deve encaminhar cópias da mensagem (BCC e CC) ou que o e-mail possui arquivos em anexo.

Dentre os comandos do protocolo SMTP o mais importante é o comando DATA, que recebe a parte principal da mensagem, seu corpo, ou seja, a mensagem em si. Como podemos perceber na Figura 1 o comando DATA é separado em diversas áreas e pode ter forma diferente de acordo com o formato da mensagem. Na linha 34 encaminhamos ao servidor a requisição de início de envio da mensagem (Comando DATA) e na linha 36 o sistema diz estar preparado para iniciar o envio. O programa, então, na linha 40 inicia o envio da mensagem. Esta mensagem começa com o cabeçalho que vai até a linha

50. No cabeçalho podemos informar diversos comandos utilizados pelo cliente para "renderizar" (apresentar em tela) o e-mail. A linha 51 é apenas uma linha em branco. A primeira linha em branco no corpo da mensagem serve para informar o *e-mail client* (*outlook, thunderbird, webmail*) que o *header* finalizou e o resto do texto no corpo da mensagem é a informação que deve ser encaminhada. Na linha 52, temos o e-mail que nada mais é do que uma frase em texto convencional. Finalmente, na linha 53 temos o final da mensagem. Uma linha em branco com um ponto sem nenhum outro caractere após o ponto. A linha 55 é a resposta do servidor dizendo que aceita receber a mensagem e que eu posso prosseguir com o protocolo.

Você pode saber mais sobre as funcionalidades do SMTP e de *Mail headers* "googleando" na internet. Existem diversas tecnologias que estão se popularizando e que dificultam o ataque de *Injection*, como o SPF. Se puder ler em inglês, um bom começo para o aprendizado é buscar os RFCs, em especial o 822 que fala como uma mensagem deve ser construída para a Internet, o 2821 que fala especificamente sobre o SMTP e o 2822 que fala amplia as funcionalidades do 822, adicionando mais informações sobre o conteúdo das mensagens.

3. O que é ataque de SMTP Injection

Um SMTP *Injection* faz uso maligno de uma convenção benigna. Diz-se *Injection* quando injetamos em um código extra no programa. É uma técnica de programação muito útil quando utilizando comandos como *eval* (em PHP) que está presente na maioria das linguagens de programação modernas. De uns tempos para cá esta técnica vem sendo utilizada para a má índole, injetando em código puro de um programa instruções que o fazem agir de uma maneira indevida. É popular para SQL, *Javascript*, graças ao AJAX, e SMTP. Cada uma destas funções tem diversas variações, mas basicamente a função principal é dar dor de cabeça ao administrador do sistema.

Lembra de nossa Figura 1? Ela apresenta uma mensagem convencional que segundo a linha 49 está sendo entregue para destin@tario.com em um ataque de SMTP *Injection*. O atacante adicionaria diversas instruções no sistema como os comandos com cópia para (CC:) e qualquer outro comando que possa ser processado pelo *header* da mensagem. Isto é permitido pelo PHP, pois ele utiliza o programa de envio de email do *shell* para encaminhar a mensagem. Segundo a estrutura de código da função *mail* do PHP, qualquer instruções repassada no parâmetro *body* da função se transformará em texto da mensagem. Os demais parâmetros são utilizados para controlar a mensagem, ou seja, se adicionarmos um texto como BCC na instrução TO, ela será processada normalmente quando o PHP encaminhar esta instrução ao *shell* do sistema operacional.

Para fazer um exemplo imagine que nossa Figura 1 foi encaminhada no formulário da Figura2 abaixo. Este tipo de formulário é um prato cheio para o atacante, não precisa nem estar no domínio da vítima, desde que este-

ja na mesma rede e utilize os mesmos recursos da vítima.

Ele deixa todos os campos, o De (From), o Para (To) e o Assunto (Subject) abertos para que o atacante possa escrever qualquer coisa. A ilustração abaixo está com os campos preenchidos com o conteúdo utilizado para gerar a transmissão relatada na Figura 1, porém se no campo De colocarmos a seguinte sequência de caracteres `haxor@attack.com%0ASubject:Mwahahaha%0ABcc:target@nothappy.com%0AContent-Type:multipart/mixed;%20boundary=frog;%0A--frog%0AContent-Type:text/html%0A%0A<u>HTML%20Message.</u>%0A%0A--frog%0AContent-Type:text/html;name=Security.html;%0AContent-Transfer-Encoding:8bit%0AContent-Disposition:attachment%0A%0A<u>HTML%20File</u>%0A%0A--frog--%0A` estes caracteres irão para o corpo da mensagem gerando uma transmissão que está disponível na Figura 3.

Agora observaremos as linhas. Entre as linhas 40 e 57 vemos que existem comandos extras como To (Linha 53) e BCC (linha 55) que vai encaminhar a mensagem para outros destinatários além do inicial (linha 49). O caractere representado pela sequência `%0A` está em hexadecimal e, depois que o PHP processa esta linha, ele transforma este valor em um caractere válido de nova linha (*enter*, ou *new line*, ou ainda *line feed*), fazendo com que tudo o que está escrito após esta sequência seja interpretada como nova linha. É por isto que na Figura 3 o `haxor@attack.com` está em uma linha e o *Subject* está em outra. Como o PHP não restringe o que está sendo passado para o shell na linha de comando, podemos passar o que desejarmos nestes campos, desde que coloquemos o que se espera do campo por primeiro. Por exemplo: no caso do nome colocamos qualquer palavra, no caso do e-mail colocamos um endereço de e-mail e no campo assunto colocamos qualquer palavra. O PHP simplesmente irá receber o valor e repassá-lo para a linha de comando e qualquer caractere especial que queira utilizar como o *tab*, basta passá-lo como hexadecimal.

Esta implementação do PHP foi criada dessa forma para que o programador pudesse ter maior liberdade ao utilizar as funcionalidades de protocolos disponíveis como instruções para clientes de e-mail. De certa forma isto faz com que o SMTP *Injection* possa ser visto também como um *exploit* do PHP, posto que é uma exploração de uma suposta falha no sistema. Como não é uma falha, mas uma forma indevida de utilizar a função *mail* do PHP, um SMTP *Injection* não se qualifica exatamente como um *exploit*, porém tem o mesmo princípio.

SMTP *Injection* é o método preferido de *spammers* para encaminhar e-mails contendo vírus, mesmo que a vítima (o servidor que está processando as requisições) possua mecanismos como SPF, eles serão vistos como e-mails dignos e recebidos por outros sistemas como mensagens verdadeiras.

Fale conosco!

De: Para:

E-Mail: E-Mail:

Assunto:

Mensagem:

Enviar

Figura 2: Formulário de envio de e-mail

```
01 Dono de destino: tario.com
02 Identificando MTAs disponíveis
03 Servidor selecionado: mx.tario.com
04 Conexão socket estabelecida!
05 R: 220 mx01.tario.com ESMTP
06 S: 35 bytes
07 T: 0.41055202484131 sec
08
09 W: EHLO lucremails.com
10 S: 21 bytes
11 R: 250 mx01.tario.com
12 S: 29 bytes
13 R: 250 PIPELINING
14 S: 14 bytes
15 R: 250 8BITMIME
16 S: 12 bytes
17 R: 250 SIZE 30000000
18 S: 17 bytes
19 R: 250 AUTH LOGIN PLAIN CRAM-MD5
20 S: 29 bytes
21 T: 0.053061962127686 sec
22
23 W: MAIL FROM: <script@site.com>
24 S: 28 bytes
25 R: 250 ok
26 S: 6 bytes
27 T: 0.81833100318909 sec
28
29 W: RCPT TO: <destin@tario.com>
30 S: 36 bytes
31 R: 250 ok
32 S: 6 bytes
33 T: 0.043802976608276 sec
34 W: DATA
35 S: 4 bytes
36 R: 354 go ahead
37 S: 12 bytes
38 T: 0.082357916641235 sec
39
40 W: MIME-Version: 1.0
41 X-SenderEngineVersion: 1.0
42 X-SenderEngineName: Idéias Pontual - IPta
43 Return-path: NomeFrom <script@site.com>
44 Content-Type: text/plain;
45 charset=ISO-8859-1
46 Reply-to: NomeFrom <script@site.com>
47 Date: Sat, 21 Jul 2007 13:53:15 -0300
48 From: NomeFrom <script@site.com>
49 To: NomeTo <destin@tario.com>
50 Subject: Teste de envio texto!
51 From: site@site.com
52 Subject: Ataque de spammers
53 To: haxor@attia.ck.com
54 Subject: Mwahaha
55 Bcc: target@nothappy.com
56 Content-Type: multipart/mixed;
57 boundary=frog;
58 --frog
59 Content-Type: text/html
60
61 <u>HTML Message.</u>
62
63 --frog--
64 Content-
65 type: text/html; name=Security.html;
66 Content-Transfer-Encoding: 8bit
67 Content-Disposition: attachment
68
69 <u>HTML File</u>
70
71 --frog--
72 Atacaram nosso site!
73
74 S: 379 bytes
75 R: 250 ok 1185036797 qp 31968
76 S: 26 bytes
77 T: 0.20024585723877 sec
78
79 W: QUIT
80 S: 4 bytes
81 R: 221 mx01.tario.com
82 S: 29 bytes
83 T: 0.044205857681274 sec
84
85 Tempo total de envio: 1.6655600070953
86 sec
```

W=comando; R=resposta; S=Tamanho do texto; T=Milsegundos entre a escrita e a leitura

Figura 3: Mensagem com comandos "injetados"

Em minha opinião, ataque de SMTP *Injection* é uma das piores pragas que ocorrem no mundo digital. Ao iniciar um ataque de SMTP *Injection*, a vítima passa a mandar diversas mensagens indesejadas (spams) e, um tempo depois, servidores passam a interpretá-lo como *spammer*. Tais tipos de *spammer* muitas vezes levam semanas para serem retirados.

Existem outras utilidades para um ataque de SMTP *Injection*, como por exemplo denegrir a marca e a imagem da vítima ou encaminhar vírus que podem ser repassados por texto – o atacante pode unir instruções repassadas no campo De com instruções repassadas no campo Mensagem em nosso formulário da Figura 2 e encaminhar o arquivo executável apenas transformando-o em hexadecimal, como o *new line* da Figura 3. Porém, nosso tema é o ataque de carga então vamos continuar com o assunto.

4. Como executar o ataque em massa

Para efetuarmos um ataque de carga, utilizamos um programa para executar. É muito mais efetivo para encaminhar centenas ou milhares de mensagens do que ficar preenchendo repetidamente o formulário atacado. Porém, além de utilizarmos um sistema automatizado de ataque, o mais indicado é adicionarmos no comando instruções de CC e BCC que encaminhem a mensagem para outras redes, de forma a forçar o servidor a se conectar com o maior número possível de outros servidores, mantendo-o ocupado o suficiente para negar a inclusão de mais mensagens na fila de entrega.

Atualmente, estes programas são bem simples de serem confeccionados. O próprio PHP dispõe de diversas metodologias para construir um sistema eficiente de ataque com bibliotecas como a *clib* e a própria função **fo-pen**. E para efetuar a função de lotar a fila, basta que tais programas fiquem em execução postando o conteúdo do formulário para o *script* que deve receber tais informações. Não precisamos nem passar pelo formulário e pela sua checagem em *Javascript*, basta postar as informações.

Para evitar que o sistema seja identificado pelo administrador, podemos ainda fazer um ataque distribuído a partir de três ou quatro máquinas, enchendo rapidamente a fila três ou quatro vezes mais rápido.

5. Como evitar por meio de *scripts*

Para evitar por meio de código é muito simples, basta fazermos o que o PHP sugere que façamos: filtrar o que recebemos de cada um dos campos disponíveis para preenchimento no formulário. Basta verificar se um deles colocou caracteres como o 0x0A, ou 0x0D, ou ainda 0x09, ou se colocou mais de uma vez o caractere @ (0x40). Se isto ocorrer, então, você está sendo atacado. Você não precisa, por exemplo, verificar o que está sendo incluído no campo da mensagem, se o atacante estiver utilizando esta parte do formulário, ele certamente terá que utilizar o outro campo para injetar o início das instruções que ele pretende utilizar. Sendo assim, o campo da mensagem pode ser poupado de verificação, se você verificar os campos De e Para.

A Figura 4 apresenta um exemplo de como verificar se o \$_POST contém instruções que indicam um ataque por meio de expressões regulares, e a Figura 5 apresenta a mesma sequência por meio do *strpos*.

```
$Validacao = "/(%0A|%0D|\\n+|\\v+)(content-type:|to:|cc:|bcc:)/i";
$bFormosAtacados = false;

foreach ($_POST AS $sValue) {
    if (preg_match($Validacao, $sValue) != 0) {
        $bFormosAtacados = true;
    }
}

if ($bFormosAtacados) {
    echo "Ataque sendo feito!";
    exit(0);
}
```

Figura 4: Exemplo de checagem de código com a função *preg_match*

```
$Validacao = array(
    "%0A",
    "\n",
    "\r",
    "%0D",
    "content-type",
    "to:",
    "cc:",
    "bcc:"
);
$bFormosAtacados = false;

foreach ($_POST AS $sPValue) {
    foreach ($Validacao AS $sVValue) {
        if (strpos($sPValue, $sVValue) != false) {
            $bFormosAtacados = true;
        }
    }
}

if ($bFormosAtacados) {
    echo "Ataque sendo feito!";
    exit(0);
}
```

Figura 5: Exemplo de checagem de código com a função *strpos*

6. Como evitar por meio de administração do servidor

Segundo Juliano Simões, Gerente de Tecnologia do Central Server, existem duas abordagens disponíveis ao administrador para reduzir o risco de ocorrência de SMTP *Injection* no servidor: uma pró-ativa e outra reativa.

As principais ações pró-ativas envolvem conscientizar os programadores sobre o problema, divulgar as formas de proteção de código disponíveis e auditar os *scripts* instalados nos websites. Para a auditoria, o recomendável é automatizar o processo com a criação de programas que façam uma varredura dos *scripts* à procura de vulnerabilidades. Este processo é tanto mais difícil quanto mais heterogêneas forem as técnicas de programação. Isto é, se a equipe de programação é conhecida, é possível definir módulos ou funções específicos que devem estar presentes nos *scripts* que enviam e-mail para impedir o ataque. Caso algum *script* seja publicado sem estes códigos, a auditoria acusará o problema.

Quando a equipe e os padrões de programação são desconhecidos do administrador, como ocorre em um provedor de hospedagem web, a auditoria precisa ser mais generalista e buscar padrões de códigos vulneráveis, como *scripts* que não filtram a presença de CRLF nas variáveis que formam o header dos e-mails. Uma vez encontradas as falhas de programação, os desenvolvedores responsáveis devem ser notificados e orientados sobre as soluções para o problema.

Conforme explicado por Simões, como as ações pró-ativas não garantem 100% de eficácia, é necessário que a equipe de suporte esteja preparada para responder reativamente a um ataque de SMTP *Injection* no menor tempo possível. Para isto, o monitoramento 7x24 da fila de saída e dos *logs* do serviço SMTP é fundamental. Uma variação abrupta em parâmetros como o número de e-mails enviados por um mesmo remetente e a taxa de rejeição das mensagens transmitidas pelo servidor são indícios da realização do ataque. Quando isto ocorre,

cabe aos administradores do serviço verificar a natureza das mensagens e, se confirmado o problema, suspender a operação do site vulnerável até que a devida correção seja implementada.

7. Considerações finais

Para a boa prevenção contra um ataque de SMTP *Injection*, o ideal é a junção de um bom programador com um projeto bem elaborado que seja hospedado em um data-center sério e com processos de monitoramento. Muitas vezes o programador se esforça para ter um bom código e acaba hospedando seu sistema em uma máquina compartilhada que contém *scripts* que possibilitam ataques e que permitam que outro domínio encaminhe um e-mail no nome de outro domínio.

Algo também importante é a utilização de SPF (Sender Policy Framework) no servidor. SPF é uma política de regras do DNS para evitar que outros domínios enviem e-mails em seu nome. Muitas vezes você tomou todos os cuidados para ter *scripts* seguros e está hospedando em um servidor que possui diversas boas práticas

de hospedagem, entretanto, este hospedeiro não possui SPF. Outro *script* que está hospedado em outro *data-center* pode encaminhar seus e-mails e seu cliente ainda pode receber diversas reclamações comentando que seu domínio está sendo atacado, quando na verdade quem está sofrendo SMTP *Injection* é outro hospedeiro.

Referências e links sugeridos

[RFC 822] – <http://www.faqs.org/rfcs/rfc822.html>
[RFC 2821] – <http://www.faqs.org/rfcs/rfc2821.html>
[RFC 2822] – <http://www.faqs.org/rfcs/rfc2821.html>
[RFC 4408] – <http://www.faqs.org/rfcs/rfc4408.html>
[manual PHP] – <http://br2.php.net/manual/en/ref.mail.php>
[Descrição de SPF no Wikipedia com link para a página oficial] – <http://pt.wikipedia.org/wiki/Spf>

Ricardo Striquer Soares - ricardo@ideiaspontual.com

Técnico em processamento de dados com mais de 15 anos de experiência. Autor de diversos softwares, tanto desktop quanto webbased, há cinco anos Ricardo é empresário do ramo de tecnologia, sendo um dos sócios da empresa Idéias.ual (<http://www.ideiaspontual.com/>), participando e gerenciando vários projetos de desenvolvimento, instalação, configuração e integração de sistemas.

Ricardo mantém uma publicação periódica no blog programabrasil.blogspot.com.

Edições Anteriores

Faça o [download](#) das três edições anteriores.



Gerando planilhas em Excel com PHP-GTK

Neste artigo iremos construir uma ferramenta para lançamento de despesas financeiras cujo objetivo é criar uma planilha no formato Excel.

O PHP é uma linguagem incrível em relação à quantidade de bibliotecas de terceiros que gravitam em torno de sua órbita. Temos bibliotecas para os mais variados fins, como envio de emails, manipulação de imagens, geração de documentos PDF, dentre outros.

O PHP possui um grande repositório de classes chamado PEAR (PHP Extension and Application Repository), como muitos de vocês já devem saber. Este grande repositório orientado a objeto é constantemente fruto de artigos em revistas e sites sobre PHP. Neste artigo em específico, vamos provar que podemos utilizar praticamente qualquer biblioteca comumente utilizada no ambiente Web no PHP-GTK. Para isso, escolhemos a biblioteca Spreadsheet Excel Writer, que é um pacote responsável pela geração de arquivos em formato Excel. Para mostrar seu funcionamento em conjunto com o PHP-GTK, iremos construir um programa para lançamentos de despesas, que irá inserindo estes lançamentos em uma listagem para posterior geração do arquivo em formato Excel.

1. Listas e Árvores

Listas e árvores no PHP-GTK são implementados através da classe GtkTreeView, que pode ser utilizada para exibir tanto árvores quanto listas.



Figura 1 – Componente GtkTreeView

O conceito mais importante por trás de uma GtkTree-

View é o fato dela implementar o padrão MVC (Model, View, Controller), ou seja, a separação entre os dados e a forma pela qual estes são exibidos em tela. Os dados, sejam eles números, textos ou imagens são armazenados em um modelo, que pode ser GtkListStore (para armazenar listas) ou GtkTreeStore (para armazenar árvores). O modelo de dados é, então, atribuído a alguma visualização (GtkTreeView). Sempre que o modelo de dados é alterado, automaticamente sua exibição é atualizada em tela, por sua visualização (GtkTreeView).

A camada Controller contém objetos que capturam ações derivadas de interação do usuário, coordenando assim, objetos das camadas Model e View para fornecer a resposta adequada ao usuário.

2. Spreadsheet Excel Writer

Como dissemos anteriormente, a biblioteca Spreadsheet Excel Writer permite a criação de planilhas no formato Excel. Ela suporta fórmulas, imagens e formatação de textos e células.

Assim como toda biblioteca que faz parte do PEAR, a Spreadsheet Excel Writer deve ser instalada através do seguinte comando:

```
#pear install SpreadSheet_Excel_Writer-0.9.1
```

Após isto, os seus arquivos devem ser disponibilizados dentro da pasta lib de sua instalação do PHP.

3. O programa

Para começar, iremos estender a classe GtkWindow e criar toda a interface em seu método construtor. A nossa janela de lançamentos terá basicamente uma caixa vertical (GtkVBox) contendo uma série de caixas horizontais (GtkHBox).

Em cada caixa horizontal, teremos um par de rótulos (GtkLabel) e um campo de entrada de dados (GtkEntry). Ao final, criamos um botão adicionar (STOCK_ADD), cujo objetivo é coletar os dados digitados pelo usuário nestes campos e adicionar em uma listagem (objeto GtkTreeView). Este botão está conectado ao método **onAddLinha**

(**).** Isto significa que sempre que o botão for clicado, este método será executado. Também temos um botão de salvar (STOCK_SAVE), cujo objetivo é coletar todos os lançamentos da listagem e gerar a planilha Excel a partir deles.

Neste método construtor, ainda definimos os tamanhos dos campos por meio do método **set_size_request()** e executamos o método **createTreeView()**, cuja responsabilidade é declarar nossa listagem de lançamentos e suas respectivas colunas. Vários pontos deste programa foram suprimidos em virtude do espaço aqui reduzido, mas poderão ser encontrados no site indicado ao final do artigo.

```
<?php
/*
 * Exportar Excel
 * Planilha de gastos com exportação de excel
 * Adianti Solutions (www.adianti.com.br)
 */
class ExportarExcel extends GtkWindow
{
    private $list;        // apresentação dos dados
    private $model;       // modelo de dados

    private $descricao;   // campo de descrição
    private $data;        // campo de data
    private $valor;       // campo de valor
    private $operacao;    // campo de operação

    /*
     * Método construtor
     * Instancia a janela e cria toda interface
     */
    function __construct()
    {
        // cria a janela
        parent::__construct();
        parent::set_border_width(4);
        parent::set_size_request(470,300);

        // cria uma caixa vertical
        $vbox = new GtkVBox;

        // cria rótulos de texto
        $label1 = new GtkLabel('Descrição : ');
        $label2 = new GtkLabel('Data: ');
        ...
        // define o tamanho dos rótulos
        $label1->set_size_request(200,-1);
        $label2->set_size_request(200,-1);
        ...
        // cria os campos de entrada de dados
        $this->descricao = new GtkEntry;
        $this->data = new GtkEntry;
        $this->valor = new GtkEntry;
        $this->operacao = new GtkEntry;

        // define os tamanhos dos campos
        $this->descricao->set_size_request(240...
        $this->data->set_size_request(80,-1);
        ...
        // cria caixas horizontais
        $hbox1 = new GtkHBox;
        $hbox2 = new GtkHBox;
        $hbox3 = new GtkHBox;
        $hbox4 = new GtkHBox;

        // adiciona os labels em suas caixas
        $hbox1->pack_start($label1, false, false);
```

```
$hbox2->pack_start($label2, false, false);
...
// adiciona os campos em suas caixas
$hbox1->pack_start($this->descricao...);
$hbox2->pack_start($this->data...);
$hbox3->pack_start($this->valor...);
$hbox4->pack_start($this->operacao...);

// adiciona as caixas na caixa vertical
$vbox->pack_start($hbox1, false, false);
$vbox->pack_start($hbox2, false, false);
...
// cria um botão de adicionar
$button =
GtkButton::new_from_stock(Gtk::STOCK_ADD);
// define a ação do botão
$button->connect_simple('clicked',
array($this, 'onAddLinha'));

// cria o objeto TreeView
$this->createTreeView();

// adiciona o TreeView na caixa vertical
$vbox->pack_start($this->list);

// cria um botão de salvar
$button =
GtkButton::new_from_stock(Gtk::STOCK_SAVE)
// define a ação do botão
$button->connect_simple('clicked',
array($this, 'onExportar'));

// adiciona a caixa vertical na janela
parent::add($vbox);
}
```

O método **createTreeView()** será executado a partir do método construtor e seu objetivo é instanciar o objeto **GtkTreeView**, para implementar a listagem de lançamentos. Esta listagem armazenará os dados em um objeto **GtkListStore** (propriedade **list**), que terá quatro colunas do tipo **string**.

```
function createTreeView()
{
    // instancia treeview
    $this->list = new GtkTreeView;

    // cria modelo
    $this->model = new GtkListStore(
        Gtk::TYPE_STRING, Gtk::TYPE_STRING...);
    // define o modelo da treview
    $this->list->set_model($this->model);

    // cria quatro colunas
    $column1 = new GtkTreeViewColumn('Desc.');
```

Sempre que o usuário clicar no botão “Adicionar”, o método **onAddLinha()** será executado. O objetivo deste método é obter os valores digitados pelo usuário nos

campos de entrada de dados que são armazenados em propriedades deste objeto (descrição, data), através do método **get_text()** da classe GtkEntry.

Quando obtemos os valores destas variáveis, adicionamos eles ao modelo de dados da listagem (GtkTreeView) através do método **append()**, passando um array com o conteúdo da linha. Posteriormente, limpamos os conteúdos dos campos com o método **set_text()** e posicionamos o foco do cursor no campo para descrição.

```
function onAddLinha()
{
    // obtém os valores digitados pelo usuário
    $descricao = $this->descricao->get_text();
    $data      = $this->data->get_text();
    $valor     = $this->valor->get_text();
    $operacao  = $this->operacao->get_text();

    // adiciona na lista
    $this->model->append(array($descricao,
                              $data, $valor, $operacao));

    // limpa os conteúdos digitados
    $this->descricao->set_text('');
    $this->data->set_text('');
    ...
    // posiciona o foco no campo descrição
    parent::set_focus($this->descricao);
}
```

O método **onExportar()** será executado sempre que o usuário clicar no botão “Salvar”. O primeiro passo é incluir a biblioteca Spreadsheet Excel Writer para disponibilizar esta classe para a aplicação. Após, abrimos um diálogo de salvar arquivos para coletar o nome e o caminho onde o usuário deseja salvar a planilha.

```
function onExportar()
{
    // inclui a biblioteca SpreadsheetWriter
    require_once "Spreadsheet/Excel/Writer..."

    // abre um diálogo de salvar arquivo
    $dialog = new GtkFileChooserDialog...;

    // exibe diálogo
    $response = $dialog->run();

    // Verifica resposta do usuário
    if ($response == Gtk::RESPONSE_OK)
    {
        // obtém o nome do arquivo
        $filename = $dialog->get_filename();
    }
    // destrói diálogo
    $dialog->destroy();
}
```

Depois de coletar o nome do arquivo, instanciamos um objeto da classe Spreadsheet_Excel_Writer, responsável por fornecer os métodos que nos permitem gerar as planilhas em formato Excel.

```
// cria uma planilha Excel
$xmls = new Spreadsheet_Excel_Writer
($filename);
```

O funcionamento desta classe é bastante simples. Primeiro definimos um conjunto de “estilos” a serem utilizados para as células. Cada estilo define características como tamanho, cor e fonte. Posteriormente, utilizamos estes estilos pelo método **write()**.

```
// define um formato para o título
$titleFormat = $xls->addFormat();
$titleFormat->setFontFamily('Helvetica');
$titleFormat->setSize('14');
$titleFormat->setColor('black');
...
// define um formato para as colunas
$columnFormat = $xls->addFormat();
$columnFormat->setFontFamily('Helvetica');
$columnFormat->setSize('12');
$columnFormat->setColor('black');
...
// define um formato para os dados
$dataFormat = $xls->addFormat();
$dataFormat->setFontFamily('Courier');
$dataFormat->setSize('12');
$dataFormat->setColor('black');
...
```

Após definirmos alguns estilos, utilizamos o método **addWorksheet()** para criar uma aba, ou uma planilha de trabalho dentro do nosso arquivo. Veja na parte inferior da figura 3, o nome “PHP Magazine”.

Após isto, escrevemos os cabeçalhos da planilha através do método **write()**, indicando o endereço de cada célula, através de coordenadas linha e coluna.

```
// Adiciona uma aba à planilha
$sheet = $xls->addWorksheet('PHP Magazi-
ne');

// escreve nas células (linha, coluna)
$sheet->write(0,1,'Caixa', $titleFormat);
$sheet->write(1,0,'Descrição'...);
$sheet->write(1,1,'Data', $columnFormat);
$sheet->write(1,2,'Valor'...);
$sheet->write(1,3,'Operação'...);
```

Depois de escrevermos os cabeçalhos da planilha, percorremos os dados da listagem (objeto GtkTreeView), coletamos célula a célula, através do método **get_value()** e escrevemos estes valores dentro da planilha através do método **write()**.

```
// percorre o modelo de dados
$i = 2;
$iter = $this->model->get_iter_first();

while ($iter)
{
    // obtém os valores do modelo de dados
    ...
    $valor =
    $this->model->get_value($iter, 2);
    $operacao =
    $this->model->get_value($iter, 3);

    // escreve os dados na planilha
    $sheet->write($i,2,$valor...);
    $sheet->write($i,3,$operacao...);

    // pega o próximo iterador
```



```

    $iter =
    $this->model->iter_next($iter);
    $i ++;
}

```

Ao final, fechamos o objeto planilha e abrimos um diálogo de sucesso com a mensagem Planilha Gerada!!

```

// fecha a planilha
$xml->close();
// abre um diálogo de mensagem de sucesso
$dialog = new GtkMessageDialog
(...'Planilha gerada !!!');
$response = $dialog->run();
// fecha o diálogo
$dialog->destroy();
}
}

```

Aqui é onde começa o nosso programa na verdade, a classe ExportaExcel é instanciada

```

// instancia a classe ExportaExcel
$janela = new ExportarExcel;
$janela->show_all();
Gtk::Main();
?>

```

Aqui você confere a tela principal do nosso software de controle de lançamentos financeiros.

Figura 2 – Tela do gerador de planilhas

Nesta tela seguinte, você confere a planilha eletrônica gerada no formato Excel.

	A	B	C	D
1		Caixa		
2	Descrição	Data	Valor	Operação
3	Material de Escritório	10/04/2007	50	D
4	Gasolina	20/04/2007	100	D
5	Receita Google Adsense	30/04/2007	200	C
6	Comprar um Mouse Novo	10/05/2007	40	D
7				
8				

Figura 3 – Planilha gerada

Considerações finais

Neste artigo, com menos de 300 linhas de código, escrevemos um simples programa para lançamentos financeiros com geração de planilhas Excel em PHP-GTK. Como o programa ficou um pouco grande para o formato da revista, optamos por cortar alguns pedaços, substituindo-os por "..." por motivos didáticos. Você pode fazer download da aplicação completa no site da pSheet, em <http://psheet.php-gtk.com.br>

Referências e links sugeridos

[PHP-GTK Brasil] – <http://www.php-gtk.com.br>
[Livro PHP-GTK] – <http://www.php-gtk.com.br/book>
[Site do Autor] – <http://www.pablo.blog.br>
[Site da Planilha] – <http://psheet.php-gtk.com.br>
[Site do PEAR] – <http://pear.php.net>

Pablo Dall'Oglio - pablo@dalloaglio.net

Pablo Dall'Oglio é formado em Análise de Sistemas pela UNISINOS. Autor do livro sobre PHP-GTK pela Novatec Editora. Programa em PHP-GTK desde sua criação em 2001. É membro do time de documentação e criador da comunidade brasileira de PHP-GTK (www.php-gtk.com.br). Atualmente, é diretor de tecnologia e proprietário da Adianti Solutions (www.adianti.com.br), onde atua como consultor de tecnologia e engenheiro de software. Pode ser contatado pelo e-mail pablo@php.net.



AJAX e PHP I: Conhecendo AJAX

Neste artigo, pretendo apresentar ao leitor a tecnologia AJAX, resolvendo algumas confusões sobre o que realmente é, e como e onde deve ser usada. Este artigo apresenta um exemplo de como implementar uma solução AJAX sem auxílio de Frameworks.

Após palestrar sobre o assunto no 1º PHPDF Road-Show, senti-me motivado a, finalmente, fazer minha contribuição para esta revista e, com isso, nasceu este artigo, um primeiro passo que ajudará todos a dar uma espiada no universo AJAX, sem se sentir totalmente perdido.

Desde que comecei a pesquisar sobre AJAX devo admitir que me apaixonei pela forma que ele permite executar tarefas simples e avançadas de maneira amigável, simples e ágil. Desde então, tenho procurado repassar este conteúdo, contribuindo brevemente com o site AJAX Online (www.ajaxonline.com.br) e publicando diversos artigos em meu blog pessoal.

1. AJAX: detergente ou tecnologia?

O caminho para o aprendizado de AJAX não é muito longo, mas mesmo assim possui diversos atalhos que podem agilizar no desenvolvimento, porém podem acabar “saindo pela culatra” mais tarde, pois prejudicam o conhecimento.

O primeiro passo e também o ponto de maior confusão, é a velha pergunta: “O que é AJAX?”. Leigos lhe dirão que é um detergente, alguns dirão que é uma nova linguagem de programação, eu lhes digo que se trata apenas de uma nova forma de ver algo mais antigo.

O AJAX é um conjunto de técnicas novas, envolvendo diversas tecnologias antigas, dentre elas: Javascript, XML, Document Object Model (DOM). Dentre estas tecnologias o único elemento novo é o XMLHttpRequest, e mesmo assim ele não é tão novo quanto parece.

O XMLHttpRequest surgiu pela primeira vez em 2000, criado pela Microsoft para ser usado no Outlook Web Access. Em 2002, a Mozilla incorporou o objeto em seus browsers e somente em 2006 foi lançado o primeiro draft na W3C. Neste ponto dá início o grande hype da “Web 2.0” e o AJAX começa a ser amplamente utilizado.

Conceitualmente, AJAX significa “Asynchronous JavaScript and XML” ou Javascript Assíncrono e XML, mas

na prática é possível utilizar objetos com notação JSON (JavaScript Object Notation) também, ao invés de XML. Mais adiante vou discutir as vantagens e desvantagens disto.

O grande conceito de AJAX é permitir que o cliente se comunique com o servidor através deste request, que é realizado em segundo plano, sem recarregar a página, efetivamente unindo a tecnologia client-side com a tecnologia server-side e potencializando a comunicação.

Atenção: use com moderação! Embora seja algo muito legal, o AJAX não deve ser usado em qualquer lugar, pois, ao invés de ajudar, pode tornar truncada a experiência do usuário, evite usar AJAX, por exemplo, como sua forma de navegação principal.

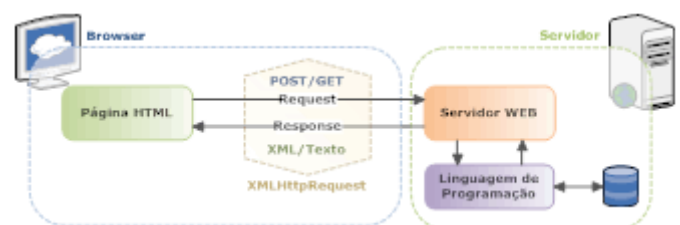


Figura 1 – Fluxo de uma requisição AJAX

2. Aprendendo AJAX

Como disse anteriormente, o caminho para aprender AJAX tem diversos atalhos e, no momento em que se decide aprender AJAX, é necessário avaliar qual caminho irá tomar. Existem dois caminhos principais para se trilhar, aprender AJAX **com** frameworks ou **sem** frameworks. Cada um tem sua vantagem como, por exemplo, com frameworks a produtividade aumenta, porém, sem ter trilhado o caminho “*primitivo*”, dar manutenção em scripts que apresentam erros pode se tornar uma tarefa difícil. Eu, pessoalmente, defendo que se deve aprender pelo caminho difícil para depois viver no caminho fácil, então vou abordar as duas formas de se usar e aprender AJAX, iniciando neste artigo pelo AJAX puro, ou seja, *primitivo*, e deixando o segundo exemplo para uma futu-

ra contribuição.

2.1. AJAX primitivo

Implementar funções de AJAX "na mão" não é tarefa difícil, ao contrario do que muitos pensam, porém é importante, antes de mais nada, entender como funciona o objeto XMLHttpRequest.

O Request simula o funcionamento do browser. Neste último, quando se clica em um link, é feita uma requisição para determinado arquivo e o resultado desta requisição é apresentado na tela do browser. No AJAX, essa requisição é feita da mesma forma, porém o resultado não se carrega na janela e sim dentro do próprio objeto de request, isso causa o efeito de segundo plano que o AJAX apresenta.

Para isso, o objeto possui alguns métodos e propriedades importantes. Primeiro as propriedades:

ReadyState: estado atual da requisição, indica se a página ainda está sendo buscada ou se o resultado já chegou. Pode possuir estes estados:

- 0 = uninitialized (não inicializado)
- 1 = loading (carregando)
- 2 = loaded (carregado)
- 3 = interactive (interagindo)
- 4 = complete (pronto)

ResponseText: resultado da requisição em formato de texto comum, usado também para JSON.

ResponseXML: resultado em formato XML.

Status: Códigos de erro ou sucesso como: 200,404,403 etc.

StatusText: mesmo erro, mas de forma textual (Not found...).

Onreadystatechange: propriedade/evento, indica a função que será executada quando a requisição mudar seu readyState

Para executar seu papel, o objeto conta também com alguns métodos:

Open ("method", "URL", async, "uname", "pswd"): este método abre uma nova requisição para a URL determinada com o método escolhido (GET ou POST). Esta requisição pode ser síncrona ou assíncrona, determinando se o código continua sendo executado independente da resposta ou se a resposta é aguardada para continuar o processamento.

Send(content): este método inicia a comunicação com a URL e recebe apenas o parâmetro (opcional) de que conteúdo deve enviar. Este conteúdo está em formato de URL, ou seja, var=valor&var2=valor2.

Abort(): este método é simples e pode ser muito importante, pois finaliza uma requisição que ainda não retornou qualquer resposta do servidor. É útil em casos onde novas requisições podem ocorrer descartando as

anteriores sem respostas, como em campos de auto-complete.

SetRequestHeader("label", "value"): este método é importante quando POST é utilizado, permitindo "setar" o conteúdo da requisição para "multipart/form-data", por exemplo. Uma falha neste ponto pode derrubar toda a requisição.

GetResponseHeader("headername") e getAllResponseHeaders(): estes métodos são úteis em verificações de segurança que permitem,

por exemplo, verificar se o conteúdo da resposta realmente está em JSON ou XML.

Para entender como usar este objeto e como devemos manipular seus métodos e propriedades, a seguir, definirei um exemplo simples de uma aplicação que pode utilizar AJAX, e acompanharei passo a passo com vocês a implementação.

3. Desenvolvendo um mural de recados com AJAX

O sistema proposto é bem simples, justamente para facilitar o entendimento do AJAX e não se focar em outros aspectos. De forma geral, o sistema deve possuir uma região onde apresenta as mensagens, o mural, e um campo que permita a alguém digitar algo, sendo isto inserido no mural sem haver um re-carregamento da página como um todo.

3.1 Modelagem

O primeiro passo de um sistema deve ser sempre a modelagem, utilizando UML ou outras formas de modelagem, porém, como este não é nosso foco, farei apenas uma modelagem simples para compreensão do sistema.

O sistema contará com duas funcionalidades: adição e visualização de imagens. Portanto a figura 2 apresenta o fluxo dos dados e eventos do momento que o usuário envia seu recado até ele aparecer no mural (por motivos de simplicidade, ao recarregar a página, as mensagens não são recarregadas no mural).

Separei o sistema em três módulos: javascript (verde), HTML (amarelo) e PHP (azul), para podermos ver a interação entre essas linguagens.

No módulo de javascript, será realizada toda a operação de AJAX propriamente dita, como demonstrado na figura acima. Para isso, devemos criar 3 funções: uma para criar o request, uma para enviar a mensagem e outra para receber o retorno e publicar no mural.

No PHP precisamos de duas funções: uma que receba os dados e grave no banco e outra que formate o retorno e envie de volta para o javascript. Neste caso, estamos usando XML, mas o uso de JSON será demonstrado em futuros artigos. A estrutura do retorno XML proposta é a seguinte:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<response>
```

```
<erro>0/1</erro>
<item id="1">Retorno em HTML ou Texto</item>
</response>
```

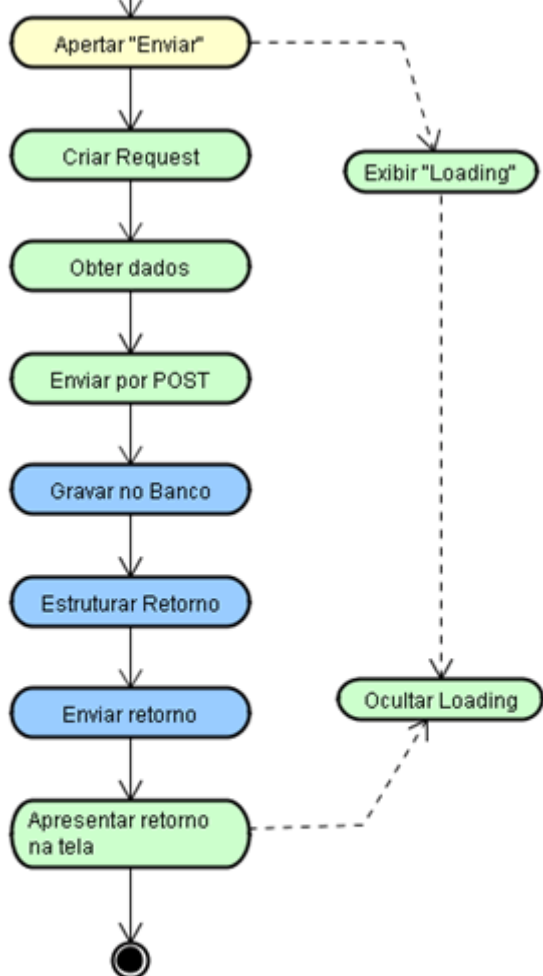


Figura 2 - Fluxo do exemplo

No HTML teremos apenas um campo de input, um botão e um div que representa o mural.

A figura 3 demonstra como as funções interagem entre si, de forma simplificada. Escolhi separar o sistema em 3 arquivos, onde cada um representa um dos módulos acima descritos.



Figura 3 - Interação das funções

A tabela onde os dados serão guardados pode ser criada com este código:

```
CREATE TABLE `mural` (
  `id` int(3) NOT NULL auto_increment,
  `msg` text character set latin1 collate lat-
  in1_general_ci,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

3.2 Módulo Javascript

A primeira função que precisamos é a função responsável por criar o request de XMLHttpRequest, mas por que fazer uma função? Simples. Como o IE implementa o objeto de forma diferente de outros browsers, implementamos estes diferentes métodos em uma função, com isso a instanciação do objeto se torna algo mais simples e pontual. Veja o código:

```
function criaRequest(){
  try {
    request = new XMLHttpRequest();
  } catch (trymicrosoft) {

    try {
      request = new ActiveXObject
("Msxml2.XMLHTTP");
    } catch (othermicrosoft) {

      try {
        request = new ActiveXObject
("Microsoft.XMLHTTP");
      } catch (failed) {
        request = false;
      }
    }
  }

  if (!request)
    alert("Error initializing XMLHttpRequest!");
  else
    return request;
}
```

Agora devemos definir a função que vai enviar estes dados. Esta função deve chamar a função anterior para obter um request e, então, iniciar seu processamento. O próximo passo é buscar o texto do campo e montar a requisição. Usaremos o método POST e, com isso, devemos definir o header de Content-Type, passando o texto do campo com o nome de variável "msg".

Neste momento, também é importante definirmos a função que será executada ao final da requisição, usando o onreadystatechange. Para dar um efeito a mais neste momento, tornamos visível um DIV com o texto "Carregando..." para que o usuário possa saber que algo está acontecendo.

```
function enviaDados(){
  //Novo Request
  linkReq = criaRequest();

  if(linkReq != undefined){
    //Pegar dados
    var msgBox = document.getElementById
('msgBox');

    //Montar requisição
```



```

        linkReq.open
        ("POST", "mural.ajax.php", true);
        linkReq.setRequestHeader('Content-
Type', 'application/x-www-form-urlencoded');
        linkReq.onreadystatechange = recebe-
Dados;

        var params = "msg="+msgBox.value;

        //Carregar DIV de "loading"
        document.getElementById
('loading').style.display = 'block';

        //Enviar
        linkReq.send(params);

        //Esvaziar form
        msgBox.value = "";
    }
}

```

Definimos a função `recebeDados` como o retorno do request, mas ela será chamada a cada alteração de estado. Portanto, precisamos verificar o novo estado para sabermos se o resultado final já foi retornado. Uma vez confirmado o estado 4, podemos tratar o retorno.

Para isso, como foi definido o retorno em XML, leremos os dados da variável `responseXML`, lendo os campos através de DOM e inserido este retorno no DIV, dentro de uma DIV própria da mensagem. Esta manipulação toda é feita através de DOM com funções que estão disponíveis desde o início dos browsers.

```

function recebeDados(){
//Verificar pelo estado "4" de pronto
if (linkReq.readyState == '4'){

    //Pegar dados da resposta XML
    var xmlRes = linkReq.responseXML;

    //Verificar erro
    var erro = xml-
Res.getElementsByTagName('erro');

    if (erro[0].firstChild.nodeValue ==
'1'){
        alert("Erro no retorno"+erro
[0].firstChild.nodeValue);
    }else{
        //Pegar mensagem
        var msg = xml-
Res.getElementsByTagName('item');

        //Pegar DIV destino
        var targetDiv = docu-
ment.getElementById('msgList');

        //Montar Nova msg
        var mDiv = docu-
ment.createElement('div');
        mDiv.id = "msg_"+msg[0].id;
        mDiv.innerHTML = msg
[0].firstChild.nodeValue;

        //Adicionar ao destino
        targetDiv.appendChild(mDiv);
    }
}

```

```

        //Remove loading
        document.getElementById
('loading').style.display = 'none';
    }
}

```

3.3 Módulo PHP

Este módulo é simples e, na verdade, pode ser substituído por qualquer linguagem server-side. Como uma requisição AJAX nada mais é do que uma requisição normal feita em segundo plano, o nosso arquivo PHP trabalha como qualquer outro script, recebendo dados, processando e retornando, sendo a única diferença que vamos retornar XML e não HTML.

Ao invés de criar duas funções como defini anteriormente, vou apenas implementar um script de forma estruturada que execute as ações das duas funções.

A primeira parte deve receber os dados e gravar em banco. Utilizei neste exemplo um banco mysql e as funções normais de mysql para não complicar muito o aprendizado.

A segunda parte deve pegar os mesmos dados e convertê-los em uma saída XML, de acordo com o padrão que escolhemos.

```

<?php

//Conexão com banco
$db = mysql_connect
("localhost","user","senha");
$db_selected = mysql_select_db('phpajax',
$db);

//Simular processo demorado para vermos o
"carregando" (descartar em produção)
sleep(3);

//Receber dados
//Gravar no banco
$sql = "INSERT INTO mural (msg) VALUES
('".$_POST['msg']."'");
$res = mysql_query($sql);

//XML de Retorno
$xmlDoc = new DOMDocument('1.0','iso-8859-1');
$response = $xmlDoc->createElement
('response');
$response = $xmlDoc->appendChild($response);

//Elemento de erro
$erro = $xmlDoc->createElement('erro',($res)?
"0":"1");
$erro = $response->appendChild($erro);

//Elemento item
$item = $xmlDoc->createElement('item',$_POST
['msg']);
$item->setAttribute('id',mysql_insert_id());
$item = $response->appendChild($item);

header('Content-Type: application/xml');

echo $xmlDoc->saveXML();

//Teste Manual

```

?>

3.4 Módulo HTML

Como descrito várias vezes, este simples arquivo tem um DIV de carregando, um DIV que será o Mural e um campo de input com botão, sem o uso de um form, pois os dados são obtidos diretamente pela função `enviaDados`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/
html; charset=iso-8859-1" />
<script src="mural.funcs.js" lan-
guage="javascript"></script>
<title>Untitled Document</title>
<link href="style.css" rel="stylesheet"
type="text/css" />
</head>

<body>

<div id="form">
<textarea id="msgBox" rows="3" cols="10"></
textarea>
<input type="button" value="Enviar" on-
click="enviaDados();" />
</div>

<div id="loading">Carregando...</div>

<div id="msgList"></div></body></html>
```

3.4 Produto Final

Para ver o produto final funcionando, verifique o endereço abaixo:

http://www.rafaeldohms.com.br/cursos/php_ajax/pratica1/mural.php

E faça download do código fonte neste endereço:

http://www.rafaeldohms.com.br/cursos/php_ajax/php_ajax_pratica1.zip

Este exemplo como eu disse, apenas apresenta os dados uma vez enviados, e não os mostra novamente quando a página é re-carregada.

Considerações finais

Neste primeiro artigo, espero ter apresentado o que realmente é AJAX e dissolvido alguns dos mitos que circulam por toda internet em relação à tecnologia.

Considero o AJAX parte fundamental da internet e peça importante no desenvolvimento de sites que aproximem o usuário e tornem sua experiência na web algo mais natural e intuitivo. Por isso, seu aprendizado é algo muito importante e saber usar este recurso de forma "crua" qualifica o profissional e pode auxiliar muito na hora do aperto.

Em um próximo artigo, irei me aventurar no mundo dos frameworks, mostrando como obter uma maior produtividade em seus trabalhos, facilitando muito o as tarefas diárias.

Caso tenham alguma dúvida ou problema, sintam-se a vontade para visitar meu blog ou utilizar o contato abaixo, pois estarei disponível para tirar qualquer dúvida.


Referências e links sugeridos

[AJAX Online] – <http://www.ajaxonline.com.br>

[Meu Blog] – <http://www.rafaeldohms.com.br>

Rafael Machado Dohms - rafael@rafaeldohms.com.br

Pós-graduado em Projetos pela Fundação Getúlio Vargas e graduado em Engenharia da Computação, pelo UniCEUB, em Brasília, trabalha com tecnologias web há 8 anos, com experiências no setor público e privado. Atualmente, é membro da coordenação do PHPDF, grupo de desenvolvedores PHP do Distrito Federal (www.phpdf.org.br) e mantém um blog focado em assuntos relacionados à tecnologia para web (<http://www.rafaeldohms.com.br>).



fisl9.0
9º Fórum Internacional
Software Livre
A tecnologia que liberta

17, 18 e 19 de abril de 2008
Centro de Eventos da PUCRS
Porto Alegre - RS - Brasil

A tecnologia que liberta

Webmail com PHP + JAVA

Neste trabalho será apresentado um novo método de autenticação para *webmail* que usa como recursos as linguagens PHP e JAVA através de tecnologias J2ME.

Neste artigo iremos propor um esquema de autenticação para *webmail* (poderá ser futuramente estendido para outros sistemas de autenticação) que usará a linguagem PHP do lado do servidor e JAVA através de tecnologias J2ME no cliente. Na verdade, o nosso grande foco consiste em que o servidor gerará um senha provisória através de alguma técnica de GNA (Geração de Números Aleatórios) e após "encriptada" por um servidor enviará para um usuário que dispõe de um celular ou PDA (*Personal Digital Assistants*) para "descriptar" esta senha, e assim este usuário poderá logar-se usando esta senha provisória. A vantagem do uso desta técnica consiste em possibilitar que um usuário autentique-se em uma máquina infectada, por exemplo, com um *keylogger*, uma vez que a senha gerada pelo servidor não poderá ser usado novamente para este usuário autenticar-se no *webmail*.

1. Motivações

A grande motivação deste artigo se concentra no grande número de artefatos maliciosos (*malware*) que tramitam na *web*. Entre estes um dos que mais se destaca é o *spyware*. Um *spyware* é um *software* que auxilia a captura de informações sobre uma pessoa, organização ou computador sem o seu consentimento. Um *spyware* pode enviar tais dados para outra identidade sem o seu conhecimento ou tomar controle do computador sem que o usuário saiba disso. De todos os tipos de *malwares* existentes o *spyware* é o mais comum. Segundo uma pesquisa conduzida pela Dell em setembro de 2004, estima-se que aproximadamente 90% dos PCs com *Windows* possuíam no mínimo um *spyware*. Este tipo de *software* foi responsável por metade das falhas reportadas por usuários da Microsoft em ambiente *Windows*. Outro estudo apontou uma média de 25 *spywares* por PC. No nosso trabalho atentaremos a presença de um tipo específico de *spyware*: o *keylogger*.

Um *keylogger* é um tipo de *spyware* cuja finalidade é capturar tudo que um determinado usuário digita em um teclado. Existem diversos tipos de uso de *keyloggers*, como por exemplo aqueles que monitoraram um funcio-

nário de uma empresa. Entretanto os *keyloggers* também são usados para fins ilícitos: capturar senhas, números de cartões de crédito e qualquer outro tipo de informação puramente sigilosa. Estas informações podem ser enviadas diretamente para o *e-mail* do indivíduo que age de má fé. Tudo isso sem o consentimento da vítima que está acessando o computador "infectado" [Pereira E. et al.]. Podemos dividir os *keyloggers* em três classes diferentes: *software keylogger*, *hardware keylogger* e *kernel keylogger* [Pereira E. et al.].

Software keylogger. Talvez seja a classe de *keylogger* mais difundida na *Web*. Estes *softwares* usam técnica de *hooking*. Os programadores deste tipo de *keylogger* utilizam funções disponibilizadas pela API (*Application Program Interface*) do sistema operacional para captar mensagem e teclas pressionadas antes que estas funções sejam tratadas. Esses *keyloggers* podem ser instalados remotamente, no entanto, são os mais lentos e facilmente detectáveis por programas como anti-vírus e *anti-spywares*.

Hardware keylogger. Trata-se de um equipamento físico que se localiza entre o teclado e o gabinete do computador da vítima. Apesar de possuir pouco recurso computacional, ele processa bem rápido as informações capturadas. Este dispositivo não é detectado por anti-vírus e nem *anti-spyware*, entretanto, pode ser visualmente detectado.

Kernel keylogger. Este *keylogger* atua no nível do *kernel* do sistema operacional. A confecção deste tipo de *keylogger* exige técnicas apuradas de programação e de sistemas operacionais. O sistema operacional encontra muita dificuldade também em detectá-lo. Este tipo de *keylogger* não é capaz de captar informações em nível de aplicação: operações de auto-completar, copiar e colar. Devido ao fato de trabalharem no núcleo do sistema.

2. Descrição do Método

Aqui discutiremos a técnica que iremos propor neste trabalho. O nosso grande objetivo é criar uma senha provisória para dar acesso a um determinado usuário em

um sistema uma única vez. Após autenticado com esta senha provisória, este usuário não poderá inserir esta senha a fim de autenticar-se neste sistema. Este fato nos garante que se caso tenha instalado algum tipo *keylogger* na máquina de acesso, a senha original não estará comprometida e caso um atacante intercepte a senha provisória, esta não será mais válida após a sua autenticação. Logo o usuário estaria livre de qualquer tipo de *software* comprometer a segurança aspecto da autenticação. Para gerar esta senha provisória o servidor pode usar alguma técnica de Geração de Números Aleatórios (GNA) que gere números bem distribuídos de aproximadamente 80 *bits*.

A grande dificuldade se concentra em enviar esta senha de maneira segura para o usuário que deseja autenticar-se em um sistema. Para isso usaremos a técnica de criptografia de chave pública: um servidor sobre o qual o usuário deseja-se autenticar gera uma senha aleatória através de GNA; "encripta" esta senha usando a chave pública do destinatário (que será conhecida e poderá estar armazenada em uma base de dados) e a sua chave privada (do servidor), para decifrar esta senha o usuário usa a chave pública do servidor e a sua chave privada. Para "descriptor" a senha o usuário usará um telefone celular ou um PDA (*Personal Digital Assistants*), por exemplo.

A escolha de curvas elípticas está intrinsecamente ligada com o fato de usarmos equipamentos com pouco poder computacional para auxiliar o envio seguro da senha provisória para o usuário. Como visto na tabela 1 podemos ter criptografia segura usando aritmética de 160 *bits*. Se usássemos o RSA com segurança equivalente teríamos que usar inteiros de 1024 *bits*. Tal diferença é agravada quando se pretende implementar nestes tipos de dispositivos. Na verdade, usaremos estes equipamentos (celulares, PDAs) para tão somente efetuar os cálculos pertinentes a decifragem da senha.

O envio da senha "encriptada" para o usuário pode ser feito através de mensagem SMS (*Short Message Service*) visto que o PHP oferece este tipo de serviço. Caso o usuário deseje, ele pode optar por inserir manualmente a senha "encriptada" em seu celular, no entanto, isto não é o mais aconselhável, uma vez que esta senha pode ser formada de alguns longos dígitos. Este usuário também tem a opção de fazer o *download* desta senha "encriptada" através de um cabo transferidor de dados, por exemplo.

Os algoritmos de GNA e os algoritmos de criptografia podem ser construídos usando rotinas em PHP, no entanto, estas rotinas usam números grandes (160 *bits* se usarmos algum método baseado em curvas elípticas). Para utilizar números desta magnitude pode ser adicionada ao PHP a biblioteca (GNU *Multiple Precision Arithmetic*) (GMP) [GMP PHP]. Além de ser uma das mais rápidas bibliotecas para lidar com inteiros grandes, esta biblioteca dispõe de funções demasiadamente usadas em criptografia, como por exemplo exponenciação modular.

3. Escolha e Justificativa do Algoritmo

A escolha de um algoritmo baseado em curvas elípticas está intrinsecamente ligada com o fato de que estaremos usando equipamentos com pouco poder computacional, neste caso celulares. Algoritmos baseados em curvas elípticas proporcionam uma chave criptográfica consideravelmente menor que alguns clássicos da criptografia, tais como o RSA. Se estivéssemos usando o RSA precisaríamos de uma chave criptográfica de aproximadamente 1024 *bits*, no entanto se trabalharmos com curvas elípticas usaríamos uma chave de 160 *bits*, mantendo a mesma segurança. Este fato decorre que o Problema do Logaritmo Discreto (problema sobre o qual o algoritmo está baseado) sobre curvas elípticas é mais difícil de ser resolvido do que o Problema da Fatoração de Inteiros que é o problema que o RSA se baseia.

Existem vários algoritmos que trabalham sobre curvas elípticas, inclusive algoritmos que já existiam antes da descoberta desta técnica em 1985. Na verdade, os algoritmos que se baseavam no Problema do Logaritmo Discreto em grupos aditivos da forma Z_p podem ser estendidos para atuar sobre curvas elípticas. Um algoritmo que se destaca é o Menezes-Vanstone que, diferentemente dos algoritmos adaptados para trabalhar sobre curvas elípticas, não precisa codificar a mensagem em pontos da curva elíptica, facilitando assim a "encriptação" da mensagem. No nosso caso o servidor gerará uma senha aleatória. Se usarmos o criptosistema Menezes-Vanstone não teremos dificuldades em "encriptar" esta senha. No entanto, usando o protocolo Diffie-Hellman, por exemplo, antes de "encriptar" a mensagem deveríamos codificá-la em pontos de uma curva elíptica, gerando assim uma dificuldade a mais para a implementação.

4. Tecnologias Envolvidas

Nesta seção resumiremos as principais ferramentas e tecnologias envolvidas na execução do método proposto.

4.1 Biblioteca GMP

Como parte do processo de implementação do método proposto, é necessária a implementação do algoritmo criptográfico escolhido. É fato que existem poucas implementações para algoritmos baseados em curvas elípticas para PHP e que a documentação existente é precária – em muitas vezes não existe.

Com estes fatores sentimos a necessidade de implementar todas as operações pertinentes ao algoritmo escolhido (neste caso, o Algoritmo Menezes-Vanstone). Como precisamos trabalhar com inteiros relativamente grandes, uma ótima opção seria usar a biblioteca GMP [GMP PHP]. E esta oferece funções que são de suma importância para a implementações de criptografia. Como por exemplo a função `gmp_nextprime(int $a)` que retorna o próximo primo maior que `$a`. O trecho de código abaixo mostra o funcionamento da função

`gmp_invert(resouce $a, recouce $b)`. Esta função calcula o inverso de `$a` módulo `$b` e também terá grande utilidade na implementação do algoritmo.

```
<?php
$c1="74077570053047859662631461731530877308490
9391351"; $pri-
mo="116920130986472233456294786617302641572475
24989731";
$invMod=gmp_invert($c1,$primo);
echo gmp_strval($invMod);
?>
```

4.2. Java e biblioteca Bouncy Castle

A tecnologia Java é uma plataforma criada pela empresa Sun Microsystems na qual o desenvolvedor escreve programas através de uma linguagem de programação que gere código executável pela máquina virtual Java. O grande diferencial desse tipo de desenvolvimento é que, como os programas são escritos para uma máquina virtual, qualquer dispositivo que tenha uma máquina virtual Java instalada pode rodar os programas, independente de hardware ou sistema operacional.

A segmentação da tecnologia Java de interesse deste trabalho é a J2ME (Java 2 Micro Edition), que consiste basicamente em uma coleção de APIs (Application Programming Interface) voltadas ao desenvolvimento de aplicações para dispositivos com processamento limitado, como celulares e PDAs, e uma máquina virtual Java para rodar nesses dispositivos.

Basicamente, a arquitetura do J2ME divide-se em três camadas: máquina virtual, configurações e perfis. A máquina virtual é a camada de mais baixo nível e roda diretamente sobre o sistema operacional do dispositivo. Ela é quem executa os programas gerados para a plataforma Java e, portanto, é o que define as limitações dos programas que serão executados nos dispositivos. A camada acima da máquina virtual é a configuração e consiste basicamente em uma especificação que define o ambiente de software para uma faixa de dispositivos definida por um conjunto de características tais como tipo e quantidade de memória disponível, processador e sua frequência de operação e o tipo de conexão de rede disponível para o dispositivo.

Em termos práticos, uma configuração é formada por um conjunto de bibliotecas que estarão disponíveis para o desenvolvedor criar programas em uma faixa de dispositivos com aplicações distintas aplicações, mas com diversas características em comum. Como exemplo de faixa horizontal de dispositivos, pode-se citar os dispositivos móveis com conexão a rede sem fio. Nessa faixa estariam incluídos aparelhos como celulares, PDAs e pagers. A justificativa para a criação de configurações é que apesar dos diversos aparelhos serem distintos em forma e funcionalidades, muitos deles tem características de processadores e memória muito similares e, por isso, são colocados sobre uma mesma especificação que determina o nível de funcionalidades e serviços que tem que ser oferecidos pela máquina virtual.

A camada de mais alto nível na tecnologia J2ME é o perfil. Ele define um conjunto de bibliotecas específicas para o desenvolvimento de programas para uma faixa vertical de dispositivos e foca mais na aplicação e segmento de mercado do aparelho do que em suas características de processador e memória. Um exemplo de faixa vertical de dispositivos são os telefones celulares. Os perfis complementam uma configuração J2ME formando um serviço completo para que as aplicações possam ser executadas. Para clarificar a diferença entre perfis e configurações, é interessante lembrar que perfis são mais específicos que configurações. Assim, se pode ter uma configuração para dispositivos com baixo poder de processamento e um perfil para os celulares. A especificação de um perfil sempre é feita para uma determinada configuração, mas uma configuração pode suportar vários perfis diferentes. Como se objetivou que o sistema de autenticação proposto por esse projeto fosse utilizável em diversos contextos e por diversos usuários, usou-se o perfil MIDP sobre a configuração CLDC para a implementação do software que rodará no dispositivo móvel por essa ser a combinação de configuração/perfil presente em quase 100% dos celulares.

Para implementar o algoritmo de criptografia com curvas elípticas é necessária a capacidade de fazer contas com números de grandeza maior do que a grandeza dos números representáveis com os tipos primitivos do Java. Como entre outras de suas limitações, a biblioteca de classes do perfil MIDP não traz a classe `BigInteger` que permite a manipulação de números com a grandeza necessária. Fez-se imprescindível a utilização de uma biblioteca de classes que trouxesse consigo uma maneira de manipular números grandes o suficiente para a implementação do sistema. A biblioteca escolhida foi a Bouncy Castle, que é uma biblioteca de classes usada para criptografia que traz uma implementação compatível com J2ME da classe `BigInteger`. As diversas outras classes da biblioteca Bouncy Castle que poderiam ter sido usadas na implementação deste sistema de autenticação não foram usadas por motivos esclarecidos no item Detalhes de Implementação.

5. Detalhes de implementação

Tanto na implementação do software do dispositivo móvel quanto na do servidor foram criadas duas classes para abstrair os conceitos fundamentais envolvidos na criptografia com curvas elípticas: Curva e Ponto.

A classe Curva recebe através de parâmetros do seu construtor os coeficientes a e b além de um número primo p usado para definição do corpo \mathbf{Z}_p (a curva é definida por $y^2 = x^3 + ax^2 + b$). A principal dentre entre as suas funções é a função `multiplicacao()` que retorna um objeto do tipo Ponto resultante da multiplicação por escalar de Ponto P por um inteiro k . Esta função ocupa algo em torno de 92% do processamento na criptografia.

A classe Ponto recebe como parâmetros em seu construtor as coordenadas x e y de ponto e um valor `booleano` que indica se o Ponto é no infinito ou não.

Mais detalhes sobre criptografia baseada em curvas

elípticas poderão ser vistas em [Lara, P C S Oliveira, F B]. Suas outras funções são *getters* e *setters* para as coordenadas do Ponto e o valor *booleano* citado anteriormente.

No caso da implementação do software que rodará no dispositivo móvel, não foram utilizadas as classes de criptografia com curvas elípticas da biblioteca Bouncy Castle porque a documentação sobre elas é insuficiente. Assim, a única classe dessa biblioteca usada foi a versão para J2ME da classe `java.math.BigInteger`. Como não existe essa classe na configuração CLDC e no perfil MIDP é necessário usar um *obfuscator*, não apenas para os nomes das classes, mas também para o nome dos pacotes para usar essa classe, pois não é possível criar classes dentro do pacote Java por questões de segurança.

As chaves públicas dos usuários são pares de pontos armazenados em um banco de dados no servidor e as chaves privadas ficam apenas nos dispositivos dos usuários. A chave pública do servidor, composta por um par de pontos também é de conhecimento de cada dispositivo móvel.

6. Considerações Finais

No presente momento o método proposto neste trabalho está em fase de implementação e testes. Como trabalhos futuros, pretendemos abrir todos os códigos sob uma licença livre e estender este conceito para outros sistemas que necessitem de autenticação de usuários, além de melhorar o desempenho do programa Java que rodará nos dispositivos móveis.

Referências e links sugeridos

[GMP PHP] – <http://www.php.net/gmp>

Lara, Pedro Carlos da Silva; Oliveira, Fábio Borges. **Curvas Elípticas: Aplicação em Criptografia Assimétrica**. In: *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais 2007*. Rio de Janeiro. (WTICG), 2007.

Muchow, Jonh W. **Core J2ME: tecnologia & MIDP**. São Paulo: Pearson Makron Books, ISBN: 85-346-1522-5 2004.

Pereira, Evandro; Fagundes, Leonardo Lemes; Neukamp, Paulo; Ludwig, Glauco; Konrath, Marlom. **Forense Computacional: fundamentos, tecnologias e desafios atuais**. VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais.

Pedro Lara - pcslara@lncc.br

Graduando em Tecnologia da Informação pelo Instituto Superior de Tecnologia e em Matemática pela Universidade Federal Fluminense através do CEDERJ, atualmente atua na área de segurança da informação e criptografia assimétrica. Sua área de interesse é Criptografia Baseada em Curvas Elípticas.

Guilherme Gall - gmgall@lncc.br

Graduando em Tecnologia da Informação e da Comunicação pelo Instituto Superior de Tecnologia em Ciências da Computação de Petrópolis. Atualmente trabalha no Serviço de Redes do Laboratório Nacional de Computação Científica.

Fábio Borges - borges@lncc.br

Possui Bacharelado em Matemática pela Universidade Estadual de Londrina (UEL) e Mestrado em Modelagem Computacional pelo Laboratório Nacional de Computação Científica (LNCC). Atualmente é Tecnologista do LNCC onde é responsável pelo Setor de Treinamento e Apoio (STA). Tem experiência na área de Segurança da Informação. Atuando principalmente nos seguintes temas: Criptografia, Esteganografia, Web.



Criação de imagens *on-the-fly*

Uma importante característica da linguagem PHP é a possibilidade de criar imagens de forma dinâmica, isto é, durante a execução do *script*. Este artigo aborda a utilização da biblioteca GD do PHP para a criação de imagens, bem como um exemplo prático que permitirá a criação de palavras-passe para sítios da web.

A criação de imagens *on-the-fly*, ou seja, em tempo real de execução do *script*, é uma ferramenta poderosa da linguagem PHP para sítios da web, principalmente álbuns de fotografias, lojas virtuais e áreas administrativas.

Talvez a aplicação mais conhecida deste método de programação seja a criação de palavras-passe (palavras ou números aleatórios) em sítios de correio eletrônico, com o objetivo de evitar a atuação de *softwares* que criem múltiplas contas de e-mail. Em segurança, também é possível substituir os endereços de e-mail de páginas da Internet por imagens, para prevenir a atuação de ferramentas de captura de e-mails para bancos de dados de *spams*.

As aplicações deste método não se limitam apenas à segurança de sítios, sendo ele também amplamente utilizado em ferramentas de comércio virtual, onde o vendedor pode enviar uma imagem via formulário. Esta imagem pode ser processada pelo servidor para mudança de escala, formato e acréscimo de marca d'água, sendo depois armazenada em disco ou gravada em banco de dados.

1. Requisitos mínimos

Para o entendimento deste artigo, é necessário que o servidor disponha da biblioteca GD. A biblioteca GD é uma biblioteca em código aberto escrita em linguagem C. Desenvolvida por Thomas Boutell (<http://www.boutell.com>), a biblioteca é atualmente mantida por Pierre-A. Joye (<http://www.libgd.org>).

A biblioteca GD suporta os formatos JPG, PNG e GIF, dentre outros formatos. É importante salientar que o formato GIF é novamente suportado. Este formato era suportado pelas versões iniciais da biblioteca, mas foi descontinuado nas versões seguintes devido à patente da compressão LZW que expirou e, por isso, o suporte ao formato foi restabelecido na versão 2.0.28.

É possível verificar se o servidor dispõe da biblioteca,

procurando pela sessão GD no resultado da função **phpinfo()**. No entanto, se o objetivo é o desenvolvimento de um *script* portátil, que possa ser instalado em servidores diferentes, é interessante que a verificação seja realizada no próprio *script* através da função **function_exists()**. Ambas as possibilidades estão implementadas no *script* a seguir.

```
<?
if(function_exists("gd_info"))
    echo "Biblioteca GD instalada.";

phpinfo();
?>
```

GD Support	enabled
GD Version	bundled (2.0.28 compatible)
FreeType Support	enabled
FreeType Linkage	with freetype
GIF Read Support	enabled
GIF Create Support	enabled
JPG Support	enabled
PNG Support	enabled
WBMP Support	enabled
XBM Support	enabled

Figura 1 – Informações sobre a biblioteca GD instalada no servidor retornadas pela função **phpinfo()**

2. Criação da imagem

Depois de constatada a presença da biblioteca GD no servidor, o próximo passo para o *script* da palavra-passe é a criação de uma figura. Isto pode ser feito com as funções **imagecreatetruecolor()** e **imagecreate()**. Ambas recebem dois parâmetros (largura e altura) e retornam um identificador de recurso de imagem para a imagem criada. Nas duas funções, o identificador de recurso de imagem pode ser destruído através da função **imagedestroy()**. É conveniente a utilização do caracte-

re @ precedendo o nome das funções de imagem, para evitar que os erros ecoem na página.

Algumas diferenças podem ser citadas sobre as duas funções, como, por exemplo, a cor de preenchimento de fundo da figura, que é a primeira cor alocada quando se utiliza **imagecreate()** ou preta quando se utiliza **imagecreatetruecolor()**. Mas a diferença principal entre as funções **imagecreate()** e **imagecreatetruecolor()** é a chamada profundidade de *bits*.

A profundidade de *bits* é um valor que quantifica o número de cores únicas que podem ser formados dentro da paleta de cores de uma figura. As cores da paleta não precisam ser, necessariamente, utilizadas na figura. Imagens que não sejam *true color* podem possuir apenas 256 cores em sua paleta, enquanto que as imagens *true color* permitem a alocação de toda a paleta de cores *true color*, ou seja, 24 *bits* de profundidade de cor.

Cada *pixel* de uma figura é formado por uma combinação das três luzes básicas: vermelho, verde e azul (*red, green, blue* – RGB). É importante comentar aqui a diferença entre as três cores básicas e as três luzes básicas. Em física, as três luzes básicas são o vermelho, o verde e o azul e a união de suas intensidades forma todas as outras luzes. As três cores básicas fazem parte do mundo da arte e pintura e são caracterizadas pelos pigmentos vermelho, azul e amarelo. Os monitores funcionam através da emissão de misturas de luzes, as impressoras trabalham com misturas de pigmentos. Portanto, os *pixels* gráficos possuem luzes no padrão RGB.

Cada luz primária é chamada de “canal de cor”. Se cada canal de cor possuir profundidade de 8 bits, formando $2^{3 \times 8}$ combinações possíveis (16.777.216), então estaremos trabalhando no sistema de cores *true color* ou SVGA. Se além destas combinações, for adicionado ainda um canal de transparência, estará formado o sistema *true color* de 32 *bits*.

Agora que o sistema de cores já está entendido, é necessário alocar-se as cores para a paleta de cores da figura. A primeira cor alocada é utilizada para preencher a figura e para a paleta de cores nas imagens criadas com **imagecreate()**, e as cores alocadas na sequência fazem parte apenas da paleta de cores. As cores são alocadas através da função **imagecolorallocate()**, que recebe quatro argumentos (identificador de recurso da imagem, intensidade vermelho, intensidade verde e intensidade azul), retornando um identificador de recurso de cor.

As intensidades das luzes devem ser expressa em um valor de 8 *bits*, podendo ser utilizada a notação decimal (0 – 255) ou hexadecimal (0x00 – 0xFF). A cor é alocada apenas para a imagem à qual foi referenciada, devendo ser alocada múltiplas vezes no caso de múltiplas imagens abertas. A cor alocada não precisa ser utilizada na figura, porém estará ocupando espaço na paleta de cores. Esta cor pode ser retirada da paleta de cores através da função **imagecolordeallocate()**. Esta função recebe como argumentos o identificador de recurso da imagem e o identificador de recurso da cor. No entanto, se uma cor que foi utilizada na imagem for retirada da

paleta, todos os elementos com aquela cor receberão uma outra cor.

Também é possível alocar cores com transparência em imagens criadas com **imagecolorallocatealpha()**. A função recebe, além dos mesmos argumentos da **imagecolorallocate()**, um argumento adicional correspondente ao canal de transparência. Este canal possui profundidade de 7 *bits*, podendo ser utilizada a notação decimal (0 – 127) ou hexadecimal (0x00 – 0x7F). As cores alocadas com transparência também podem ser retiradas da paleta através da função **imagecolordeallocate()**.

Para verificar o número de cores utilizadas na paleta de uma imagem, utilize-se a função **imagecolorstotal()**, sendo necessário apenas informar o identificador de recurso da imagem desejada.

Além da alocação de cores sólidas e alocação de cores transparentes para a paleta, também é possível definir-se uma cor transparente para a imagem, com a função **imagecolortransparent()**. A transparência é uma característica da imagem, não da cor, isto significa que quando uma cor é declarada como transparente, todas as áreas preenchidas com aquela cor se tornarão transparentes.

4. Desenhando na imagem

Diversos elementos podem ser adicionados a uma figura, como linhas, quadrados, retângulos, círculos, elipses e polígonos genéricos.

A menor unidade geométrica é o ponto e, de forma análoga, a menor unidade gráfica é o *pixel*. A função **imagepixel()** permite que um dado *pixel* de uma imagem seja preenchido com uma cor, sendo necessário apenas definir a imagem, as coordenadas *X* e *Y* e a cor desejada. O sistema de coordenadas é baseado em um plano cartesiano com as abscissas normais e as ordenadas invertidas, ou seja, a imagem apresenta sua origem (coordenada 0,0) no canto superior esquerdo.

O elemento geométrico imediatamente superior ao ponto é a linha. Assim como a linha é geometricamente definida como uma sequência infinita de pontos alinhados, a linha gráfica é uma sequência finita de *pixels* alinhados. As linhas são desenhadas na figura através da função **imageline()**. A função requer o identificador de recurso de imagem, as coordenadas do ponto de início da linha, as coordenadas do ponto de término da linha e a cor.

Quadrados são elementos geométricos formados por quatro lados iguais e quatro ângulos iguais, enquanto os retângulos são formados também por quatro ângulos iguais, porém possuem dois pares de lados iguais. Portanto, os quadrados são, em verdade, casos particulares de retângulos, onde os dois pares de lados apresentam o mesmo valor. Retângulos e quadrados podem ser desenhados por meio das funções **imagerectangle()** e **imagefilledrectangle()**, sendo necessário definir o identificador de recurso de imagem, as coordenadas *X* e *Y* de um dos vértices, as coordenadas *X* e *Y* do vértice

oposto ao passado e a cor. Para se desenhar quadrados, é necessário que a diferença entre as coordenadas *X* e coordenadas *Y* dos dois vértices passados sejam iguais.

Circunferências são elementos geométricos que possuem todos os seus pontos a uma mesma distância de um ponto chamado centro. Círculos são formados pela circunferência e por todos os pontos internos a ela. As elipses são figuras geométricas cuja soma das distâncias de um ponto qualquer à dois pontos fixos chamados focos, é constante o ponto equidistante entre os dois focos é chamado de centro. Desta forma, circunferências são casos particulares de elipses. Portanto, toda circunferência é uma elipse, onde os dois focos e, conseqüentemente, o centro, pertencem a um mesmo ponto.

Elipses e circunferências podem ser desenhadas com as funções **imageellipse()**, que desenha uma elipse vazada, e **imagefilledellipse()**, que desenha uma elipse preenchida com a cor. Ambas as funções requerem o identificador de recurso da imagem, as coordenadas *X* e *Y* do centro da elipse, a largura e a altura e a cor. Para se desenhar círculos, deve-se apenas utilizar o mesmo valor para a largura e a altura da elipse.

Arcos podem ser desenhados com as funções **imagearc()** e **imagefilledarc()**. Ambas as imagens recebem como argumentos o identificador de recurso da imagem, as coordenadas do centro, a largura e a altura da elipse à qual o arco faz parte, os ângulos de início e final do arco e a cor. Os ângulos seguem a notação do eixo de coordenadas polares, possuindo o ângulo de 0° na posição das três horas e crescendo no sentido anti-horário.

A função **imagefilledarc()** ainda recebe um argumento extra: o estilo do preenchimento. Este argumento deve ser passado como uma máscara de OR dos *flags* **IMG_ARC_PIE**, **IMG_ARC_CHORD**, **IMG_ARC_NOFILL** e **IMG_ARC_EDGED**.

Os *flags* **IMG_ARC_PIE** e **IMG_ARC_CHORD** são mutuamente exclusivos, significando que não podem ser utilizados em conjunto. **IMG_ARC_PIE** desenha apenas o arco da elipse, enquanto que **IMG_ARC_CHORD** simplesmente une as duas extremidades do arco da elipse com uma linha. **IMG_ARC_NOFILL** produz apenas uma borda, ao invés de preencher a figura com a cor e **IMG_ARC_EDGED** une as extremidades do arco com o centro da elipse.

Caso seja necessário desenhar uma figura fechada formada apenas por vértices, pode-se utilizar as funções **imagepolygon()** e **imagefilledpolygon()**. Estas funções precisam de quatro argumentos: o identificar de recurso da imagem, um vetor de coordenadas *X* e *Y*, o número de vértices e a cor.

Os elementos gráficos a serem desenhados (com exceção do *pixel*) podem ter suas espessuras modificadas através da função **imagesetthickness()**. O padrão das linhas também pode ser modificado com a função **imagesetstyle()**, podendo ser utilizada para desenhar linhas pontilhadas ou tracejadas, por exemplo.

O preenchimento de uma área com uma cor é reali-

zado pelas funções **imagefill()** e **imagefilltoborder()**. Esta última função permite que seja determinado um espaço definido para o preenchimento.

5. Escrevendo na imagem

A biblioteca GD permite que caracteres e *strings* sejam impressos na figura através das funções **imagechar()**, **imagecharup()**, **imagestring()** e **imagestringup()**. Todas estas figuras trabalham com o mesmo conjunto de argumentos: o identificador de recurso da imagem, a fonte, as coordenadas *X* e *Y*, a string a ser impressa e a cor. No caso das funções **imagechar()** e **imagecharup()**, apenas o primeiro caractere da função é impresso. As funções **imagechar()** e **imagestring()** escrevem na figura na horizontal, enquanto que as funções **imagecharup()** e **imagestringup()** escrevem verticalmente na figura.

A fonte pode ser uma fonte interna ou uma fonte carregada externamente. As fontes internas são chamadas por meio dos números inteiros de 1 a 5. O carregamento de fontes externas não faz parte do escopo deste artigo. A figura 2 demonstra as diferenças entre as cinco fontes internas.

Fonte interna número 1
Fonte interna número 2
Fonte interna número 3
Fonte interna número 4
Fonte interna número 5

Figura 2 – Cinco fontes internas da função **imagestring()**.

6. Processamento de imagem

A modificação de imagens pré-existentes é, muitas vezes, mais importante que a criação de imagens novas. Uma fotografia de uma câmera digital de alta resolução pode, por exemplo, atingir 2048 x 1536 *pixels*. No entanto, se a imagem for mostrada dentro de um espaço de 400 x 300 *pixels* em uma página da Internet, muitos dos *pixels* da imagem não serão mostrados, embora a fotografia inteira deva ser carregada pelo *browser* antes de sua exibição. Para diminuir o tempo de abertura da página, o espaço utilizado em disco no servidor e, conseqüentemente, a banda transmitida, pode-se processar as imagens carregadas para diminuir suas dimensões.

A menor unidade geométrica é o ponto, caracterizado por ser uma estrutura unidimensional. Entretanto, a menor unidade visual possível em imagens é o *pixel*. A diferença entre o ponto e o *pixel* é que uma área é formada por infinitos pontos, enquanto que uma figura é sempre formada por um número finito de *pixels*. Isto significa que durante o processamento de imagens onde o número de *pixels* de uma figura é modificado, parte da informação associada àquela imagem será modificada, seja por interpolação (acréscimo de *pixels*) ou por decimação (extração de *pixels*).

Para o processamento de imagens pré-existentes, é necessário abrir a imagem utilizando as funções:

```

imagecreatefromgif();
imagecreatefromjpeg();
imagecreatefrompng();
imagecreatefromwbmp();
imagecreatefromxbm();
imagecreatefromxpm().

```

Todas estas funções requerem apenas uma *string* com o nome do arquivo a ser aberto. Depois de aberta a imagem, é interessante saber o tamanho da imagem, o que pode ser conseguido com as funções **imagesx()** e **imagesy()**, passando apenas o identificador de recurso de imagem.

Se duas ou mais imagens estiverem abertas ao mesmo tempo, pode-se copiar parte de uma imagem para outra. Para este fim, pode-se utilizar as funções **imagecopy()**, **imagecopymerge()**, **imagecopymergegray()**, **imagecopyresampled()** e **imagecopyresized()**.

A diferença básica entre estas é que a função **imagecopy()** realiza uma cópia simples de uma parte de uma imagem em outra imagem; **imagecopymerge()** funciona de forma semelhante à **imagecopy()**, exceto por ser possível definir um grau de transparência na porção copiada da imagem de origem; **imagecopymergegray()** é semelhante à função anterior, exceto por transformar os *pixels* de destino em tons de cinza antes da mescla das imagens; a função **imagecopyresampled()** e **imagecopyresized()** realizam a mesma tarefa, copiar uma parte de uma imagem em outra imagem com escalas diferentes, exceto que **imagecopyresampled()** realiza uma interpolação na figura, possuindo resultados visuais levemente mais agradáveis que **imagecopyresized()**.

Para rotacionar a imagem em certo número de graus, é possível utilizar-se a função **imagerotate()**. Também é suportada a correção de gama através da função **imagegammacorrect()** e a aplicação de filtros à imagem com **imagefilter()**.

6. Finalização da imagem

Quando todo o processamento da imagem já tiver sido realizado, o próximo passo é a finalização da imagem. A imagem pode ser mostrada na tela ou salva em arquivo. Para isto, deve-se utilizar uma das seguintes funções, dependendo do formato final desejado:

```

imagegif();
imagejpeg();
imagepng();
imagewbmp();
imagexbm().

```

Todas as funções listadas acima requerem como argumento o identificador de recurso de imagem e um argumento opcional do nome do arquivo. Se este argumento for passado, a imagem será salva no arquivo especificado, caso contrário, a imagem será enviada ao

browser no formato especificado. Caso seja selecionada a opção para exibição em *browser*, é importante o uso da função **header()** para indicar ao *browser* que o conteúdo se trata de uma imagem.

7. Script da palavra-passe

Agora que as funções da biblioteca GD já foram vistas, o *script* da palavra-passe pode ser desenvolvido seguindo os passos abaixo:

- Verificar requisitos mínimos;
- Criar uma imagem;
- Criar a paleta de cores;
- Desenhar linhas aleatórias;
- Escrever a palavra-passe aleatória;
- Rotacionar a imagem;
- Finalização da imagem.

Os passos enumerados acima estão indicados no *script* abaixo.

```

<?
// Requisitos mínimos
if(!function_exists("gd_info"))
{
    echo "Biblioteca GD não instalada.";
    exit();
}
// Criar a imagem
$image = @imagecreate(100,100);
// Criar a paleta de cores
$branco = @imagecolorallocate
($image,0xFF,0xFF,0xFF);
$azulescuro = @imagecolorallocate
($image,0x00,0x00,0x80);
$verde = @imagecolorallocate
($image,0x00,0xFF,0x00);
$verdeescuro = @imagecolorallocate
($image,0x00,0x80,0x00);
$vermelhoescuro = @imagecolorallocate
($image,0x80,0x00,0x00);
// Desenhar linhas aleatórias
for($i = 0;$i < 3;$i++)
    @imageline($image,rand(0,99),rand
(0,99),rand(0,99),rand(0,99),$vermelhoescuro);
for($i = 0;$i < 3;$i++)
    @imageline($image,rand(0,99),rand
(0,99),rand(0,99),rand(0,99),$verdeescuro);
for($i = 0;$i < 3;$i++)
    @imageline($image,rand(0,99),rand
(0,99),rand(0,99),rand(0,99),$verde);
// Escrever palavra-passe aleatória
for($i = 0;$i < 4;$i++)
    @imagestring($image,6,32 + 10 * $i,32 +
rand(-5,5),rand(1,9),$azulescuro);
// Rotacionar a imagem
$image = @imagerotate($image,90,$branco);
// Finalização da imagem
@imagepng($image);
?>

```

Considerações finais

Espera-se que este artigo tenha ilustrado o modo de funcionamento da biblioteca GD, caracterizando uma poderosa ferramenta para edição de imagens dinamicamente. O artigo demonstrou a criação de uma imagem, bem como seu processamento para a criação de palavras-passe. O *script* demonstrado pode ser modificado para ser utilizado em sistemas de segurança de *login* em sítios da Internet.

Referências e links sugeridos

[PHP Manual] – <http://www.php.net>

Leandro Schwarz - leandroschwarz@gmail.com

Engenheiro eletricista pela Universidade Federal de Santa Catarina (UFSC). Atuando desde 2000 com desenvolvimento WEB, possui sólidos conhecimentos em PHP e MySQL.

Atualmente é mestrando em Engenharia Elétrica no Instituto de Engenharia Biomédica da UFSC. Projeta *websites* e lojas virtuais como autônomo.

Amplie os horizontes da sua empresa

Divulgue sua empresa e produtos para a comunidade PHP do Brasil.
Conheça os planos para anunciar no portal e na revista.
Solicite um orçamento através de comercial@phpmagazine.com.br.

Quinta edição!

Participe da quinta edição da PHP Magazine. Visite nosso portal e veja as informações para submissão de artigos para a revista. Não esqueça o deadline: 25/04/2008.

Visite nosso Portal



www.phpmagazine.com.br