
# Boas Práticas de Desenvolvimento Seguro

Julho / 2.012

## Histórico de Revisões

| Data | Versão | Descrição | Autor |
| --- | --- | --- | --- |
| 29/07/2012 | 1.0 | Versão inicial | Ricardo Kiyoshi |
| | | | |
| | | | |
| | | | |

# Conteúdo

# 1. Segurança da Informação em Sistemas e Aplicações

Aplicações são softwares desenvolvidos para serem importantes ferramentas aos processos de negócio de uma empresa. A segurança dessas aplicações possui três aspectos principais que devem ser sempre considerados: a infra-estrutura tecnológica, os aspectos humanos e a própria aplicação. Serão abordados neste documento aspectos de segurança relacionados á própria aplicação.

# 2. Aspectos Culturais no Desenvolvimento de Aplicações

Algumas características culturais criadas ao longo de toda a evolução da área de desenvolvimento de softwares e aplicações influenciam na segurança das aplicações desenvolvidas.

O foco e a motivação de um desenvolvedor são em grande parte o desenvolvimento de funcionalidades, facilidades de uso, interfaces amigáveis e tudo mais que poderá automatizar e facilitar as atividades dos usuários finais. Outro ponto de preocupação constante do desenvolvedor é a atualização tecnológica. Ferramentas e tecnologias de desenvolvimento evoluem constantemente em grande velocidade obrigando os desenvolvedores a estarem sempre estudando e pesquisando as novas tecnologias. Os prazos também influenciam no desenvolvimento.

Na maioria das vezes, os prazos para o desenvolvimento das aplicações são definidos pela área comercial para atender as demandas do mercado. Mesmo em desenvolvimento interno, muitas vezes o prazo é estipulado em função da necessidade do negócio do cliente interno. Esta pressão por prazos, aliada à falta de práticas de técnicas de engenharia e processo de software leva muitas vezes á existência de vulnerabilidades de segurança na aplicação.

# 3. Aplicações Web

As aplicações web têm uma importante característica que requer cuidados adicionais no tratamento da segurança. As aplicações web são executadas em cima de padrões abertos que são amplamente utilizados pelo mercado. Basicamente podemos separar uma aplicação web em 3 camadas distintas: a estação, o servidor e o banco de dados.

A estação possui um navegador web que recebe e envia os dados para o servidor web. O servidor recebe as requisições e os dados enviados pela estação, processa-os e utiliza um banco de dados para armazenar e recuperar as informações.

A proteção da aplicação web tem que ser implementada no servidor e no banco de dados. Todo o conteúdo que é enviado e processado pelo navegador da estação está sujeito à alteração e manipulação. Enviar informações confidências à estação e procurar garantir sua proteção através dos recursos do navegador é muito perigoso, pois o formato como essas informações são transmitidas para o navegador é padronizado e de conhecimento público. Um usuário malicioso poderá simplesmente substituir o navegador por outro navegador que não tenha esses recursos de segurança ou mesmo desenvolver seu próprio navegador acessando e manipulando as informações escondidas.

# 4. Identificação do Grau de Segurança para a Aplicação

A segurança numa aplicação não é algo que se possa ver com facilidade. Diferentemente de outros requisitos como performance, quantidade de falhas e interface com usuário; a segurança na aplicação é difícil de quantificá-la e qualificá-la.

A segurança não é um atributo no qual podemos simplesmente assumir que temos ou não temos. Praticamente toda aplicação tem um nível de segurança

implementado que pode ser melhorado. Por outro lado, por mais que desenvolvemos e investimos em segurança da aplicação sempre existirá algum recurso ou tecnologia disponível para aumentar a segurança existente. A segurança na aplicação é como uma escala contínua que vai do menor grau de proteção para um alto grau de proteção. Por exemplo, numa simples tela de autenticação com usuário e senha, podemos ter desde uma implementação com vulnerabilidades a ataques conhecidos (baixa proteção) até a integração com recursos de biometria e utilização de tokens (alto grau de proteção).

Atualmente, existem diversos recursos de hardware e software que podem ser utilizados na segurança das aplicações. A identificação do grau de segurança que realmente precisamos depende de um estudo prévio dos ativos que estamos protegendo e uma análise dos riscos relacionados a estes ativos. A avaliação do resultado deste trabalho em conjunto com um levantamento de custos para implantar os recursos de segurança irá apontar para a melhor solução custo benefício de segurança que devemos implantar.

# 5. Responsabilidade da Segurança nas Aplicações

A responsabilidade da segurança de uma aplicação não é exclusivamente dos desenvolvedores.

A definição de qual o grau de segurança é o mais adequado para a aplicação deve ser estabelecido em conjunto com profissionais especializados em segurança da informação e com os gestores da aplicação.

Os profissionais de segurança da informação (equipe interna ou empresas especializadas) devem participar para identificar as ameaças existentes para a aplicação e calcular os riscos envolvidos. Este trabalho tem que ser realizado em conjunto com o gestor da aplicação que é a pessoa (ou equipe) que realmente conhece o negócio e os ativos que devem ser protegidos.

Outra responsabilidade dos profissionais de segurança é a validação da aplicação para verificar se os requisitos de segurança definidos estão corretamente implantados. Esta validação é realizada através de testes de intrusão, análises de vulnerabilidades, testes em conjunto com o desenvolvedor, avaliação do processo de desenvolvimento.

# 6. Recomendações

## *6.1. Acesso ao banco de dados*

Convém que todos os acessos aos bancos de dados Oracle e Sql Server não sejam feitos através de concatenação direta de comandos SQL. Convém que sejam utilizados objetos de bibliotecas específicas de acesso a dados (por exemplo, biblioteca ADO) para a montagem e passagem de parâmetro dos comandos SQL. Convém que cada parâmetro seja especificado através do objeto ADO Parameter, definindo tipo e tamanho quando necessário.

A utilização destas bibliotecas realiza um controle de caracteres especiais que impedem a existência de vulnerabilidades de injeções de comandos SQL.

## *6.2. Filtrar campos de entrada da aplicação*

Dados que são recebidos pela aplicação e são exibidos posteriormente num ambiente web, devem sofrer uma validação no servidor web durante o recebimento por formulários, parâmetros get, cookies ou qualquer outra fonte.

Mesmo que os dados são armazenados em banco de dados e não são exibidos imediatamente numa página web, é recomendado o filtro contra ataques de injeção de scripts e códigos HTML. Em muitos casos, informações armazenadas em banco de dados são acessadas por outros módulos de sistemas e se contiverem scripts e códigos HTML maliciosos podem ser processado indevidamente no ambiente do usuário.

O filtro deve evitar a entrada de scripts e códigos HTML, como por exemplo <script>, <iframe>, <img>, <style>. Deve-se ficar atento que em muitos casos é possível a exploração de Cross Site Scripting (XSS) aproveitando situações onde existe a concatenação de string na formação do código HTML que é enviado ao cliente. Por exemplo, no caso abaixo é possível explorar XSS sem utilizar a injeção das tags descritas anteriormente.

Buffer = "<script>alert('mensagem: " + msg + "')"

Se a variável "msg" tiver o conteúdo ');alert(document.cookie+', o cookie será exibido.

Outra solução é a utilização de produtos do tipo Web Application Firewall (WAF). O WAF ficará instalado de forma que todas as requisições da aplicação passem por ele antes. Além de outras funcionalidades, o WAF irá analisar esses potenciais ataques e evitar que eles cheguem até a aplicação.

### 6.2.1. Não confie nas validações padrões da linguagem

A linguagem normalmente possui apenas validações básicas na entrada de dados, quando possui. Não confie somente nelas. Utilize validações extras próprias na entrada de dados da sua aplicação. Somente você pode definir o que realmente é necessário validar na sua aplicação.

### 6.2.2. Valide tamanho, formato, tipo e intervalo

Um atacante pode passar conteúdo malicioso para explorar vulnerabilidades da sua aplicação. Verifique se o conteúdo é válido, observando o tamanho, formato, tipo e intervalos de valores nas entradas de dados. Você pode utilizar expressões regulares para filtras as informações.

### 6.2.3. Valide todos os pontos de entradas

Verifique todos os pontos de entradas da aplicação. A aplicação pode receber dados através parâmetros get, parâmetros post e cookie. Valide todas

estas entradas independentes se as informações são originadas de uma requisição de página Html, requisição Ajax, aplicações Win32, etc.

### 6.2.4. Não confie em validações implementadas no lado cliente

Toda informação, formatação ou scripts enviados para o cliente podem ser manipulados. As funções de segurança do navegador podem ser facilmente desabilitadas ou a aplicação pode ser acessada por um navegador desenvolvido pelo atacante. O padrão de comunicação web (http) é totalmente aberto. Não confie em validações ou funcionalidades de segurança realizadas no lado cliente.

## *6.3. Fazer a codificação HTML na resposta*

O código HTML é baseado em caracteres de controle, como por exemplo, sinais de <, >, =, etc. Quando o navegador identifica um desses controles, ele assumirá um tratamento específico como início ou fim de uma tag de marcação.

Existem situações onde queremos que o sinal "<" seja tratado com o um caracter normal e não como um controle. Neste caso existe um padrão de codificação HTML para indicar que aquele caracter não significa um controle HTML. Neste exemplo o caracter "<" deverá ser representado pela seqüência "<".

A codificação HTML consiste em trocar todos os caracteres de controle HTML dentro dos dados que serão enviados para a página.

Convém que todas as informações que tem origem em banco de dados ou outras fontes externas e que estão sendo respondidas para os usuários passem pelo processo de codificação HTML. Por exemplo, se existe o caracter "<" dentro de um texto digitado pelo usuário, este caracter deve ser transformado em "<" antes de ser enviado para a página que o usuário vai ler.

## *6.4. Evite que os navegadores memorizem a informações importantes em formulários*

Existem ferramentas que conseguem extrair todas as informações guardadas pelos navegadores. Procure evitar o armazenamento destas informações utilizando o parâmetro:

/<form … AUTOCOMPLETE="off"> - para todos os campos/

/<input … AUTOCOMPLETE="off"> - para apenas um campo/

## *6.5. Tráfego de informações sigilosas*

Dados confidenciais, como por exemplo, a senha do usuário, não deve ser exposta durante as transmissões, seja em uma comunicação servidor->cliente ou cliente->servidor. Para proteção contra a interceptação dos dados deve-se utilizar HTTPS com um certificado válido em todos procedimentos em que dados sigilosos serão trafegados.

## *6.6. Tratamento de erro*

Não mostre ao usuário erros com informações de sistema, diagnóstico ou debug. Exiba apenas uma mensagem de erro genérica caso ocorra uma exceção. Grave em log os detalhes dos erros. Se necessário, exiba um código de erro para mapear o erro ocorrido com os detalhes gravados em log para análise.

## *6.7. Exija senhas fortes*

Defina uma política de senha garantindo senhas fortes. A política de senha pode conter:

- Tamanho mínimo de senha
- Troca de senha freqüente
- Prevenção contra uso de senhas antigas
- Exigência de senha complexa

## *6.8. Solicite autenticação dos web services*

Trate a autenticação e controle de sessão dos web services como se fossem páginas e formulários web. Se um web service irá devolver informações que não públicas, implemente mecanismos de autenticação.

Embora web services são projetados para serem consumidos por outros sistemas, eles são executados dentro do protocolo HTTP, podendo ser facilmente chamados por clientes web services.