Guia para Desenvolvimento Seguro de Software do STJ

- Guia de Referência –

Janeiro 2022

Versão 1.0

Sumário

APRESENTAÇÃO E OBJETIVOS

Este documento é o guia para desenvolvimento seguro de *software* no âmbito do Superior Tribunal de Justiça (STJ). O seu objetivo é apresentar boas práticas a serem adotadas por analistas, desenvolvedores e instaladores de *software* – sejam eles prestadores de serviço terceirizados ou do quadro permanente do Tribunal, tornando o processo de concepção dos sistemas construídos dentro do STJ mais confiável, auditável, estável e protegido contra ameaças.

O presente guia funciona como complemento em relação às normas em vigor, sobretudo ao Processo de Desenvolvimento Seguro de Software do STJ, à Política de Gestão de Soluções de Software de Tecnologia da Informação do Superior Tribunal de Justiça, à Cartilha de Segurança da Informação definida pela Secretaria de Controle Interno e à norma ABNT NBR ISO-IEC 27002:2013.

FORMATO

O documento é estruturado em torno de “diretrizes”, recomendações de boas práticas a serem seguidas em cada um dos tópicos listados. Seu formato é detalhado abaixo. Após, discute-se a estrutura do documento e disposição de seu conteúdo.

Diretrizes

Uma diretriz é escrita em forma imperativa ou imperativa negativa --- “*Deve-se* [...]” ou “*Não se deve* [...]” e pode estar classificada, em função da necessidade de proteção aos dados e das obrigações imputadas ao programador ou analista, em três níveis --- cada um cumulativo com as medidas do nível anterior, salvo em sobreposição de escopo. Os níveis são:

- **Mínimo**: deveres a serem seguidos na construção de sistemas para que se obtenha um nível de segurança considerado empiricamente mínimo.
- **Padrão**: recomendações pertinentes à construção de sistemas para que se obtenha um nível de segurança considerado empiricamente padrão à data da última revisão do documento.
- **Forte**: medidas adicionais pertinentes à construção de sistemas para que se obtenha um nível de segurança empiricamente forte à data da última revisão do documento.

Na ausência da indicação de níveis de segurança em uma dada diretriz seu nível é considerado mínimo.

Estrutura

Este guia é direcionado, sobretudo, a desenvolvedores de software inseridos na construção dos sistemas do **STJ**. Considerando esse cenário, o guia foi projetado para facilitar a consulta expressa, mas sem omitir informações

detalhadas. O documento começa com um guia de referência rápida que somente traz as orientações mais relevantes -, seguido das diretrizes principais.

Um glossário está disponível ao final do documento e deve ser utilizado em caso de necessidade de detalhamento das seções iniciais.

ATUALIZAÇÕES

O resultado de cada revisão do presente instrumento busca acompanhar, sobretudo os tópicos relevantes para a necessidade do desenvolvimento de sistemas do **STJ** e não se esquiva em revisar seus requisitos conforme esta realidade se transforma.

O mecanismo que emprega para isto é a revisão constante de seus conteúdos, como é a essência do desenvolvimento ágil em uso por essa Coordenadoria de Desenvolvimento de Soluções de Software.

A seguir estão listadas linhas gerais que devem ser observadas quando da atualização do documento:

- Deve-se descrever as diretrizes de maneira breve. Detalhes podem ser incluídos nos anexos técnicos ao final do documento.
- Deve-se utilizar a estrutura de tópicos do formato do documento (seus estilos; Título, Legenda, Corpo, entre outros) sempre que possível.
- Deve-se reportar à chefia qualquer discordância das práticas aqui descritas com portarias e/ou leis cujo conteúdo e normatização se sobreponha às diretrizes aqui apresentadas ou no caso destas adentrarem a competência de outros setores do STJ.
- Deve-se adicionar conteúdo partindo das medidas que provém o nível de segurança mínimo referente àquele aspecto; níveis mais elevados de segurança (padrão, forte) devem ser descritos depois ou elencados no guia, se necessário.
- Deve-se escrever diretrizes objetivas, um ponto de cada vez, mas com a aplicabilidade mais geral possível.
- Deve-se incluir a motivação e explicações preliminares sobre uma diretriz, se necessário, no preâmbulo de sua seção.

GUIA DE REFERÊNCIA RÁPIDA

Tópico	Descrição	Diretriz **Mínima**	Diretriz **Padrão**	Diretriz **Forte**
Armazenamento de Dados	Armazenamento para Dados Abertos	Acesso de escrita restringido por senha.	-	-
Armazenamento de Dados	Armazenamento para Dados Fechados	Acesso de leitura/escrita restrito por senha.	-	Deve-se armazenar dados criptografados.
Armazenamento de Dados	Permissões de Acesso a Dados em Banco	Aplicação não deve utilizar usuário root.	Aplicação não deve ter permissões DDL, somente permissões estritamente necessárias.	Correspondência 1-1 entre usuário de sistema e de banco.
Armazenamento de Dados	Gerenciamento e Distribuição de Senhas para Acesso a Dados	As senhas não devem ser armazenadas em código fonte	Senhas devem seguir padrão deste documento. Não utilizar mesma senha para homologação e produção. Salvar de forma segura dados de usuários e sistemas que utilizam a senha	-
Controle de Usuários: Acessos e Permissões	Identidade do Usuário e Nível de Acesso	Usuário e senha nominais.	Dar ciência das permissões e níveis de acesso. Utilizar grupos do AD.	Utilizar certificado digital.
Controle de Usuários: Acessos e Permissões	Autenticação de Usuários	Não armazenar senhas em texto plano sem utilizar um algoritmo de *hash* seguro e *salt*.	Deve-se utilizar autenticação via AD e/ou o framework OAuth2 sempre que possível para autenticar usuários internos.	Utilizar tecnologias que permitam aumentar a segurança na autenticação do usuário como "autenticação em 2 fatores", biometria, e-mails de confirmação, etc.
Controle de Usuários: Acessos e Permissões	Autenticação em Sistemas Web	HTTPS ao menos nas telas de *login*	HTTPS em todo o sistema e verificações adicionais.	Utilizar chaves de acesso ou certificados digitais que permitam a identificação dos acessos na integração dos sistemas.
Comunicação Segura	Comunicação entre sistemas e/ou módulos	Controle de duplicação e integridade da informação	Controle de autenticação e confidencialidade	Controle para não-repúdio e registro de entrega
Ataques aos Sistemas e suas Defesas	Prevenção de ataques	Prevenir os 10 riscos de ataques mais conhecidos	Submeter o sistema às ferramentas de	-

		da OWASP (OWASP Top Ten)	testes de invasão.	
Auditoria, Rastreamento e Logs	Rastreamento das operações realizadas pelos usuários nos sistemas	Documento de requisitos do *software* deverá definir as informações a serem armazenadas e o local de armazenamento.	Documento de requisitos do *software* deverá definir políticas de retenção e revisão dos logs.	-
Prevenção, Reação e Mitigação de Falhas de Segurança	Diretivas de backup	Incluir no plano de produto as necessidades e responsabilidades de *backup* de dados e código-fonte	Definir procedimento e capacitar responsáveis pela restauração de backups	Criar *baselines* de versões e realizar simulações de restauração de dados continuamente
Prevenção, Reação e Mitigação de Falhas de Segurança	Testes de Softwares	Realizar testes manuais antes de liberações de versão de *software*	Elaborar testes automatizados, cenários de testes e outras políticas que garantam segurança, sigilo e não vulnerabilidade do *software*	Propor constantes desafios com intuito de identificar falhas de segurança em *software.*
Prevenção, Reação e Mitigação de Falhas de Segurança	Ocorrências de falhas de segurança	Tornar o sistema indisponível para correção da falha	Acompanhamento pós-ocorrência. Análise de causa raiz Perguntas que devem ser respondidas: quais APIs causaram o problema? qual o fluxo de dados pode impactar? quais ferramentas ou carga útil foram usadas para o problema? Por fim, apresentar um plano para corrigi-lo.	Identificar se a ocorrência é um CVE [1](*Common Vulnerabilites and Exposures*) ou vulnerabilidade conhecida,e classifica-la com o CVSS (*Common Vulnerability Scoring System*[2]). Dessa forma é possível identificar o nível de gravidade e o impacto bem como planejar ações de prevenção e mitigação.
Criptografia e Hash	Tamanho de chave para Cifradores simétricos/assimétricos.	128/1024 bits	192/2048 bits	256/4096 bits
Criptografia e Hash	Modo de cifrador de bloco.	Mais seguro que ECB	-	-

[1] Mais informações em CVE *Security Vulnerability Database. Security vulnerabilities, exploits, references and more* (cvedetails.com) e CVE - CVE (mitre.org)

[2] Calculadora para definição do CVSS (*Common Vulnerability Scoring System*) em https://www.first.org/cvss/calculator/3.1

Criptografia e Hash	Requisitos cifradores.	Somente chave sigilosa, melhor ataque força-bruta.	Não usar cifradores/modos obsoletos. *E.g.*, DES, RC4, *etc.*	-
Criptografia e Hash	Requisitos função de *hash* criptográfico.	Usar *salt* sempre que possível.	Não usar cifradores/modos obsoletos. *Eg* ., MD5, SHA1, *etc.*	
Senhas	Armazenamento	Senhas devem ser armazenadas criptografadas e com *hash*.	Criptografia usada para o armazenamento com a descrita no nível *padrão* .	Criptografia usada para o armazenamento com a descrita no nível *forte* .
Ciclo de Vida de *Software*	Desenvolvimento de produto	Deve haver etapa de Análise de riscos de segurança	-	-
Ciclo de Vida de *Software*	Documentação e codificação.	Documentar medidas de segurança, inclusive no código da aplicação.	Práticas de segurança integradas no pipeline de integração contínua de forma a preservar os aspectos de segurança do *software*	-
Comunicação inter-sistemas	Comunicação segura entre sistemas e módulos	HTTPS	HTTPS com certificado digital reconhecido internacionalmente (WebTrust ou ETSI audits), banco de dados,VPN	WS-ReliableMessaging
Ambiente de desenvolvimento	Armazenamento do código fonte	Sistema de controle de versão GITLAB ou SVN (legado)	-	-
Ambiente de desenvolvimento	Acesso ao código fonte	Servidores da CDES e colaboradores de contratos de desenvolvimento.	Definir com chefia caso-a-caso.	
Ambiente de desenvolvimento	Segregação dos Ambientes (DEV,PROD,HOM,TESTE)	Banco de dados e servidor de aplicação individualizados	-	Acesso restrito ao ambiente de produção
Ambiente de desenvolvimento	E-mails dos sistemas	E-mail criado especificamente para o sistema	-	-

DIRETRIZES

Armazenamento de Dados

Esta seção apresenta definições e diretrizes que tratam do armazenamento de informações, sigilosas ou não, e de sua disponibilização. Define taxonomia para classificação de dados e descreve procedimentos para o

armazenamento seguro dessa informação em bancos de dados. Detalha o gerenciamento de permissões de acesso e distribuição de senhas a serem adotadas para operacionalização essas estruturas.

Sigilo
No escopo deste documento os dados serão classificados, quanto ao seu *sigilo*, como:
Abertos. Dados públicos (informação pública), cujo conteúdo pode ou deve ser divulgado ao público externo.
Fechados. Dados cujo acesso é restrito a um grupo específico de pessoas (informação secreta, sigilosa ou interna).

Procedimentos e Meios para Armazenamento de Dados
Diretrizes para armazenamento de dados ***abertos***:

Mínimo
- Não se deve utilizar meio de armazenamento que não possua acesso para escrita restrito por senha.

Diretrizes para armazenamento de dados ***fechados***:
Mínimo
- Não se deve utilizar meio de armazenamento que não possua acesso para leitura e escrita restrito por senha.

Forte
- Deve-se armazenar dados criptografados.

Permissões para Acesso a Informações em Bancos de Dados

Diretrizes que tratam de permissões de acessos às informações contidas em bancos de dados por usuários e aplicações.

Observação. Deve ser dada especial atenção às permissões de acesso das tabelas de auditoria do sistema.
Observação. A alocação de permissões para usuários da aplicação deve ser feita em função dos recursos disponíveis (*eg*, quando existe somente um usuário de banco de dados disponível para acessar o sistema), *mensurados ainda na fase de projeto de software.*

Mínimo
- Não se deve disponibilizar a aplicações acesso a algum banco de dados utilizando *login* de usuário com permissões de *root*.

Padrão
- Não se deve disponibilizar às aplicações o acesso a algum banco de dados utilizando *login* de usuário com permissões para execução de comandos em Data Definition Language (DDL).
- Não se deve disponibilizar a aplicações acesso a algum banco de dados utilizando *login* de usuário com permissões além das estritamente necessárias ao seu funcionamento.

Forte

- Deve haver correspondência um-para-um entre cada usuário de uma dada aplicação e do banco de dados.

Gerenciamento e Distribuição de Senhas para Acesso a Dados

Esta seção apresenta Diretrizes para o gerenciamento e uso das senhas que protegem os dados armazenados:

Mínimo
- Não se deve utilizar o armazenamento de senhas em código-fonte.

Padrão
- Deve-se armazenar de forma segura os dados de usuários e os sistemas que utilizam cada senha fornecida.
- Não se deve utilizar as mesmas senhas para ambientes de desenvolvimento ou homologação e produção.

Gerenciamento de Acessos e Permissões de Usuários

Esta seção apresenta definições e diretrizes que tratam do controle de acesso aos dados impostos aos usuários e a atribuição das permissões necessárias.

Autorização e Autenticação de Usuários

Diretrizes para a verificação da identidade de usuários ao realizarem operações nos sistemas e para determinação de identidade do usuário e seu correspondente nível de acesso às informações.

Mínimo
- Não se deve armazenar senhas em texto plano sem utilizar um algoritmo de *hash* seguro e *salt*.
- Deve-se utilizar controle de usuário e senha nominais para determinar a identidade do usuário.

Padrão
- Deve-se utilizar autenticação via AD e/ou o framework *OAuth2* sempre que possível para autenticar usuários internos.
- Deve-se dar ciência ao usuário das permissões e níveis de acesso que possui.
- Deve-se utilizar grupos de *Active Directory* (AD) para determinar as políticas de acesso e roles de usuário.

Forte
- Deve-se utilizar certificado digital para determinar a identidade do usuário.

Observação. Autenticação AD *versus* OAuth2.

No *OAuth2*, ao contrário do AD, a autenticação é feita diretamente em uma página do externa, via HTTPS, sendo que o sistema web *não tem acesso* às credenciais inseridas pelo usuário. Depois do *login*, a página externa retorna um

token para a aplicação. Esse *token* garante que o usuário foi autenticado corretamente.

No OAuth2 é exigida uma conexão ativa com a Internet. Em situações de contingência onde o usuário não tenha acesso à página externa, não será possível autenticar-se no sistema.

Autenticação em Sistemas Web

Diretrizes específicas para autenticação em sistemas *web*, que realizam esse controle fazendo a informação trafegar em meios de acesso compartilhados, utilizando protocolos do nível de Aplicação OSI (*eg*, HTTP).

Observação. Sendo o HTTP um protocolo *stateless*[3] , que utiliza *cookies*[4] para manter sessões de usuário, faz-se necessário garantir tanto a segurança da troca de credenciais (*login*) quanto a segurança das demais páginas acessadas pelos usuários dos sistemas web. O protocolo HTTPS visa contribuir para que essa segurança seja garantida. Os sistemas web podem utilizar certificados que o STJ já possua ou certificados fornecidos por autoridades certificadoras reconhecidas internacionalmente (alguns, inclusive, gratuitos[5]).

Mínimo

- Deve-se utilizar HTTPS para controle de autenticação ao menos nas telas de login do sistema.

Padrão

- Deve-se utilizar HTTPS em todas as telas do sistema.
- Não se deve limitar o controle de autenticação em sistemas web ao uso de HTTPS, empregando-se outras tecnologias o acesso dos usuários pode (conforme o sistema) ser restringido, ainda, por outros critérios. Por exemplo, garantir que, uma vez autenticado, o usuário não possa acessar o sistema de outro endereço IP, a menos que se autentique novamente.

Comunicação Segura

Esta seção apresenta definições e diretrizes que tratam da transmissão segura de dados sensíveis entre sistemas, de modo a salvaguardar a integridade, autenticidade e demais atributos pertinentes ao uso dos dados comunicados.

Diretrizes para assegurar a segurança da comunicação de dados em função das características apresentadas, conforme os níveis elencados abaixo, de maneira cumulativa.

---

[3] Mais informações em https://pt.wikipedia.org/wiki/Protocolo_sem_estado

[4] Mais informações sobre *cookies* em https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

[5] Há sites como "Let's Encrypt" (https://letsencrypt.org/) que fornecem certificados gratuitos

Mínimo

- Deve-se empregar canal de comunicação com controle de duplicação e perda de informações/mensagens.
- Deve-se empregar canal de comunicação que provenha controle de integridade dos dados transmitidos (*eg*, HTTPS).

Padrão

- Deve-se empregar canal de comunicação com controle de autenticação (*eg*, HTTPS, certificados digitais gerados por autoridades confiáveis, VPNs).
- Deve-se armazenar de maneira segura os dados a serem transmitidos em ambas as extremidades da comunicação (vide a respectiva seção deste documento).
- Deve-se empregar canal de comunicação que provenha confidencialidade dos dados transmitidos (*eg*, HTTPS, VPNs).

Forte

- Deve-se empregar canal de comunicação que provenha garantia de não-repúdio dos dados transmitidos (*eg*, certificados digitais emitidos por entidade confiáveis).
- Deve-se utilizar logs confiáveis das informações transmitidas, com confirmação de entrega e recepção das mensagens (*eg*, *WS-ReliableMessaging* para SOAP WS).

Ataques a Sistemas e suas Defesas

Esta seção apresenta diretrizes para reforçar a resiliência de sistema a ataques contra sistemas e aplicações.

Observação. Recomenda-se que sejam prevenidos os principais ataques conhecidos em relação à segurança do sistema, de forma a evitar que ataques mal-intencionados possam comprometer a segurança do sistema, expor dados sigilosos e realizar operações não autorizadas, dentre outras vulnerabilidades.

Mínimo

- Deve-se prevenir os 10 ataques mais conhecidos (OWASP Top 10 [6]).

Padrão

- Deve-se submeter os sistemas a ferramentas de testes de invasão automatizados.

Auditoria, Rastreamento e Logs

Esta seção apresenta diretrizes para a manutenção de registros/logs para posterior auditoria, rastreamento e consulta de incidentes ligados à segurança dos sistemas. Cada sistema possui uma criticidade diferente no que se refere a restrição de acesso a dados, não-repúdio e histórico de operações realizadas no banco de dados. Por esse motivo, essa seção não define quais informações devem ser auditadas, mas sim sugere possíveis itens que podem ser auditados,

[6] Mais informações em: https://owasp.org/www-project-top-ten/

rastreados ou logados. Estes itens, então, devem ser avaliados pelos gestores do produto.

Observação. Exemplos de eventos que podem ser registrados:

- operações de *login* e *logout*;
- acessos a determinadas telas ou seções do sistema;
- acesso a informações com alguma restrição (*eg*, documentos sigilosos, processos em segredo de justiça, dados pessoais ou bancários);
- operações de inclusão, alteração ou exclusão de registros no banco de dados;
- alteração de perfil de acesso (para sistemas que possuem acesso com diferentes perfis); e
- execução de *jobs* e tarefas automatizadas.

Observação. Exemplos de informações que podem ser armazenadas, relativas a cada evento:

- data e hora;
- usuário que efetuou a operação;
- endereço IP;
- identificador da sessão do usuário (quando aplicável, *eg*, *cookie*);
- tela (página) do sistema de onde a operação foi realizada;
- identificador da instância (para sistemas clusterizados);
- para operações de inserção, alteração ou exclusão, o tipo da operação, nome da tabela que foi manipulada, ID do registro e, se for o caso, valores anterior e atual de cada campo;
- parâmetros informados pelo usuário (*eg*, parâmetros GET ou POST), tomando cuidado de não armazenar dados sensíveis, como senhas;
- tempo de resposta do sistema;
- para execução de *jobs* e tarefas automatizadas, armazenar o resultado da operação: falha, sucesso, cancelada, etc.

Observação. Exemplos de forma de captura dos dados para auditorias:

- alterações aplicadas no banco de dados podem ser auditadas via triggers[7];
- auditar as alterações a partir da própria aplicação[8] --- algumas informações poderão não ser registradas (eg, operações SQL realizadas por fora da aplicação).
- em sistemas web desenvolvidos em Java, um Filtro[9] pode interceptar as requisições feitas à aplicação.

Mínimo

[7] Para aplicações que utilizam PostgreSQL, há uma proposta de rotina de auditoria de DML em https://wiki.postgresql.org/wiki/Audit_trigger_91plus .

[8] Para aplicações que utilizam Hibernate, é possível utilizar "Envers" https://docs.jboss.org/envers/docs/ ou outro *event listener*.

[9] Mais informações sobre filtros em http://www.oracle.com/technetwork/java/filters-137243.html

- Deve-se definir no documento de especificação de requisitos do sistema quais informações deverão ser registradas e o local de armazenamento dos dados da auditoria.

Padrão

- Deve-se definir no documento de especificação de requisitos do sistema quais são as políticas de retenção (tempo mínimo de armazenamento dos dados de auditoria) e de revisão dos logs --- *eg*: procedimentos para revisar os logs, analisando se não há indícios de operações indevidas no sistema.

Prevenção, Reação e Mitigação de Falhas de Segurança

Esta seção apresenta diretrizes para a realização de procedimentos que garantam uma reação adequada à ocorrência de falhas de segurança. Detalha-se o emprego de backups, testes e tratamento de ocorrências.

Backups

Diretivas para a elaboração de processos e procedimentos envolvendo a construção, uso e manutenção de backups.

Observação. A adequação às diretrizes de backup depende, muitas vezes, de políticas e atuação da área de infraestrutura, mas são importantes aspectos a serem considerados e monitorados no desenvolvimento de aplicações seguras.

Mínimo

- Deve-se incluir no plano de projeto a especificação da necessidade e a atribuição da responsabilidade de realização de backups do banco de dados e dos códigos-fonte do sistema, bem como as políticas de acesso a este backup.

Padrão

- Deve-se definir um procedimento estruturado para a restauração de backups.
- Deve-se definir e capacitar responsáveis pela recuperação dos backups.

Forte

- Deve-se criar baselines das versões do sistema, facilitando a recuperação ágil para uma versão anterior.
- Deve-se realizar simulações de restauração de dados continuamente.

Testes

Diretivas para a elaboração de processos e procedimentos para a elaboração, execução e verificação de procedimentos para testes de segurança. No que diz respeito à segurança, é desejável a preocupação com a definição de testes capazes de encontrar vulnerabilidades no sistema, permitindo, assim, que sejam realizadas as devidas correções para evitar que as vulnerabilidades cheguem ao ambiente de produção.

Mínimo

- Deve-se realizar testes manuais de segurança antes de cada versão do software que modifique sua estrutura (telas de *login*, serviços não autenticados, novos formulários com interação com o usuário, etc).

Padrão

- Deve-se garantir, através de testes automatizados, que os serviços e dados sigilosos estão protegidos e disponíveis apenas para os usuários detentores das informações.
- Deve-se elaborar uma política de testes, automatizados ou não, visando a garantia de não vulnerabilidade aos principais ataques conhecidos em sistemas.
- Deve-se definir cenários de testes voltados à garantia dos requisitos não funcionais do software, preferencialmente realizado por uma equipe de testes diferente da equipe de desenvolvimento do software, com intuito de se evitar vícios.
- Deve-se definir cenários de testes, principalmente nos aspectos de segurança, para os casos de atualizações na arquitetura do sistema (servidores de aplicação, banco de dados, versões de browser, versões de sistema operacional, etc).

Forte

- Deve-se propor constantes desafios entre as equipes para testar segurança dos sistemas em formato de competição.
- Deve-se usar ferramentas especializadas em teste de segurança de aplicações, tanto de forma estática (*SAST - Static Application Security Testing*) como de forma dinâmica (*DAST - Dynamic Application Security Testing*) para explorar possíveis vulnerabilidades das aplicações desenvolvidas na CDES.

Ocorrências

Diretivas para construção de procedimentos visando o saneamento e mitigação da ocorrência de falhas de segurança. Quando da ocorrência de uma falha de segurança, é necessária ação imediata para identificar e reconhecer o incidente, mitigar tal ocorrência e prevenir novas falhas.

Mínimo

- Deve-se manter procedimento planejado para que o sistema se torne imediatamente indisponível para realização de manutenção corretiva.

Padrão

- Acompanhamento pós-correção de ocorrências de falha de segurança.

Ambiente de Desenvolvimento

Esta seção apresenta diretrizes para a instalação, configuração e gerenciamento de ambientes de desenvolvimento de sistemas.

Acesso ao Código-Fonte

Diretivas para controle de acesso dos desenvolvedores ao código-fonte das aplicações.

Observação. Quanto ao sigilo do código-fonte dos sistemas desenvolvidos, devem ser, por padrão, de livre acesso aos servidores da STI. As demais situações deverão ser analisadas, projeto a projeto, pela chefia.

Mínimo

- Deve-se utilizar um sistema de controle de versão com controle de acesso e recuperação em caso de falhas, *eg*: Gitlab.

Padrão

- Deve-se utilizar um controle de versão distribuído, que mantém um repositório completo em cada máquina de desenvolvimento, *eg*: Git.

Separação de Ambientes

Diretivas para a separação de ambientes de desenvolvimento/testes/homologação (DEV / TESTE / HOM) do ambiente de produção (PROD).

Observação. As aplicações desenvolvidas devem considerar o uso de repositórios seguros de dados, especialmente na forma de banco de dados com acesso controlado ou arquivos criptografados, quando o uso de banco de dados não for possível/desejável.

Mínimo

- Deve-se utilizar bancos de dados distintos para cada ambiente.
- Deve-se utilizar servidores de aplicação/web distintos para cada ambiente.
- Deve-se prover acesso ao ambiente de desenvolvimento/ testes/ homologação apenas aos integrantes da equipe de desenvolvimento e aos interessados no produto.

Padrão

- Deve-se prover um instalador expresso para a instalação do ambiente necessário para a execução de uma dada aplicação.
- Deve-se realizar testes periódicos para assegurar a segurança do ambiente de desenvolvimento/testes/homologação.

Forte

- Não se deve fornecer senhas de acesso ao ambiente de produção aos desenvolvedores.

Parametrização para Proteção de Dados

Esta seção apresenta diretrizes para a configuração de proteção a dados sensíveis. São detalhados parâmetros para criptografia, *hash* e gerenciamento de senhas.

Criptografia e *Hash*

Diretrizes para a configuração e utilização de algoritmos de criptografia e *hash* visando prover confidencialidade a dados.

Observação. Dados sigilosos e sensíveis devem ser criptografados sempre que possível. O método de criptografia empregado deve obedecer às particularidades dos dados e de sua utilização, seguindo os parâmetros aqui listados.

Observação. Deve-se utilizar *hashes* criptográficos sempre que possível, sobretudo nos seguintes casos: verificação da integridade de dados; armazenamento e verificação de senhas; provimento de identificador "único" para objetos em um sistema e geração de números pseudo-aleatórios.

Mínimo

- Deve-se utilizar um método criptográfico que siga o princípio de *Kerckhoffs*[10]; o método de encriptação e seus parâmetros devem ser públicos e estar documentados, somente a chave criptográfica deve ser mantida em sigilo.
- Não se deve utilizar um cifrador que admita um método conhecido para quebra da chave criptográfica melhor do que a força bruta, baseada em tentativa e erro.
- Não se deve utilizar o modo de cifrador de bloco *electronic codebook* (ECB) ou modos menos seguros.
- Não se deve utilizar um tamanho da chave menor que 128 bits (cifrador simétrico) ou 1024 bits (cifrador assimétrico).
- Não se deve utilizar função de *hash* sem algum tipo de *salt*.

Padrão

- Não se deve utilizar algoritmos considerados obsoletos para criptografia e *hash* criptográfico. Exemplos: MD5, SHA1, DES/3DES, RC2, RC4, MD4.
- Não se deve utilizar um tamanho da chave menor que 192 bits (cifrador simétrico) ou 2048 bits (cifrador assimétrico).
- Não se deve distribuir chaves criptográficas sem a utilização de uma infraestrutura de chave pública e, portanto, sem a utilização de um cifrador assimétrico.

Forte

- Não se deve utilizar um tamanho da chave menor que 256 bits (cifrador simétrico) ou 4096 bits (cifrador assimétrico).

Ciclo de Vida de Software

Esta seção apresenta diretrizes para reforço da segurança de software nas diferentes fases de seu ciclo de vida; projeto, codificação e manutenção. Traz, ainda, diretrizes para a aplicação com as pessoas envolvidas nestas diferentes fases.

Produto

[10] Shannon, Claude (4 October 1949). "Communication Theory of Secrecy Systems". Bell System Technical Journal. 28: 662. Retrieved 20 June 2014.

Diretrizes para tornar seguras práticas utilizadas durante a etapa de elaboração de produto, inserida dentro da metodologia de desenvolvimento do STJ.
Observação. As práticas aqui elencadas são consonantes com o que consta na Resolução STJ/GP n. 11 de 12 de novembro de 2015, que institui a política de segurança da informação neste Tribunal.

- Deve-se empregar modelo de produto de software que contemple:
  - etapa de modelagem de ameaças;
  - definição clara dos riscos de segurança; e
  - nível de severidade que o comprometimento de dados sensíveis traria ao sistema e à instituição.
- Não se deve omitir, durante o desenvolvimento do produto e sua execução, a definição de responsabilidades pela segurança de dados do sistema e como essa responsabilidade será verificada.
- Deve-se utilizar cronograma de desenvolvimento que contemple pontos de verificação de segurança do sistema desenvolvido ao longo de sua construção.

Codificação

Diretrizes para tornar seguras práticas utilizadas durante a etapa de codificação de sistemas:

- Deve-se documentar, inclusive no código da aplicação, as medidas protetivas aplicadas no código-fonte, de modo a indicar precisamente o procedimento utilizado e suas peculiaridades.
- Não se deve armazenar senhas em código-fonte.
- Não se deve utilizar códigos da Internet sem conhecer a fonte ou entender seu funcionamento.
- Deve-se evitar ao máximo usar funções ou recursos obsoletos (*deprecated*), restringindo de forma ampla o seu acesso. Novas soluções devem ser estudadas para substituir, o mais rapidamente possível, tais recursos.
- Deve-se aplicar, sempre que possível, o conceito de validação positiva, que se trata de um mecanismo que usa critérios pré-definidos para validar o tamanho, caracteres, formato, e as regras de negócio que se aplicam sobre os dados antes de aceitar a entrada. Qualquer dado que não atenda aos critérios deve ser rejeitado.
- Todo ponto de interação de dados com o usuário do sistema (*input*) deve ter sua validação feita na entrada do dado e na sua apresentação (*output*).
- Validações de segurança devem ser realizadas no servidor (*Webserver*).

Manutenção

Diretrizes para tornar seguras práticas utilizadas durante a etapa de manutenção de sistemas:

- Não se deve habilitar as atualizações automáticas de software ou componentes utilizados na construção de um sistema, sob pena de introdução indevida de falhas de segurança.

- Não se deve modificar software de terceiros, salvo quando estritamente necessário; controles de segurança internos podem ser invalidados. A mudança deve ser feita pelo desenvolvedor original do sistema sempre que possível.

Pessoal

Diretrizes para a perpetuação de práticas de desenvolvimento seguro junto às pessoas que operam as diferentes fases do ciclo de vida do software:

- Deve-se proporcionar treinamento e capacitação de programadores para aquisição e revisão de princípios de segurança computacional e desenvolvimento de software seguro.

MELHORIAS NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Para se construir um sistema seguro, devem ser realizadas melhorias de processo, priorizando os aspectos de segurança em cada fase do ciclo de vida de desenvolvimento do sistema, independentemente do modelo de ciclo de vida utilizado.

Etapas do processo de desenvolvimento seguro

O SDL (*Security Development Lifecycle* – Ciclo de Vida do Desenvolvimento da Segurança) é um processo de desenvolvimento de software criado pela Microsoft em 2004, que ajuda os desenvolvedores a construir softwares mais seguros e tratar os requisitos de conformidade, reduzindo simultaneamente o custo de desenvolvimento.

Uma visão geral das etapas envolvidas no SDL é mostrada na figura abaixo. Cada uma das etapas será descrita mais adiante, em termos dos aspectos de segurança que nelas são contemplados.

Figura 1 - Etapas do Processo de Desenvolvimento Seguro da Microsoft

Treinamento

Permite compreender os conceitos básicos de segurança e os mais recentes desenvolvimentos em segurança e privacidade, o que auxilia na redução do número e da gravidade das vulnerabilidades exploráveis do software, bem como a reagir adequadamente aos cenários de ameaça em constante mudança.

O treinamento deve envolver os conceitos fundamentais para a construção de um software melhor, incluindo: design de segurança, modelagem de ameaças, desenvolvimento seguro, testes de segurança e melhores práticas relacionadas à privacidade.

Requerimentos

A fase de concepção do projeto é o melhor momento para considerar questões de privacidade e segurança fundamentais e analisar como alinhar a qualidade e os requisitos regulatórios com os custos e as necessidades de negócios.

As práticas desta fase são: “Estabelecer Requisitos de Segurança e Privacidade”, “Especificar *Quality Gates* e *Bug Bars*” e “Realizar Avaliações de Risco de Segurança e Privacidade”.

Prática - Estabelecer Requisitos de Segurança e Privacidade

Recomenda-se definir e integrar os requisitos de segurança e privacidade no início da especificação do software. Isso ajuda a tornar mais fácil identificar

os marcos e os entregáveis principais e a minimizar os desvios em planos e cronogramas. A análise de segurança e privacidade inclui a alocação de especialistas em segurança, a definição de critérios de privacidade e de segurança mínimos para um aplicativo e a implantação de um sistema de acompanhamento de vulnerabilidades de segurança e de itens de trabalho.

Prática - Especificar Quality Gates e Bug Bars

Recomenda-se definir os níveis mínimos aceitáveis de qualidade de segurança e de privacidade no início do projeto. Isso ajuda uma equipe a compreender os riscos associados a problemas de segurança, identificar e a corrigir erros de segurança durante o desenvolvimento e a aplicar os padrões durante todo o projeto.

Estabelecer uma *bug bar* representativa envolve definir claramente os limiares de severidade de vulnerabilidades de segurança (por exemplo, não deve haver vulnerabilidades conhecidas no aplicativo com uma classificação de "crítica" ou "importante" no momento da liberação), nunca permitindo que ceda, uma vez que tenha sido definida.

Prática - Realizar Avaliações de Risco de Segurança e Privacidade

Recomenda-se examinar o design de software baseando-se em requisitos regulatórios e de custos. Isso auxilia uma equipe a identificar quais partes de um projeto exigirão a modelagem de ameaças e revisões de design de segurança antes do lançamento e a determinar a classificação de impacto de privacidade de um recurso, produto ou serviço.

Projeto

A fase de projeto é fundamental para estabelecer as melhores práticas em torno do design e especificações funcionais e realizar análises de risco que ajudarão a atenuar as questões de segurança e privacidade ao longo de um empreendimento.

As práticas relativas a esta fase são: "Estabelecer Requisitos de Design", "Realizar Análise/Redução de Superfície de Ataque" e "Fazer Modelagem de Ameaças".

Prática - Estabelecer Requisitos de Design

Aborda questões relativas à segurança e à privacidade no início do projeto, o que ajuda a minimizar o risco de desvios de cronograma e a reduzir o custo de projetos.

As especificações de design devem descrever os recursos de segurança e de privacidade que serão expostos diretamente para o usuário, tais como aqueles que requerem autenticação para acessar dados específicos, antes do uso de um recurso de privacidade de alto risco.

Além disso, as especificações de design devem descrever como implementar toda a funcionalidade fornecida por um determinado recurso ou função de forma segura.

Validar todas as especificações de design com relação à especificação funcional envolve especificações de design precisas e completas, incluindo requisitos de design de criptografia mínimos e uma revisão da especificação.

Prática - Realizar Análise/Redução de Superfície de Ataque

Aborda a redução das oportunidades de os atacantes explorarem um ponto fraco ou vulnerabilidade potenciais, o que requer analisar cuidadosamente a superfície de ataque global e inclui desabilitar ou restringir o acesso aos serviços do sistema, aplicando o princípio de privilégio mínimo e empregando defesas em camadas, sempre que possível.

Prática - Fazer Modelagem de Ameaças

Representa aplicar uma abordagem estruturada para os cenários de ameaças durante o projeto. Isso ajuda uma equipe mais eficazmente e menos dispendiosamente a identificar vulnerabilidades de segurança, determinar os riscos destas ameaças e estabelecer as mitigações adequadas.

Implementação

O foco desta etapa é auxiliar o usuário final a tomar decisões informadas sobre as formas mais seguras para implantar o software. É também o momento de estabelecer as melhores práticas para detecção e remoção de problemas de segurança do código.

As práticas relativas a esta fase são: “Usar Ferramentas Aprovadas”, “Proibir Funções Inseguras” e “Realizar Análise Estática”.

Prática - Usar Ferramentas Aprovadas

Refere-se a publicar uma lista de ferramentas aprovadas e suas verificações de segurança associadas (tais como opções e os avisos do compilador/*linker*), o que ajuda a automatizar e reforçar facilmente as práticas de segurança a um baixo custo.

Manter a lista regularmente atualizada significa que as últimas versões da ferramenta são usadas e permite a inclusão de novas funcionalidades e proteções de análise de segurança.

Prática - Proibir Funções Inseguras

Refere-se a analisar todas as funções e API do projeto e vetar aquelas determinadas como inseguras, o que ajuda a reduzir possíveis bugs de segurança, com pouco custo de engenharia. As ações específicas incluem usar compiladores mais atuais ou ferramentas de verificação de código para analisar o código com relação às funções na lista de proibidas e, em seguida, substituí-las por alternativas mais seguras.

Prática - Realizar Análise Estática”

Trata-se da análise do código-fonte antes da compilação, fornecendo um método escalável de revisão do código de segurança e auxiliando a garantir que as políticas de codificação seguras estão sendo seguidas. Esse tipo de ferramenta pode ser usado como parte do plug-in IDE ou pode ser acionada a verificação juntamente com a compilação diária.

Verificação

Envolve um esforço abrangente para garantir que o código atende aos princípios de segurança e de privacidade estabelecidos nas etapas anteriores.

As práticas relativas a esta fase são: “Realizar Análise Dinâmica”, “Realizar Testes de *Fuzzing*” e “Conduzir Revisão de Superfície de Ataque”.

Prática - Realizar Análise Dinâmica

É importante garantir que a funcionalidade de um programa opere como planejado. Assim, esta prática desempenha a verificação em tempo de execução do software, checando a funcionalidade usando ferramentas que monitoram o comportamento do aplicativo quanto à corrupção de memória, problemas de privilégio do usuário e outros problemas de segurança críticos.

Prática - “Realizar Testes de Fuzzing

Esta prática induz deliberadamente à falha do programa introduzindo dados malformados ou aleatórios no aplicativo. Isso ajuda a revelar questões de segurança em potencial anteriormente ao lançamento, exigindo pouco investimento de recursos.

Prática - Conduzir Revisão de Superfície de Ataque

É comum para um aplicativo desviar significativamente das especificações funcionais e de design criadas durante as fases de requisitos e de design de um projeto de desenvolvimento de software. Assim, a prática “Conduzir Revisão de Superfície de Ataque” realiza a revisão da superfície de ataque após a conclusão do código, o que ajuda a garantir que quaisquer alterações de design ou implementação para um aplicativo ou sistema foram levadas em conta e que quaisquer novos vetores de ataque criados como resultado das alterações foram revistos e mitigados, incluindo modelos de ameaça.

Release

O foco desta etapa está na preparação de um projeto para liberação em produção, incluindo planejamento de maneiras para executar, de forma efetiva, tarefas de manutenção depois da liberação e abordar vulnerabilidades de segurança ou privacidade que podem ocorrer posteriormente. O objetivo final é garantir a segurança do software que será entregue ao cliente dentro das expectativas estabelecidas.

As práticas relativas a esta fase são: “Criar Plano de Resposta a Incidentes”, “Conduzir Revisão Final de Segurança” e “Certificar a Release e Arquivar”.

Prática - Criar Plano de Resposta a Incidentes

Elabora-se um plano de resposta a incidentes, que é crucial para ajudar a tratar novas ameaças que podem surgir ao longo do tempo. Ele inclui a identificação de contatos de emergência de segurança adequados e estabelece planos de manutenção de segurança para código herdado de outros grupos dentro da organização e para código de terceiros licenciados.

Prática - Conduzir Revisão Final de Segurança

Realiza-se a revisão de todas as atividades de segurança que foram feitas, o que ajuda a garantir a prontidão do lançamento do software.

A Revisão Final de Segurança (FSR) geralmente inclui a análise de modelos de ameaças, das saídas de ferramentas e da análise de desempenho com os *quality gates* e *bug bars* definidos durante a etapa de requisitos. A FSR redunda em um de três resultados diferentes: FSR aprovada, FSR aprovada com exceções ou FSR com escalação.

Prática - Certificar a Release e Arquivar

Certifica-se o software antes do lançamento de uma nova versão, o que ajuda a garantir que os requisitos de segurança e de privacidade foram atendidos.

Arquivar todos os dados pertinentes é essencial para a realização de tarefas de manutenção depois do lançamento da nova versão e ajuda a reduzir os custos a longo prazo associados com a engenharia de software sustentável. O arquivamento deve incluir todas as especificações, código-fonte, binários, símbolos privados, modelos de ameaças, documentação, planos de resposta de emergência, e termos de licença e de serviços para qualquer software de terceiros.

Resposta

A equipe de desenvolvimento deve estar apta e disponível para responder adequadamente a quaisquer relatórios de vulnerabilidades e ameaças emergentes de software.

Esta fase possui uma prática: “Executar Plano de Resposta a Incidentes”.

Prática - Certificar a Release e Arquivar

Trata-se de ser capaz de implementar o plano de resposta a incidentes instituído na fase de release, o que é essencial para ajudar a proteger os usuários contra vulnerabilidades de segurança ou de privacidade que surjam no software.

GLOSSÁRIO

ATAQUES E DEFESAS

SQL Injection. É uma forma de ataque em sistemas, realizado via interface, no qual o usuário informa trechos de SQL em campos de texto (ou até mesmo em telas de *login* ou de pesquisa), alterando a consulta prevista pelo desenvolvedor, sendo que o atacante poderá receber privilégios especiais ou poderá manipular indevidamente o banco de dados[11].

OWASP. O *Open Web Application Security Project* é uma comunidade aberta, iniciada em 2001, com o objetivo de capacitar as organizações sobre como desenvolver, adquirir, operar e manter aplicações confiáveis, em relação à segurança. O projeto oferece de forma gratuita documentos, ferramentas, fóruns e estudos sobre segurança em aplicações.

OWASP TOP 10 Um documento da OWASP bem conhecido no mercado é o Top 10. É uma lista que contém os 10 tipos de ataques mais críticos e conhecidos em aplicações. O documento descreve de forma detalhada os riscos, exemplos e também como prevenir contra estes tipos de ataques.

CRIPTOGRAFIA

Encriptação. É o processo de converter texto em claro em texto cifrado.

Decriptação. É o processo de converter texto cifrado em seu texto em claro original.

Cifrador. É um par de algoritmos que realizam a encriptação e a decriptação.

Chave. É uma sequência de bits utilizada como parâmetro secreto no cifrador, necessária para realizar encriptação e/ou decriptação. A única maneira de descobrir uma chave deve ser por força-bruta; tentar todas as alternativas no espaço de chaves possíveis. O algoritmo utilizado pelo cifrador deve garantir que chaves longas implicam em um tempo impraticável para descobrir a chave por tentativa-e-erro.

[11] Mais informações sobre *SQL Injection* podem ser encontradas em [https://www.owasp.org/index.php/SQL_Injection]

Cifrador Simétrico. É um cifrador que usa a mesma chave para encriptação e decriptação. Mais rápidos, em geral. Exemplos: Advanced Encryptation Standard (AES)[12] e Data Encryptation Standard (DES)[13].

Cifrador Assimétrico. É um cifrador que usa chaves diferentes, uma pública , uma privada ,para encriptação e decriptação. Mais lentos, em geral, mas com usos para assinatura e verificação de autenticidade. Exemplos: Rivest-Shamir-Adleman (RSA)[14] e Elliptic Curve Cryptography (ECC)[15].

Hash Criptográfico. É uma função matemática que mapeia uma entrada de tamanho arbitrário, em bits, para uma saída de tamanho fixo e que é utilizada para criptografia. A função também é de "mão única", no sentido de que é impossível inverter-la. A função deve ser determinística, de rápida computação e de alta entropia.

Salted Hash. Fragmento adicionado ao conteúdo original do *hash* para que a saída mude mesmo que o conteúdo original seja o mesmo.

Cifrador de bloco. Cifrador que que opera sobre blocos de bits de tamanho fixo com uma transformação invariável que é especificada por uma chave simétrica.

AMBIENTE DE DESENVOLVIMENTO

Open Relay. Os servidores de correio eletrônico são classificados como Open Relay quando ele processa um e-mail onde o remetente e o destinatário não são usuários do servidor em questão.

COMUNICAÇÃO SEGURA

WS-*ReliableMessaging*. Protocolo para entrega segura de mensagens SOAP.

SOAP. Protocolo para troca de mensagens em formato XML.

REST. Protocolo para comunicação entre sistemas utilizando os métodos do protocolo HTTP.

---

[12] Joan Daemen, Steve Borg e Vincent Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002. ISBN 3-540-42580-2.

[13] National Bureau of Standards, Data Encryption Standard, FIPS-Pub.46. National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977

[14] Rivest, R.; Shamir, A.; Adleman, L. (February 1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM. 21 (2): 120–126.

[15] Koblitz, N. (1987). "Elliptic curve cryptosystems". Mathematics of Computation. 48 (177): 203–209.

REFERÊNCIAS BIBLIOGRÁFICAS

Guia para Desenvolvimento Seguro de Software do Tribunal Regional do Trabalho da 4ª Região, disponível em: https://www.trt4.jus.br/portais/media/175722/guia-desenvolvimento-seguro-2018.pdf

Requisitos técnicos para aquisição e desenvolvimento de sistemas seguros de tecnologia da informação do DECEA, disponível em: https://publicacoes.decea.mil.br/api//storage/uploads/files/01b1b949-bc78-4205-95d1b2555cf1d158.pdf

Utilização de cookies HTTP, disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

CVE *security vulnerability database*. *Security vulnerabilities, exploits, references and more*, disponível em: https://www.cvedetails.com/ e https://cve.mitre.org/

Calculadora para CVSS (*Common Vulnerability Scoring System*), disponível em: https://www.first.org/cvss/calculator/3.1

Lista da OWASP com os 10 tipos de ataques mais críticos e conhecidos em aplicações, disponível em: https://owasp.org/www-project-top-ten/

Proposta de rotina de auditoria de DML para aplicações que utilizam PostgreSQL, disponível em: https://wiki.postgresql.org/wiki/Audit_trigger_91plus

Ferramenta *Hibernate Envers* para auditoria de dados em aplicações que utilizam *Hibernate*, disponível em: https://docs.jboss.org/envers/docs

Filtros para capturar dados para auditorias em sistemas web desenvolvidos em Java, disponível em: http://www.oracle.com/technetwork/java/filters-137243.html

Ataque por SQL Injection, disponível em: https://owasp.org/www-community/attacks/SQL_Injection

Shannon, Claude (4 October 1949). "Communication Theory of Secrecy Systems". Bell System Technical Journal. 28: 662. Retrieved 20 June 2014.

Joan Daemen, Steve Borg e Vincent Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002. ISBN 3-540-42580-2.

National Bureau of Standards, Data Encryption Standard, FIPS-Pub.46. National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977.

Rivest, R.; Shamir, A.; Adleman, L. (February 1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM. 21 (2): 120–126.

Koblitz, N. (1987). "Elliptic curve cryptosystems". Mathematics of Computation. 48 (177): 203–209