Recomendações de Segurança para Desenvolvimento de Aplicações Web

Índice

1. Introdução

As aplicações web disponibilizadas na rede, sejam elas sites, *webservices* ou sistemas acessíveis via rede, podem ser acessadas por quase quaisquer usuários que possuam conexão com a Internet. Essa facilidade de acesso possibilita o ataque indiscriminado a tais serviços, ocasionando, em determinados casos, diversos problemas, tais como: divulgação de dados confidenciais, indisponibilidade do serviço por tempo indeterminado, denigração da imagem da organização, etc.

Ainda baseado na afirmativa do *Gartner Group*, que informa que mais de 70% dos *cyber* ataques ocorrem em aplicações web, e baseado na afirmativa do *WhiteHat Security*, onde 8 em 10 sites da web possuem sérias vulnerabilidades, a necessidade de aumentar a segurança de tais aplicações torna-se essencial.

Infelizmente, a maioria dos desenvolvedores só se preocupa com a segurança da aplicação após algum problema relacionado à mesma, querendo aplicar todas as medidas preventivas somente depois que o software está desenvolvido. Dessa forma, infringe-se uma das primeiras premissas para aumentar a segurança da aplicação: pensar em segurança desde o início do desenvolvimento do software.

Pensando nisso, foi desenvolvido este documento para servir de guia básico na tarefa de garantir níveis de segurança maiores para sistemas web.

1.1 Controle de Versão

Versão	Autor	Comentário
1.0	USC - ATI	Criação do documento
1.1	USC - ATI	Revisão do documento

1.2 Objetivo

O propósito principal deste documento é dar uma visão geral das principais falhas encontradas em aplicações web e apresentar contramedidas para solucionar cada um dos problemas listados. Dessa forma, foge do escopo deste documento entrar em detalhes sobre cada uma das falhas apresentadas ou ainda servir como um manual para realização de ataques. Na seção de referências existem vários *links* para documentos que tratam com maiores detalhes as vulnerabilidades aqui apresentadas.

1.3 Público - Alvo

Como público-alvo inclui-se todos os envolvidos no desenvolvimento de aplicações do Governo do Estado de Pernambuco, desde programadores a gerentes.

2 Vulnerabilidades Comuns

Nesta seção são apresentadas as falhas comumente encontradas em aplicações web, exibindo a sua descrição e as respectivas recomendações para evitá-las.

2.1 Injeção de SQL

O objetivo deste ataque é inadvertidamente consultar, inserir, remover ou alterar os dados do banco de dados. Para isso, o atacante insere comandos SQL através dos parâmetros de entrada da aplicação, para serem concatenados com o código SQL original, e assim, executar outro comando SQL. O atacante, geralmente, utiliza formulários que não realizam tratamento na entrada de dados dos usuários para injetar SQL no código da aplicação. Abaixo é exibido um exemplo de uma aplicação imaginária, escrita em ASP utilizando o banco de dados SQL Server, que realiza a seguinte consulta para autenticação dos usuários:

```
Select * From Usuario Where login='"& login &"' And senha = '"& senha &"'"
```

O atacante poderia inserir nos campos *login* e senha os seguintes valores respectivamente, "**' or 1=1 --**" e "**123**", resultando no código:

```
Select * From Usuario Where login='' or 1=1 -- ' And senha = '123';
```

Dessa forma, a aplicação executará uma consulta SQL que retornará um conjunto com todos os usuários do sistema, pois a expressão "**or 1=1**" torna o comando verdadeiro e a expressão "**--**" comenta o restante do código. Como vários sistemas de autenticação utilizam apenas o primeiro usuário da lista retornada, que freqüentemente é um administrador do sistema, o atacante então seria autenticado no sistema com as credencias deste super-usuário.

2.1.1 Recomendações

Para contornar esse problema, a principal recomendação é realizar uma validação de todos os dados de entrada, principalmente no lado do servidor, a fim de evitar a utilização de caracteres maliciosos. A validação de dados realizada somente no lado do cliente pode ser facilmente contornada copiando a página e removendo o código de validação, geralmente feito em *javascript*.

A política de validação recomendada a ser implementada é a de permitir somente caracteres previamente selecionados e negar todo o resto das entradas. Isso pode ser alcançado, por exemplo, através do uso de expressões regulares.

Finalmente, recomenda-se utilizar P*repared Statement* ou *Stored Procedures* nos comandos SQL. O principal objetivo é impedir que os dados de entrada afetem a sintaxe do comando SQL, separando a lógica do código dos dados informados. Além de aumentar a segurança, a aplicação é também beneficiada com um aumento no desempenho do comando a ser executado.

2.2 Cross Site Scripting

Cross site scripting (geralmente referenciado por XSS) ocorre quando um atacante utiliza uma aplicação web para enviar código malicioso que será acionado posteriormente por uma ação de um usuário regular. O código malicioso está, na maioria das vezes, sob a forma de script. O ataque já está bastante difundido e ocorre em qualquer aplicação web que utilize a entrada do usuário, sem validação, na saída gerada pela aplicação.

Um possível ataque, utilizando esta vulnerabilidade, é a descoberta do *cookie* de sessão do usuário, permitindo ao atacante seqüestrar tal sessão e ter o controle total de sua conta. Outros ataques incluem a divulgação de arquivos de usuários, a instalação de cavalos de tróia, o redirecionamento do usuário para outra página ou site e a modificação do conteúdo de páginas.

2.2.1 Recomendações

O ponto principal para prevenir esta vulnerabilidade nas aplicações é a confirmação de que o conteúdo de páginas dinamicamente geradas não contenha *scripts* indesejados. Isso pode ser alcançado filtrando-se toda a entrada do usuário na aplicação, incluindo *cookies*, *urls* e dados transmitidos via POST e GET. Além disso, é interessante também filtrar a saída do servidor web para o usuário, pois um atacante poderia inserir um script hostil diretamente no banco de dados ou em um momento em que a aplicação não possuía filtragem dos dados. Dessa forma, qualquer dado persistente que seja transmitido entre o navegador e o servidor web deve ser filtrado, a fim de atingir todo o escopo do problema.

Para realizar a filtragem dos dados recomenda-se utilizar uma abordagem positiva, que consiste em negar todas as entradas com exceção

dos dados previamente escolhidos. Por exemplo, não há necessidade de um campo do tipo data no formulário receber entradas com valores diferentes de números e talvez barras '/' (para separar dia, mês e ano). Dessa forma, o desenvolvedor não terá que adivinhar ou atualizar todas as formas de entradas maliciosas para negar todos os caracteres específicos.

A abordagem positiva da filtragem de dados é a recomendada pois existem várias formas de representar o mesmo caractere. Por exemplo, a seguinte tabela abaixo lista algumas formas de representar o caractere '<', que dependendo do contexto pode ser hostil.

Representação do caractere '<'			
%3C	&#0060	<	&#x003c
&lt	&#00060	<	&#x0003c
<	&#000060	<	&#x00003c
&LT	&#0000060	<	&#x000003c
<	<	&#x3c	<
&#060	<	&#x03c	…

2.3 Ataques a sistemas de autenticação

Para garantir a acessibilidade das informações nos sistemas por pessoas que possuam a autorização para tal, várias aplicações implementam um controle de autenticação, obrigando o usuário a informar *login*/senha válidos para acessar o sistema. Apesar desse controle, freqüentemente os desenvolvedores dos sistemas exibem informações de erros detalhadas, na tentativa de ajudar os usuários, como por exemplo, 'Usuário não cadastrado' ou 'Senha inválida'. Abaixo são listados os principais problemas provenientes dessas mensagens de erro:

- **Enumeração de *logins* válidos:** Nesta técnica vários usuários válidos do sistema poderão ser listados, facilitando o ataque de dicionário ou de força bruta. Isso é possível devido à mensagem de erro informar se determinado usuário existe ou não no sistema.
- **Ataque de dicionário:** Esta técnica utiliza a política da tentativa e erro baseado num dicionário de palavras, na qual o atacante poderá utilizar este conjunto de palavras nos campos de *login* e senha a fim de se autenticar no sistema. Com a descoberta de *logins* válidos (item anterior), o processo de descoberta da senha reduz-se drasticamente.
- **Ataque de força bruta:** Esta outra técnica baseia-se na mesma idéia do ataque de dicionário, mas ao invés de utilizar um dicionário de palavras, utiliza todas as possibilidades de formações de palavras, elevando muito o tempo de descoberta de pares de usuário/senha válidos. Da mesma forma que o item anterior, caso o atacante possua *logins* válidos o tempo gasto neste processo de autenticação será reduzido consideravelmente.

2.3.1 Recomendações

O primeiro passo é evitar mensagens de erros detalhadas, como nos caso exibidos. As mensagens de erros devem ser específicas o suficiente para informar o problema ao usuário, e restritas o máximo para não disponibilizar informações que facilitem o ataque de pessoas mal intencionadas. Para que sejam evitados ataques de dicionário ou de força bruta, pode-se utilizar um controle de *logins* mal sucedidos, estipulando-se um limite na quantidade de erros de *login*, bloqueando o usuário por um período de tempo ou pode-se utilizar *CAPTCHA*[1]. Abaixo é exibido um exemplo do uso de CAPTCHA, na qual só aparece após algumas tentativas erradas.

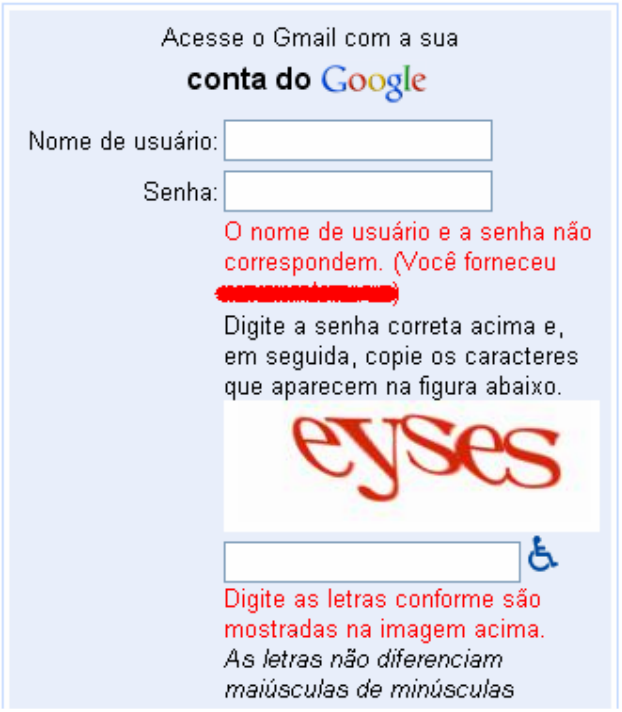

Figura 1 – Exemplo de CAPTCHA do site www.gmail.com

2.4 Seqüestro de sessões

A fim de garantir um controle de autenticação dos usuários na aplicação, muitos sistemas web utilizam *cookies* para guardar um identificador único do usuário, evitando assim, que o usuário informe as suas credenciais para cada solicitação enviada ao servidor. Por exemplo, quando um usuário se autentica em uma aplicação web, o sistema valida seu *login* e senha e associa um *cookie* ao usuário solicitante. Quando o usuário solicita outra página, ao invés de informar novamente o *login* e senha, o servidor captura o *cookie* e verifica qual usuário possui o identificador informado, carregando a página solicitada no

1 CATPCHA é um mecanismo para geração de testes com a finalidade de verificar se o usuário é uma pessoa ou um sistema, na qual geralmente são utilizadas imagens contendo letras e números e um campo onde o usuário informa o conteúdo da imagem.

caso do identificador estar associado a um usuário ou negando o acesso em caso contrário.

O ataque pode surgir quando captura-se ou "adivinha-se" o *cookie* de outro usuário. Então, torna-se possível acessar a aplicação e se passar pelo usuário que está associado àquele *cookie* capturado. Isso pode acontecer de várias maneiras:

- Através de outros ataques que consigam capturar o *cookie*, como por exemplo, *Cross Site Scripting*.
- Caso a comunicação dos dados não seja criptografada, ou seja, caso seja utilizado HTTP ao invés de HTTPS (HTTP sobre SSL).
- Se o *cookie* estiver armazenado no *cache* do navegador.

2.4.1 Recomendações

Apesar do uso de HTTPS garantir a criptografia dos dados transmitidos entre o cliente e o servidor, ainda é possível capturar o *cookie* dos usuários através de outro ataque, neste caso, o *Cross Site Scripting*. Portanto, a eliminação dessa vulnerabilidade já diminui consideravelmente o risco de ocorrer um seqüestro de sessão.

Além disso, ainda são recomendados alguns itens:

- Utilização de um valor pequeno para o tempo de expiração do *cookie*.
- Evitar utilizar *cookies* persistentes, impedindo assim que um atacante o roube, caso tenha acesso físico ao computador do usuário.
- Separar os *cookies* de autenticação dos *cookies* de personalização, que são utilizados para armazenar as preferências dos usuários.

2.5 Visualização de arquivos indevidos

Algumas aplicações mantêm arquivos em locais impróprios ou mantêm arquivos que não deveriam estar na aplicação. Alguns exemplos do primeiro tipo são arquivos de configuração e informações sigilosas. Exemplos do segundo tipo são: arquivos com senhas (da aplicação ou do banco de dados), arquivos com o código-fonte acessível. Em ambos os casos a divulgação destas informações pela aplicação pode tornar a aplicação suscetível a outros ataques ou permitir que dados confidenciais sejam disponibilizados.

2.5.1 Recomendações

A recomendação é mover os arquivos de configuração (ex: .cfg, .conf, .ini, etc.), de backup (ex: .bak, .old, etc.) e sigilosos (ex:, .class, .log, .xls, .pdf, .mdb, .tar.gz, etc.) para uma pasta que não seja acessível pela web e implementar na aplicação um controle de acesso aos arquivos sigilosos.

Caso a disponibilização de arquivos sigilosos seja necessária, recomenda-se que a própria aplicação ofereça essa funcionalidade, respeitando a autenticação e a autorização do usuário solicitante. Os demais tipos de arquivos, configuração e backup, não devem ser acessados via web.

2.6 Execução de comandos no Sistema Operacional

Esta técnica de ataque permite a execução de comandos do Sistema Operacional através da manipulação nas entradas de dados da aplicação. Isso é possível quando a aplicação não valida corretamente a entrada do usuário antes de usá-la no sistema. Dessa forma, todos os comandos irão executar com as mesmas permissões do serviço que executou o comando, seja ele o servidor web, o banco de dados, etc.

Algumas aplicações web incluem parâmetros informando um arquivo que será exibido para o usuário. Caso não seja feita a validação de entrada informada pelo usuário, um atacante pode modificar o valor do parâmetro executando um comando do sistema operacional. Os exemplos abaixo exibidos são de uma aplicação fictícia rodando sobre um sistema operacional baseado em Unix. Essa aplicação possui uma página para exibição de arquivos solicitados pelo usuário, de modo que o nome do arquivo é informado no parâmetro *arquivo* da url. Dessa forma, o atacante poderia trocar o valor dessa variável por um código malicioso. O código original é exibido abaixo:

```
http://exemplo/teste.php?id=15&arquivo=relatorio.pdf
```

Alterando-se o valor para "`; rm -r *`", o atacante consegue remover todos os arquivos e diretórios do diretório corrente e abaixo dele.

```
http://exemplo/teste.php?id=15&arquivo=; rm -r *
```

Um outro modo, que algumas aplicações web utilizam são as funções *exec*, que permitem a execução de comandos do sistema operacional. Caso a aplicação permita a introdução de dados pelo usuário que sejam usados em tais funções sem a devida validação, um atacante poderia executar comandos do sistema operacional remotamente.

2.6.1 Recomendações

Como estas funções podem oferecer falhas críticas à infra-estrutura da empresa, recomenda-se que todos os dados provindos do usuário e utilizados como parâmetro nestas funções sejam validados antes de serem executados. Também utilize, se possível, chamadas a bibliotecas ao invés de processos externos (*exec()*, *system()*, etc.) com a finalidade de recriar a funcionalidade desejada. Além disso, certifique-se que a aplicação é executada com o mínimo

de privilégios possível para funcionar corretamente, a fim de restringir o efeito da execução de códigos maliciosos.

Caso a aplicação não faça uso de tais funções e caso seja possível, recomenda-se o bloqueio da execução das chamadas de sistema através da configuração do servidor web.

2.7 Elevação de privilégios

Nesta vulnerabilidade, a idéia central é a de adquirir mais permissões que o atribuído ao usuário do sistema. Para evitar que usuários indevidos acessem recursos das aplicações web, estas comumente utilizam um controle de autorização para saber se determinado usuário possui permissão de acesso ao recurso solicitado.

Esta vulnerabilidade pode ser encontrada em aplicações web que realizam esse controle apenas através de menus dinamicamente criados, específicos para cada grupo de usuários, não verificando a requisição no lado do servidor, permitindo assim que o atacante acesse uma página, proibida para ele, bastando apenas informar o endereço diretamente na barra de endereços.

2.7.1 Recomendações

Todo controle deve ser validado, no mínimo, no lado do servidor, podendo também ser validado no lado do cliente, para efeito de apresentação e otimização da aplicação. Qualquer recurso protegido do sistema só poderá ser acessado através de usuários devidamente autenticados e autorizados. Portanto, revise o código da aplicação e verifique se todos os recursos protegidos possuem tal controle implementado. Utilize ainda o conceito do *privilégio mínimo*[2], para executar os processos que mantêm a aplicação.

3 Outras Recomendações

Além dos itens apresentados anteriormente, outros itens com o mesmo grau de importância devem ser levados em conta. Abaixo, são exibidos os pontos a serem verificados com sua respectiva descrição.

3.1 Utilização do protocolo HTTPS

O protocolo HTTP não oferece grau algum de segurança para a comunicação de dados. Com a ampliação da internet e devido à necessidade

2 Privilégio mínimo representa o conceito que utiliza o mínimo de permissões necessárias para efetuar o serviço requisitado.

de tráfego de dados seguros, foi implementado HTTPS[3] para prover vários serviços de segurança para as aplicações e usuários das mesmas.

Os principais serviços oferecidos são: confidencialidade[4] da informação entre o servidor e o cliente através da criptografia e a autenticação[5] do servidor para o cliente através de certificados digitais.

3.2 Política de senhas

Um dos elos fracos de qualquer aplicação que necessite de autenticação são as senhas. Vários usuários utilizam senhas fracas, possibilitando que um ataque de força bruta descubra um par de *login* / senha válido.

Para contornar esse problema, recomenda-se que o sistema implemente uma política de senhas, impedindo que senhas fracas sejam admitidas. Como exemplo, pode-se requisitar que o usuários informe uma senha com no mínimo 8 caracteres, utilizando letras maiúsculas, minúsculas, símbolos e números, sendo trocada periodicamente. Para maiores informações sobre políticas de senhas, veja o folder *Dicas de Segurança: Escolhendo e protegendo suas senhas* – USC / ATI.

3.3 Upload de arquivos

Vários sistemas permitem aos seus usuários enviar arquivos para a aplicação com diversas finalidades, dentre elas a centralização de documentos no sistema, a inclusão de arquivos no e-mail, o armazenamento remoto de arquivos, etc. Apesar dos benefícios, esta funcionalidade geralmente apresenta algumas vulnerabilidades sérias ao sistema.

O principal ponto a ser verificado é a extensão do arquivo enviado ao servidor. Extensões como .exe, .bat, .cmd, .src, .dll, .vb, .vbs, .asp, .php, .js, .jsp, etc. são tidas como suspeitas e devem ser bloqueadas pela aplicação. A abordagem normalmente utilizada é permitir somente as extensões previamente selecionadas e negar todas as demais extensões.

Ainda é recomendado utilizar uma partição específica para o armazenamento dos arquivos enviados, que seja diferente da partição do sistema operacional e do servidor web, com isso evita-se a falta de espaço para os arquivos do sistema operacional e o acesso direto aos arquivos pela web, respectivamente.

Para completar, caso o sistema permita, especifique uma quota de tamanho total por usuário para o envio de arquivos, impossibilitando que usuários mal-intencionados acabem todo o espaço da partição destinado para o *upload*.

3 HTTPS (HyperText Transfer Protocol Secure) é o protocolo HTTP sobre SSL ou TLS.

4 A confidencialidade diz que a informação só está disponível para aqueles devidamente autorizados.

5 Autenticação é a capacidade de garantir que um usuário é de fato quem ele diz ser.

3.4 Privilégios mínimos no Banco de dados

Hoje, várias aplicações estabelecem uma conexão com um banco de dados e por conseqüência requerem um usuário e senha para acessar este repositório. Infelizmente, muitos desses usuários são administrativos, permitindo que operações não necessárias à aplicação possam ser executadas.

Como boa prática de segurança, recomenda-se utilizar um usuário de banco de dados específico para a aplicação, com o mínimo de privilégios necessários para rodar o sistema completamente, impedindo dessa forma que funções desnecessárias possam ser executadas na base de dados da aplicação ou até mesmo que outras bases de dados possam ser afetadas, nos casos em que o atacante consiga ter acesso às credenciais do banco de dados ou consiga injetar comandos SQL.

3.5 Criptografia das senhas

Este item refere-se às senhas armazenadas nos bancos de dados utilizadas em sistemas de autenticação da aplicação. O problema principal da guarda de senhas em texto plano é a possibilidade de alguém utilizá-las indevidamente na aplicação em questão ou em outras aplicações, nos casos de usuários que possuem a mesma senha em diversos sistemas.

Por isso, recomenda-se utilizar um algoritmo de *hash*[6] de acesso público, amplamente utilizado e testado, a fim de garantir uma maior segurança às senhas da aplicação. Para isso, podem ser utilizados alguns algoritmos de amplo reconhecimento e disponíveis em várias linguagens de programação, que podem ser utilizados para alcançar essa necessidade. Dentre tais algoritmos, pode-se listar: **RIPEMD-160, SHA256**, **SHA384** e **SHA512**, ordenados pelo grau de segurança, do menor para o maior. Não utilize os algoritmos **MD5** e **SHA1**, já que é possível quebrar os mesmos.

3.6 Campos hidden no código HTML

Algumas aplicações utilizam campos escondidos no código HTML para controle interno da aplicação, seja para armazenar uma opção do usuário ou até para definir o nível de acesso do usuário. Independente da finalidade pretendida pelo campo, uma pessoa mal intencionada poderia modificar o valor deste parâmetro facilmente, permitindo assim situações desastrosas.

Para evitar essa situação, verifique o objetivo destas variáveis "escondidas" e tenha certeza de que caso sejam modificadas não afetarão a segurança da aplicação.

[6] O algoritmo de hash gera, a partir de uma entrada qualquer, uma mensagem de tamanho fixo, única e irreversível. Por única, entende-se que duas entradas diferentes dificilmente terão o mesmo hash (resultado do algoritmo).

3.7 Atualização de softwares

A atualização dos softwares que oferecem o suporte a aplicação é tão importante para a segurança da aplicação, do servidor e da rede quanto às demais recomendações apresentadas neste documento. Dessa forma, verifique por atualizações no site do desenvolvedor dos softwares utilizados pela sua aplicação para evitar que vulnerabilidades alheias à sua aplicação sejam utilizadas.

3.8 Configuração de softwares

Manter os softwares atualizados não basta. A configuração correta é essencial para evitar problemas futuros. Um dos erros cometidos com freqüência é a utilização da mesma configuração do ambiente de desenvolvimento para o ambiente de produção.

Determinadas configurações são próprias para cada ambiente, como por exemplo, a ativação de debug. Esta opção só deve estar habilitada no ambiente de desenvolvimento, para efeito de manutenção da aplicação, evitando-se que possíveis informações sensíveis sobre a estrutura do aplicativo sejam divulgadas.

Diversos aplicativos fornecem uma senha padrão para administração e testes. Um dos primeiros passos da configuração do software é modificar a senha inicial ou padrão.

4 Referências

- **Tempest technologies**
  http://www.tempest.com.br/

- **OWASP (The Open Web Application Security Project). Vários artigos, ferramentas, etc. sobre segurança em aplicações web.**
  http://www.owasp.org/

- **Spett, Kevin. SQL Injection: Are your web applications vulnerable?**
  http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf#search=%22white%20paper%20sql%20injection%22

- **Friedl, Steve. SQL Injection Attacks by Example**
  http://www.unixwiz.net/techtips/sql-injection.html

- **Spett, Kevin. Blind SQL Injection: Are your web applications vulnerable?**
  http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf#search=%22spi%20dynamics%20blind%20sql%20injection%22

- **Cross Site Scriptting FAQ**

http://www.cgisecurity.com/articles/xss-faq.shtml

- **Ollmann, Gunter. Paper: HTML Code Injection and Cross-site scripting**
  http://www.technicalinfo.net/papers/CSS.html

- **Denial of Service Attacks**
  http://www.cert.org/tech_tips/denial_of_service.html

- **The CAPTCHA Project**
  http://www.captcha.net/