Universidade do Minho
Escola de Engenharia

Mestrado Engenharia Informática

PROJETO 1

Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF)

GRUPO 13

Bruno Rodrigues: pg41066

Carlos Alves pg41840

1 ÍNDICE

2 INTRODUÇÃO

Este trabalho pratico sugerido na Unidade Curricular – Engenharia de segurança tem como objetivo realizar investigação sobre um tema de desenvolvimento seguro de software, sendo este atribuído especificamente, neste caso “Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF)”.

Inicialmente é necessário recorrer a metodologias sejam elas formais ou informais no desenvolvimento de um software ou na sua manutenção. Para isso um ciclo de vida de desenvolvimento de software (SDLC) consiste essencialmente neste tipo de metodologias. Dentro deste modelo, por assim dizer, pode-se verificar um número alargado de alternativas a ser usadas, tais como em espiral, cascata e desenvolvimento de operações (DevOps), entre outras. Ainda assim, no contexto de segurança, são poucos os modelos que a abordam de forma explicita e detalhada. De tal modo, para ser incrementado tais aspetos de segurança é necessário que sejam adicionadas práticas de desenvolvimento de software que visem tal aspeto ao próprio modelo SDLC.

Em suma, as práticas em questão, sendo elas bem implementadas podem de certo modo, diminuir a probabilidade de existir vulnerabilidades ou até mesmo mitiga-las. A grande vantagem de fazer uso de tais modelos é a de diminuir o esforço preciso para alcançar um nível de segurança satisfatório.

Com isto, o aparecimento de **SSDF - Secure software Development Framework**, que consistem basicamente em mais práticas, mas de alto nível e com base em padrões já estabelecidos, permitem assim ajudar de igual modo a atingir os objetivos de segurança no desenvolvimento de software.

O uso de tais práticas oferecem diversas vantagens tais como:

- Ajuda uma empresa a documentar o seu software e definir uma melhoria de segurança no futuro.
- As práticas como já referido são padronizadas, permitindo assim que sejam facilmente aplicáveis em diferentes contextos.
- É facilmente incorporado a qualquer fluxo de trabalho no desenvolvimento de software que já esteja a decorrer, inclusive pode ser aplicado a software desenvolvido para fornecer suporte a TI, sistema de controlo industrial e ainda Internet das Coisas.

3 SSDF - SECURE SOFTWARE DEVELOPMENT FRAMEWORK

Como referido anteriormente, o SSDF nada mais são que práticas de alto nível com base em padrões já estabelecidos. Estas práticas podem ser organizadas em diferentes grupos:

- **Preparar a organização (PO)** – como o próprio nome indica, este grupo é responsável por garantir que tanto as pessoas, os processos e as tecnologias da organização estejam preparadas para o desenvolvimento de software seguro.
- **Proteger o software (PS)** – grupo responsável pela proteção do software em relação a violações e acessos não autorizados.
- **Produzir Software bem protegido (PW)** – Aqui é pretendido que seja produzido software "well-secured" que possuía um mínimo de vulnerabilidades de segurança nos lançamentos dos mesmos.
- **Responder a relatórios de vulnerabilidades (RV)** – grupo onde é identificado as vulnerabilidades nas diferentes versões do software produzido e ainda efetuar uma ação rápida e adequada na resolução nessas vulnerabilidades como em outras que possam ocorrer no futuro.

Além disto, cada prática é definida com alguns elementos: como a **prática**, onde esta deve apresentar algumas informações básicas sobre a prática em questão, uma **explicação**, razão para ser usada e identificador exclusivo. A **Tarefa** define as ações necessárias para incorporar tal prática. Um **exemplo de implementação**, onde seja possível verificar um exemplo quer seja de um processo ou até mesmo outro método que possa ser usado para implementar a prática em questão. Por fim temos a **Referência**, que é apenas um documento de práticas de desenvolvimento seguro estabelecido e ainda para uma tarefa em particular. Isto se for representado numa tabela seria uma tabela deste formato.

PRÁTICA	TAREFA	EXEMPLO DE IMPLEMENTAÇAO	REFERÊNCIAS
Preparar a Organização (PO)			
Proteger o software (PS)			
Produzir Software bem protegido (PW)			
Responder a Relatórios de Vulnerabilidades (RV)			

Figura – Estrutura

Depois de toda este conjunto de terminologias e conceitos podemos transcrever algumas das práticas descritas no documento publicado pela NIST.

4 PREPARAR A ORGANIZAÇÃO (PREPARE THE ORGANIZATION)

Como referido anteriormente, neste grupo pretende-se assegurar que as pessoas, processos e tecnologias da organização estejam preparadas a desenvolver software seguro.

Para tal estão instauradas quatro práticas para este grupo:

PO 1 - Definir requerimentos de segurança para o desenvolvimento de software	**PO 2 - Implementar cargos e responsabilidades**
PO 3 - Implementar um conjunto de ferramentas de suporte	**PO 4 - Definir critérios para a verificação da segurança de software**

4.1 DEFINIR REQUERIMENTOS DE SEGURANÇA PARA O DESENVOLVIMENTO DE SOFTWARE

4.1.1 Descrição

De uma forma genérica esta prática tem como objetivo compilar toda a informação requerida, como os requerimentos de segurança para o desenvolvimento de software, desde políticas da organização, objetivos e estratégias, e assegurar que esta informação seja de conhecimento geral.

Isto evita que uma tarefa seja desnecessariamente refeita devido à falta de transparência.

4.1.2 Tarefa

Identificar todos os requisitos de segurança aplicáveis ao desenvolvimento geral de software da organização e manter os requisitos ao longo do tempo.

4.1.3 Exemplos de Implementação

De seguida temos os exemplos de implementação, nesta prática são enumerados bastantes exemplos apenas iremos transcrever alguns de forma a não obter um relatório demasiado grande. De modo a exibir a totalidade das tabelas retiradas do documento publicado pela NIST, iremos coloca-las em anexo neste relatório.

1. Definir políticas que especificam os requisitos de segurança para o software da organização atender, incluindo práticas seguras de codificação para os desenvolvedores seguirem.
2. Definir políticas para proteger a infraestrutura de desenvolvimento, como estações de trabalho do desenvolvedor e repositórios de código.
3. Rever e atualizar os requisitos depois que tenha sido dada uma resposta a um incidente sobre uma vulnerabilidade
4. Fazer uma revisão periódica de todos os requisitos de segurança.

4.2 IMPLEMENTAR REGRAS E RESPONSABILIDADES

4.2.1 Descrição

Assegurar que toda a gente envolvida nos modelos esteja preparada para assumir as suas responsabilidades e os respetivos cargos no SDLC.

4.2.2 Tarefa

A prática Implementar Regras e Responsabilidades conta com duas tarefas distintas:

- PO.2.1 – Criar regularmente novas funções como também alterar as diversas responsabilidades já implicadas a funções existentes, de modo a possibilitar uma maior abrangência nas partes que constituem o SSDF. Além que deve ser realizado revisões ás funções e as responsabilidades e atualiza-las conforme necessário.

- PO.2.2 – Providenciar formação específica para todas as pessoas em funções responsáveis pelo desenvolvimento. Rever periodicamente a formação e atualiza-la se necessário.

4.2.3 Exemplos de Implementação

PO.2.1

Como exemplos temos que definir funções e responsabilidades relacionadas ao SSDF para todos os membros da equipa de desenvolvimento de software. Além disto, a integração de unções de segurança a essa mesma equipa também é uma forma de implementar esta prática.

A realização de uma revisão anual de todas as funções e responsabilidades e não menos importante o educar, principalmente os colaboradores que sofram de várias mudanças de papeis/funções.

PO.2.2

Na segunda tarefa (PO.2.2), pode ser descrito ou até mesmo documentado os resultados esperados na formação de determinada função. Por último, o adquirir/criar formação para cada função, no caso de ser necessário uma personalização da formação já instruída, a organização deve então adapta-la e modifica-la.

4.3 IMPLEMENTAR UM CONJUNTO DE FERRAMENTAS DE SUPORTE

4.3.1 Descrição

Fazer uso da automação para reduzir o esforço humano e melhorar a precisão, consistência e abrangência das práticas de segurança em todo o SDLC, e ainda demonstrar o uso de tais técnicas sem que seja necessário adicionar esforços e despesas significativas.

4.3.2 Tarefa

Conta com três tarefas, são elas:

- PO.3.1: Especificar quais ferramentas ou tipos de ferramentas devem ser incluídas em cada conjunto de ferramentas e quais ferramentas ou tipos de ferramentas são obrigatórios.
- PO.3.2: Implantar e configurar ferramentas, integra-las ao conjunto de ferramentas e manter as ferramentas individuais e o conjunto de ferramentas como um todo.
- PO.3.3: Configurar as ferramentas para recolher evidências do seu suporte às práticas seguras de desenvolvimento de software.

Exemplos de implementação

PO.3.1.1

- Usar tecnologias automatizadas para a gestão das ferramentas.
- Identificar diversas ferramentas de segurança, de modo a integrar no conjunto de ferramentas do desenvolvedor.

PO.3.2.1

- Avaliar, escolher e obter ferramentas.
- Analisar analiticamente os *logs* das ferramentas quanto a possíveis problemas, sejam de segurança ou outros.

PO.3.3.1

- Usar o fluxo de trabalho já existente ou sistemas de rastreamento de bugs para desenvolver uma espécie de auditoria de ações relacionadas a todas as medidas de desenvolvimento seguras que já foram realizadas.

- Determinar a frequência que as informações recolhidas devem ser revistas e implementar processos para a realização de auditorias a esses dados.

Fluxo de Trabalho

4.4 DEFINIR CRITÉRIOS PARA VERIFICAÇÕES DE SEGURANÇA DE SOFTWARE

4.4.1 Descrição

Esta prática deverá ajudar a garantir que o software resultante do SDLC atenda às expectativas da organização, definindo critérios para verificar a segurança do software durante o desenvolvimento.

4.4.2 Tarefa

Conta com duas tarefas, são elas:

- PO.4.1 – Definir critérios para verificações de segurança de software num ou mais pontos no SDLC.
- PO.4.2 – Implementar processos/mecanismos de forma a reunir as informações necessárias em apoio aos critérios.

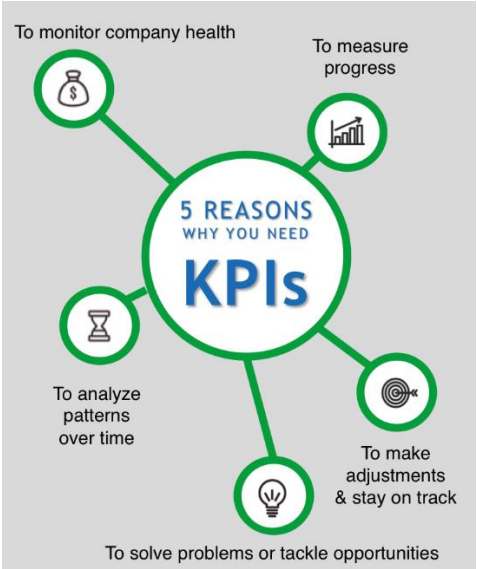
4.4.3 Exemplos de Implementação

PO.4.1.1

- Verificar se os critérios indicam adequadamente a eficácia com que o risco de segurança está sendo gerido.
- Definir indicadores-chave de desempenho (KPIs) para segurança de software.

PO.4.1.2

- Incorporar ferramentas adicionais, para dar suporte à geração e recolha de informações que suportam os critérios.
- Automatizar os processos de tomada de decisão utilizando os critérios.



Uso de KPI's e os seus beneficios

5 PROTEGER O SOFTWARE (PROTECT SOFTWARE)

Como já referimos, este grupo visa a proteção do software em relação a violações e acessos não autorizados.

No documento em estudo, verifica-se a existência de três práticas fundamentais dentro deste grupo:

- PS.1 – Proteger todas as formas de código contra acesso não autorizado e ainda adulteração;
- PS.2 – Fornecer um mecanismo para verificar a integridade da versão do software;
- PS.3 – Arquivar e proteger cada versão de software.
-

5.1 PROTEGER TODAS AS FORMAS DE CÓDIGO CONTRA ACESSO NÃO AUTORIZADO E AINDA ADULTERAÇÃO

5.1.1 Descrição

Visa ajudar a evitar alterações não autorizadas no código, inadvertidas e intencionais, que podem contornar ou negar as características de segurança pretendidas do software. Tal prática permite impossibilitar de certa forma o roubo do software e ainda dificulta no descobrimento de vulnerabilidades desconhecidas, mas encontradas por atacantes.

5.1.2 Tarefas

Armazenar todo o código elaborado no desenvolvimento do software, com base no princípio do mínimo privilégio, isto é, somente o pessoal autorizado tenha acesso a determinado código.

5.1.3 Exemplo de implementação

- Armazenar o código-fonte num repositório de códigos e restringir o acesso ao mesmo.
- Usar assinatura de código e hash criptográficas para ajudar a proteger a integridade e a proveniência dos executáveis e arquivos, respetivamente.

5.2 Fornecer um mecanismo para verificar a integridade da versão do software

5.2.1 Descrição

Responsável por ajudar os consumidores de software a garantir que o software adquirido é legítimo e que não foi violado.

5.2.2 Tarefas

Disponibilizar as informações de verificação para os consumidores de software.

5.2.3 Exemplo de implementação

- Usar uma autoridade de certificação (CA) estabelecida para assinatura de código, permitindo assim que os utilizadores verifiquem a validade das assinaturas.
- Verificar periodicamente os processos de assinatura de código, incluindo renovação e proteção de certificados.

5.3 Arquivar e proteger cada versão de software.

5.3.1 Descrição

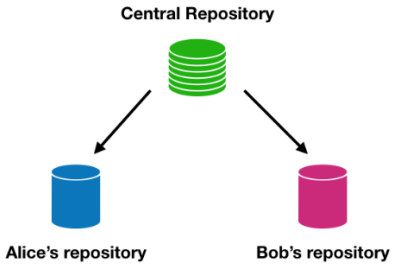
Prática deve ser usada para ajudar a identificar/analisar/eliminar vulnerabilidades descobertas no software após o lançamento.

5.3.2 Tarefas

Armazenar com segurança uma cópia de cada versão do software, incluindo o código, bibliotecas de terceiros, documentação e ainda algumas informações referentes á integridade de cada versão.

5.3.3 Exemplo de implementação´

Um exemplo implementação é o armazenar todos os ficheiros num repositório e restringir o acesso a esse repositório.



6 PRODUCE WELL-SECURED SOFTWARE (PW)

Neste grupo, é suposto que a organização produza software "well-secured" que possua um mínimo de vulnerabilidades de segurança nos seus lançamentos.

- PW.1 – Levar em consideração os requisitos de segurança e as informações de risco durante o design do software;
- PW.2 – Rever o software design para verificar a conformidade com os requisitos de segurança e informações de risco;
- PW.3 – Verifique se o software de terceiros está em conformidade com os requisitos de segurança;
- PW.4 – Reutilizar software existente e bem protegido quando possível, em vez de duplicar a funcionalidade;
- PW.4 – Desenvolver código aderente às práticas de codificação segura;
- PW.5 – Configurar os processos de compilação para melhorar a segurança de executável;
- PW.6 – Rever e analisar o código legível por humanos para identificar vulnerabilidades e verificar se está em conformidade com os requisitos de segurança pretendidos;
- PW.7 – Testar o código executável para identificar vulnerabilidades e verificar se está em conformidade com os requisitos de segurança;
- PW.8 – Configurar o software para ter configurações seguras por padrão;

6.1 PW.1

Nesta prática é vitável determinar quais os requisitos de segurança a implementar e ainda determinar quais os riscos à segurança que poderão surgir além de os mitigar. Com isto, é possível tornar o desenvolvimento mais eficiente. Aqui a elaboração de um modelo de ameaças ajuda a equipa a perceber tais riscos. Para isto, é necessário que as equipas recebam formação na construção de modelos de ameaças e que tenham em atenção áreas mais sensíveis ou de maior risco, de modo a proteger dados mais sensíveis.

6.2 PW.2

Permite ajudar a garantir que o software considere os requisitos de segurança impostos e atenda ás informações de risco identificadas. Tem como tarefa pegar num colaborador qualificado e que não esteja envolvido com o software design e usá-lo para rever e atestar se todos os requisitos de segurança estão em conformidades. Além deste exemplo, é aconselhado que se reveja os modelos de risco (modelo de ameaças) criados, de modo a verificar se realmente não falte nada.

6.3 PW.3

Prática responsável pela redução de riscos associados com o uso de módulos/serviços, que poderão ser uma potencial fonte de vulnerabilidades. Esta prática tem a tarefa de comunicar os requisitos aos fornecedores ou outra entidade que possa fornecer módulos ou até mesmo serviços à organização podendo estes serem reutilizados pela própria empresa. Isto é feito através de um pedido aos fornecedores desses mesmos módulos e serviços, "provas" onde nos diga que tais produtos estão em conformidade com os requisitos de segurança da empresa. Porem, também é aconselhado que se garanta que cada modulo/serviço usado, sofra de uma constante verificação e atualização pelo provedor, sendo assim possível corrigir vulnerabilidades nesses produtos.

6.4 PW.4

Com esta prática é possível reduzir os custos de desenvolvimento do software, e diminuir o risco de serem introduzidas novas vulnerabilidades no software. Mas tal fato, apenas poderá ser visto se nos softwares forem incorporadas técnicas e funcionalidades de segurança. Exemplo protocolos criptográficos. As tarefas são semelhantes com as que já forma referidas no ponto anterior, sendo que se baseiam na criação/obtenção de módulos, serviços, bibliotecas e outros de forma segura e claro, seguindo o SDLC.

Alguns exemplos de como implementar:

- Manter uma lista de software comercial aprovado.
- Seguir as práticas de segurança estabelecidas pela organização para o desenvolvimento seguro de software.
- Designar quais recursos de segurança devem ser suportados pelo software a ser desenvolvido.

6.5 PW.5 – PW.6 – PW.7

Práticas muito semelhantes, baseiam-se na diminuição do número de vulnerabilidades e na redução dos custos. Diferem apenas no momento em que é realizado, sendo que uma deverá ser posta em prática durante a criação do código fonte, e a outra durante a fase de testes. A prática numerada no documento como PW.7, permite se for bem implementada, ajudar a identificar vulnerabilidades antes do lançamento do software, possibilitando a correção de erros e vulnerabilidades antes que este seja disponibilizado para o publico e sofra explorações.

É aconselhado que sejam validados todos os inputs não confiáveis e programar bem todas as saídas. Evitar usar funções e chamadas que possam ser inseguras, além de usar um ambiente de desenvolvimento que ajude os colaboradores a codificar e seguir práticas seguras. Ao fazer uso de tal ambiente, também permitirá que o desenvolvedor reveja o próprio código em busca de falhas.

Em grande parte, os exemplos e ***Task's*** descritas destas práticas, consistem no uso de ferramentas (*automated tools*) no auxílio de correção de erros e diminuição de vulnerabilidades ou na documentação de erros e vulnerabilidades encontradas pelos próprios programadores.

Automated Tools

6.6 PW.8

Como na prática anterior, a prática de Testar o código executável permite ajudar a identificar de igual modo, vulnerabilidades antes que o produto(software) seja lançado e posteriormente corrigidas. O foco é na utilização de métodos/ferramentas automatizadas, resultando num menor esforço e num menor número de recursos usados pela empresa. Testes de penetração, ferramentas de *fuzz testing* entre outros exemplos de utilização. É ainda aconselhado que todo este processo seja documentado.

6.7 PW.9

Ao configurar o software para ter configurações seguras por padrão, faculta a melhorias a níveis de segurança do software no momento da instalação, tais medidas previnem ou diminuem o risco de o programa seja comprometido.

Essencialmente, ao determinar como as configurações são configuradas, isto se forem realizados testes que garantam que as configurações estejam a funcionar conforme esperado, evitando possíveis problemas operacionais, de compatibilidade ou outros. Tais configurações devem ser documentadas pela organização, descrevendo o que cada uma faz e que possível impacto teria se for alterada para outra.

7 RESPONDER A RELATÓRIOS DE VULNERABILIDADES(RV)

Por fim temos o último grupo, onde são identificadas as vulnerabilidades nas diferentes versões do software produzido e ainda é suposto efetuar uma ação rápida e adequada na resolução dessas vulnerabilidades como em outras que possam ocorrer no futuro. Dividido em três práticas, sendo elas:

- RV.1 – Identificar e Confirmar Vulnerabilidades continuadamente.
- RV.2 – Avaliar e priorizar a correção de todas as vulnerabilidades.
- RV.3 – Analisar vulnerabilidades para identificar as respetivas causas.

7.1 RV.1 – IDENTIFICAR E CONFIRMAR VULNERABILIDADES CONTINUADAMENTE.

7.1.1 Descrição

Ajuda a garantir que as vulnerabilidades sejam identificadas mais rapidamente, desta forma possibilitando uma rápida atuação por parte do corpo responsável de correção de bugs e vulnerabilidades da organização. O foco é reduzir a chance de um invasor conseguir explorar qualquer vulnerabilidade.

7.1.2 Tarefas e exemplos de implementação

Obter informações dos usuários, estabelecer periodicamente análises ou até mesmo testes ao software através de monitoramento a bases de dados como a NVD. O uso de ferramentas automatizadas como no grupo PW.7 permitiram uma mitigação de vulnerabilidades. Por último, mas não menos importante, possuir uma boa política de resposta a relatórios de vulnerabilidades descobertas no próprio software.

7.2 RV.2 – AVALIAR E PRIORIZAR A CORREÇÃO DE TODAS AS VULNERABILIDADES.

7.2.1 Descrição

Ao ser feito a avaliação e priorização a correção das vulnerabilidades, a organização terá o benefício de as vulnerabilidades sejam corrigidas num curto espaço de tempo.

7.2.2 Tarefas e exemplos

Aqui o objetivo seria pegar nas vulnerabilidades conhecidas do software e analisa-las de que forma seria possível explora-las e se seria necessário muitos recursos ou muito esforço por parte de uma atacante. Além disto, é importante que a organização esteja pronta para corrigir tais vulnerabilidades mesmo que elas ainda não tenham sido exploradas, a priorização também é fundamental, no caso do software possa possuir possíveis vulnerabilidades numa quantidade

anormal. A solução mais recomendada, como já referido enumeras vezes, é usar *tracking software* para documentar tais eventos.

7.3 RV.3 – ANALISAR VULNERABILIDADES PARA IDENTIFICAR AS RESPETIVAS CAUSAS.

7.3.1 Descrição

Reduz a frequências de futuras vulnerabilidades

7.3.2 Tarefas e exemplos

Isto pode ser conseguido através de analises a todas as vulnerabilidades e verificar o que causa tal vulnerabilidade. Fundamentalmente, as tarefas desta prática baseiam-se nas análises das causas das vulnerabilidades, na revisão do software e ainda na revisão do processo SDLC e atualiza-lo conforme for necessário. Reduzindo assim a probabilidade da causa do problema seja sempre a mesma. As organizações neste âmbito devem então documentar as descobertas feitas com todas as analises acima referidas.

Ciclo de vida da gestão de vulnerabilidades

8 CONCLUSÃO

Com este trabalho foi possível conhecer as práticas e diretivas que a **NIST** aconselha que as organizações abordem para manter e desenvolverem os seus softwares de forma segura. Infelizmente como é apenas um parecer e não uma obrigação, dificilmente os especialistas, principalmente os com mais anos de experiência irão mudar a sua forma de trabalhar porque a NIST recomenda. Mas no nosso ver, isto poderia ser também mitigado se houvesse realmente formações (workshops, por exemplo).

Apesar que neste relatório não foram colocados todos os exemplos de como implementar todas as práticas aconselháveis, mas tentamos escolher e abordar aquelas que achamos mais pertinentes.

Para concluir, podemos verificar que nem todas as práticas são de fácil implementação ou aplicáveis, mas tudo depende dos recursos e infraestrutura que a organização utilize no desenvolvimento de software.

9 BIBLIOGRAFIA

Armerding, T. (18 de Julho de 2019). *Synopsys*. Obtido de https://www.synopsys.com/blogs/software-security/nist-ssdf-secure-software-development-framework/

Donna Dodson, M. S. (2019). *Mitigating the Risk of Software Vulnerabilities by Adopting a Secure Software Development Framework (SSDF).* : National Institute of Standards and Technology.