Universidade do Minho

Mestrado Integrado em Engenharia Informática

Engenharia de Segurança

Projeto 1 - Desenvolvimento seguro de *software*

Práticas fundamentais para o desenvolvimento seguro de *software*

Grupo 2
Paulo Gameiro - A72067
Pedro Rodrigues - PG41092
Rafaela Soares - A79034

Braga, Portugal
23 de Março de 2020

Conteúdo

1 Introdução

O presente relatório consiste num conjunto de orientações para o desenvolvimento seguro de *software* - desde a fase inicial de conceptualização até à utilização. Estas orientações incorporam processos de desenvolvimento dinâmico e cobre algumas preocupações e abordagens sobre aplicativos *web* e *cloud services*.

O ritmo da inovação continua a aumentar drasticamente e muitas empresas de *software* mudaram os seus ciclos de desenvolvimento a favor de lançamentos altamente iterativos e mais frequentes, incluindo alguns que se tornaram "contínuos".

Adicionalmente, a dependência de componentes de terceiros, comerciais e *Open Source Software*, está em ascensão e estes são frequentemente tratados como caixas-negras, sendo que a sua análise é feita com um nível diferente de escrutínio comparativamente ao *software* desenvolvido internamente, sendo esta uma diferença que pode introduzir riscos.

Em sequência a isso, também existe a necessidade de respeitar e cumprir muitos *standards*, regulamentos, leis e portarias, sendo que as equipas de desenvolvimento podem sentir sérias dificuldades em cumprir requisitos de segurança.

Reconhecer estas preocupações como também a constante inovação dos métodos de ataque é o primeiro passo para a alteração de práticas inaceitáveis nos dias de hoje. Acredita-se que as medidas que serão apresentadas são fulcrais para a interiorização de novas práticas ou melhoria das mesmas, assim como para a (re)consideração das tecnologias utilizadas.

Nesse âmbito, foi estudada a publicação *Fundamental Practices for Secure Software Development* da organização SAFECode publicada a março de 2018, sendo que existem publicações mais recentes.

Palavras-chave: *Software*, *Threat Modeling*, *Vulnerability*

2 *Design*

Os autores advogam que um *software* a desenvolver deve englobar requisitos que façam jus às políticas de segurança interna, assim como às leis ou regulamentos externos. Deste modo, acreditam que a arquitetura e o *design* do *software* deve ser elaborado de forma a prevenir *exploits* a ameaças conhecidas baseadas no ambiente operacional do mesmo.

Nesse sentido, apresentam a elaboração de processo de modelagem de ameaças, consideração apropriada dos requisitos de segurança e princípios de *design* seguro como uma forma de facultar à equipa de desenvolvimento a identificação dos recursos de segurança que são necessários para proteger os dados (?, 2018, p.9).

2.1 *Threat Modelling*

A prevenção de *exploits* a ameaças conhecidas inicia-se no processo de levantamento de requisitos, sendo a segurança um requisito não funcional. Neste momento inicial de desenvolvimento, pode-se abordar a questão da segurança de forma abstrata, apenas considerando os hipotéticos componentes a serem utilizados.

Posto isto, pode-se proceder à modelagem de ameaças (*Threat Modelling*) - representação abstrata, numa certa perspetiva, de um sistema, como auxílio para decisões ou projeções acerca do mesmo -, no intuito de se identificar as possíveis e hipotéticas ameaças, riscos, vulnerabilidades e contramedidas do sistema em questão.

Este processo apresenta diversos benefícios, uma vez que se pode identificar e solucionar falhas, mesmo antes da fase de implementação do código. Portanto, é necessário que seja feito no início do ciclo de vida do desenvolvimento.

2.2 Princípios de *design* seguro

Adicionalmente aos **princípios de *design* de sistema seguro publicados por Jerome Saltzer e Michael Schroeder** - *Economy of mechanism*, *Fail-safe defaults*, *Complete mediation*, *Least privilege*, *Least common mechanism*, *Psychological acceptability* e *Compromise recording* -, referem que existem outros princípios fulcrais para a segurança dos sistemas de *software*, sendo estes (?, 2018, p.10):

- ***Defense in depth***: projetar o sistema para resistir a ataques, mesmo que isso implique a descoberta de uma única vulnerabilidade ou rejeição de um único recurso. Pode ser necessário envolver a inclusão de vários níveis de mecanismos de segurança ou a criação de um sistema que se bloqueie, em vez de permitir que um invasor obtenha controle completo do mesmo.
- ***Fail securely***: em contrapartida à *Defense in depth*, espera-se que um sistema permaneça seguro, mesmo que encontre um erro ou se bloqueie.
- ***Design for updating:***: uma vez que é indubitável o facto de que o sistema se possa deparar com vulnerabilidades, os desenvolvedores devem planejar a instalação segura e confiável de atualizações de segurança.

De referir que a revisão da arquitetura e *software* deve ser feita com base nos princípios acima referidos, uma vez que um sistema mal projetado pode comprometer o sistema, devido a um agente mal intencionado poder-se aproveitar de uma falha de *design* ou lógica.

2.3 Desenvolvimento de uma estratégia de criptografia

É indubitável que a forma mais eficaz e económica de considerar a criptografia é durante o processo de *design*. Contudo, o desenvolvimento de uma estratégia de criptografia exige um processo cognitivo denso.

No intento de facultar tal procedimento, é sugerido alguns componentes principais para uma estratégia de criptografia (?,2018, p.11):

- **Definições sobre o que proteger**: relativamente ao tráfego na Internet, defende-se que este deve encriptado durante o transporte e não deve ser feito em redes privadas apenas com base em uma razão sólida. No que concerne ao armazenamento de dados em base de dados, arquivos, entre outros não menos relevantes, é necessário a criação de critérios claros para categorizar os dados a serem criptografados, assim como a consideração de mecanismos aceitáveis.

- **Designação dos mecanismos a serem usados para criptografia**: a vasta opção de algoritmos de criptografia, bibliotecas criptográficas, chaves e técnicas deve ser considerada pormenorizadamente, de forma a evitar hipotéticas escolhas erradas. Aconselha-se o desenvolvimento de padrões claros de criptografia, utilização de bibliotecas criptográficas examinadas previamente, utilização de algoritmos fortes e comprimentos de chave.

- **Decisão sobre uma solução de gestão de chaves e certificados**: controlar quem (seja uma pessoa ou um serviço) tem acesso e fornecer uma auditoria clara de *log* do mesmo. No que diz respeito aos mecanismos de chaves e certificados, deve-se implementar novas chaves e certificados, assim que os anteriores estiverem perto do prazo de expiração. De notar que as chaves codificadas (ou outros segredos) no código fonte devem ser evitadas, uma vez que se tornam vulneráveis.

- **Implementação com agilidade criptográfica em mente**: especificar como é que as aplicações e serviços devem implementar a criptografia, de forma a permitir a transição para novos mecanismos criptográficos, bibliotecas e chaves quando necessário.

2.4 Padronização da gestão de identidade e acesso

A verificação de identidade num determinado acesso pode ser necessária, uma vez que algumas ações requerem autorização. Assim, é necessário estabelecer controlos de acesso, de forma a padronizar uma abordagem de autenticação e autorização.

A identificação do local onde as verificações de autorização devem ser implementadas é definida pelo processo de *Threat Modelling* referido acima e a obtenção de um mecanismo robusto deve-se também à adoção dos princípios de *design* seguro.

Segue-se, de seguida, o que deve ser incluído na gestão de acesso(?,2018,p.13):

- **Mecanismo de autenticação**: deve ser estabelecido uma forma de garantir que o utilizador é quem diz que é.

- **Mecanismo(s) pelo qual(is) um serviço ou componente lógico se autentica para outro e como as credenciais são armazenadas**: as credenciais de serviço não devem ser armazenadas num repositório de origem.

- **Mecanismo(s) que autoriza(m) as ações de cada principal**: implementar uma estratégia de autorização em base do menor privilégio, para que todos os principais tenham apenas o número mínimo de permissões necessárias para concluir as suas tarefas e nenhuma permissão para executar tarefas fora do conjunto permitido.

2.5 Estabelecimento de requisitos de *Log* e práticas de auditoria

No âmbito de se armazenar detalhes relativos a um hipotético incidente de segurança, utilizar arquivos *log* de aplicações, sistemas e segurança é uma ótima alternativa.

A criação e manutenção dos *logs* engloba um conjunto crítico de decisões de *design* de *software*, sendo este:

- Restringir a captura de informações apenas aos dados necessários;

- Identificar cuidadosamente quais das informações de segurança são relevantes e precisam ser registadas;

- Identificar cuidadosamente onde os *logs* serão armazenados;
- Identificar cuidadosamente por quanto tempo os *logs* serão mantidos e como serão protegidos;
- Usar recursos de *log* do sistema operacional a utilizar ou outras ferramentas estabelecidas, em vez de se criar uma nova infraestrutura de *log*;
- Verificar se os *logs* de segurança podem ser configurados durante o tempo de execução, o que permite fornecer alertas mais eficazes para investigar eventos de segurança;
- Não excluir arquivos de *log* locais rapidamente, se estes não se encontrarem num local central e arquivado.

Este conjunto baseia-se nos requisitos de negócios e do sistema, pelo modelo de ameaça e pela disponibilidade das funções de criação e manutenção de *logs* no ambiente implementado (?,2018,p.14).

3 Práticas de programação segura

No âmbito de se evitar hipotéticos erros de programação, é fundamental convergir para a minimização do número de vulnerabilidades de segurança não intencionais a nível de código.

Para esse efeito, pode-se recorrer à definição de padrões de programação - seleção de linguagens, estruturas e bibliotecas mais apropriadas e seguras. Para além disso, é imprescindível a garantia de uso adequado destes padrões e revisão manual do código(?,2018,p.15).

Apresenta-se, de seguida, algumas práticas a se adotar.

3.1 Estabelecer padrões e convenções de programação

Considerar os problemas de segurança que advêm das tecnologias escolhidas no intuito de:

- Informar toda a equipa de desenvolvimento acerca de tecnologia a adoptar, padrões, convenções e reutilização de programação;
- Escolher uma solução de resolução de um problema em questão como padrão;
- Investir na reutilização de recursos;
- Inicializar a ponderação de testes e consequente validação.

3.2 Utilizar apenas funções seguras

Uma vez que várias linguagens de programação possuem funções inseguras, é recomendado a implementação de ferramentas de auxílio na identificação e revisão do uso de funções perigosas.

Para além disso, é também recomendado a utilização de versões atuais do compilador e *Toolchain*, assim como opções seguras de compilador. Como, ao longo do tempo, são melhorados os recursos de segurança, é essencial utilizar as versões mais recentes.

De salientar que a ativação das opções seguras do compilador e não desativação dos padrões seguros por questões de desempenho ou compatibilidade com versões anteriores é uma mais valia.

3.3 Utilizar ferramentas de análise de código para localizar problemas de segurança antecipadamente

Como as *frameworks* têm limitações de proteção e existe a possibilidade de serem introduzidos erros pelos desenvolvedores, recomenda-se a utilização de ánalise de código.

3.4 Manipular dados com segurança

Todos os dados provenientes do utilizador devem ser considerados como não seguros, uma vez que a origem da mesma pode não estar definida ou ser incorretamente compreendida. Pretende-se evitar que esses dados se transformem em código de execução ou forneçam acesso não intencional a recursos.

Normalmente, a validação de entrada é identificada como uma defesa aplicada na fronteira com a Internet, o que não é uma abordagem adequada. A melhor abordagem é garantir que cada área da pilha de componentes se defenda contra as entradas maliciosas, sendo validado o *input*.

Adicionalmente à validação, podem ser agregadas outras técnicas, tais como:

- **Codificação**: transformação dos dados no intuito de serem interpretados num contexto específico.
- ***Canonicalization***: processo de tradução de diferentes representações em um formato consistente.
- ***Sanitization***: remoção de dados não pertencentes ao previamente estabelecido.

Caso os dados sejam complexos e densos, é sugerido a utilização de uma biblioteca criada por especialistas nos formatos pretendidos.

3.5 Lidar com erros

A existência de erros é inevitável, mesmo com a realização de testes de validação. Por esse motivo, é necessário que um sistema seja capaz de manipular e reagir, da melhor forma possível, a cenários imprevisíveis, apresentando uma mensagem de erro, por exemplo. Posto isto, é recomendado utilizar manipuladores de erro genéricos ou manipuladores de exceção para cobrir erros imprevistos.

De referir que a notificação do erro ao utilizador não deve relevar detalhes técnicos do problema, devendo esta ser genérica. E que a notificação aos administradores deve ser feita pela abordagem de registo em *log*.

4 Gerir riscos associados à utilização de *software* de terceiros

De forma a conseguir oferecer uma melhor qualidade nos seus produtos, os programadores têm apostado, cada vez mais, na utilização de *frameworks* e bibliotecas de terceiros.

A utilização destes componentes, apesar das vantagens que traz associadas à reutilização de componentes que permite uma maior eficácia no desenvolvimento, traz também riscos inerentes, tendo em conta que, normalmente, não é possível perceber o que contém o código que está a ser importado e, como tal, contém desvantagens relativamente ao conteúdo desenvolvido internamente.

De modo geral, deve-se escolher *frameworks* e bibliotecas estáveis e conhecidas como seguras, que forneçam defesa contra ameaças conhecidas.

Sem a realização de testes intensivos a estes *frameworks*, abre-se a porta para a ocorrência de falhas de segurança que podem ser difíceis de detetar, deixando os programadores e os utilizadores com uma falsa sensação de segurança.

5 Testes e Validação

Uma parte essencial no desenvolvimento de *software* seguro é a realização de testes de segurança. É fundamental principalmente em organizações que não seguem muitas práticas de desenvolvimento seguro, sendo que a realização de testes de segurança é útil na identificação de fragilidades no produto ou serviço.

No caso de organizações que seguem as normas de segurança, a realização deste tipo de testes é importante para validar a qualidade das suas políticas de desenvolvimento.

5.1 Testes Automáticos

Os testes automáticos de segurança permitem a realização de vários testes rapidamente e repetidamente a uma maior escala e podem ser realizados utilizando ferramentas existentes que permitem a deteção de determinados tipos de vulnerabilidades.

Ainda assim, enquanto estas ferramentas permitem resultados eficientes e rápidos, apresentam também um grande número de falsos positivos e negativos do que testes realizados manualmente, principalmente porque as ferramentas atuais de testes automáticos não aprendem à medida que vão descobrindo novas formas de deteção.

5.1.1 Utilizar Ferramentas de Análises Estáticas de Testes de Segurança

As análises estáticas são realizadas através da procura de falhas no código fonte ou na linguagem de compilação intermédia, onde é possível detetar padrões conhecidos baseados na lógica da aplicação, ao contrário da identificação de falhas na aplicação em *runtime*, que não é tão eficiente.

A utilização de ferramentas de análises estáticas, que seguem uma série de padrões para encontrar falhas, beneficiam da experiência de pessoas que ajustam a sua configuração, de forma a tornar os resultados mais fiáveis para os programadores e reduzir o número de falsos positivos.

5.1.2 Realizar Análises Dinâmicas de Testes de Segurança

As análises dinâmicas de testes de segurança são executadas numa determinada versão de um programa ou serviço, normalmente realizando uma série de ataques pré-construídos de forma automatizada seguindo o que se espera que sejam os passos que um atacante realizaria.

Estes testes são executados contra a versão completa do *software* enquanto este é executado e, portanto, permitem testar cenários que são apenas possíveis quando todos os componentes estão integrados, ao contrário dos testes estáticos.

Apesar disso, estes testes dinâmicos trazem desvantagens relativamente aos estáticos, principalmente porque são mais lentos.

5.1.3 *Fuzz Parsers*

Fuzzing é uma forma especializada de testes dinâmicos, normalmente descrita como o ato de gerar ou alterar dados e transferi-los para os analisadores de dados da aplicação e verificar como reagem.

O benefício que este método traz é que uma vez configurada a automatização, podem ser executados inúmeros testes contra o código, sendo o único custo o tempo de computação. No entanto, a sua configuração pode muitas vezes ser demorada.

5.1.4 Analisar Vulnerabilidades na Rede

Esta categoria de testes está mais relacionada com a segurança de redes e da informação porque as ferramentas de detecção de vulnerabilidades recorrem a vulnerabilidades previamente descobertas, mas não são tão úteis em testes realizados em aplicações recentes.

Este tipo de testes é normalmente utilizado em aplicações executadas em máquinas onde a segurança do sistema operativo onde se encontram é fundamental.

5.1.5 Verificar Configurações Seguras e Usar Plataformas de Mitigação

A grande maioria da automatização da segurança tem como objectivo a identificação de vulnerabilidades de segurança, mas é também extremamente importante garantir que o *software* mantenha configurações que permitam a mitigação de problemas que possam aparecer.

Uma verificação regular de que o *software* e os serviços estão a usar correctamente as plataformas de mitigação disponíveis facilita bastante o *deployment* e a execução do software.

5.1.6 Realizar Testes Automáticos das Funcionalidades de Segurança

As organizações que utilizem testes unitários e outro tipo de testes automáticos de verificação de funcionalidades gerais da aplicação devem estender esses mesmos mecanismos de forma a verificar também as funcionalidades de segurança presentes no *software*.

Verificar que uma determinada funcionalidade está correta não é muito diferente de verificar que as funções de segurança se encontram a funcionar como previsto e, como tal, devem ser adotadas as mesmas estratégias de teste.

5.2 Testes Manuais

Os testes manuais requerem normalmente mais trabalho e mais recursos do que os testes automáticos. Requerem também o mesmo investimento de tempo sempre que se pretender realizar o mesmo teste.

Por outro lado, os humanos têm uma maior facilidade de aprendizagem e conseguem evoluir na identificação de falhas e problemas.

5.2.1 Realizar Verificações Manuais das Funcionalidades de Segurança

Tal como no caso das verificações automáticas, as funcionalidades de segurança devem também ser incluídas nos testes unitários. As organizações que realizam garantias de qualidade manuais devem incluir a verificação de funções e mitigações de segurança.

As organizações que realizam testes automáticos devem continuar a realizar testes automáticos, tendo em conta que um utilizador poderá identificar uma falha num botão ou caixa de texto mas não conseguirá perceber se os *logs* de segurança estão a ser feitos corretamente.

5.2.2 Realizar Testes de Penetração

Os testes de penetração são realizados tendo em vista o pensamento que um atacante teria ao tentar encontrar vulnerabilidades na aplicação. Conseguem encontrar uma maior variedade de vulnerabilidades e analisar o *software* ou serviço num grande contexto, baseando-se no ambiente em que corre, as ações que realiza ou até as formas como os utilizadores vão interagir com o *software*.

Os *penetration testers* vão aprendendo com experiências anteriores e aprimorando as suas formas de testar o *software*, adaptando-se assim à evolução dos atacantes à medida que vão surgindo novas vulnerabilidades.

Contudo, os testes de penetração requerem um maior tempo para a realização destes testes e requerem também uma grande experiência dos trabalhadores especializados. São também desvantajosas em termos de escalabilidade sendo que um aumento da necessidade de testes requer também mais trabalhadores.

Este tipo de testes é especialmente fundamental em funções e componentes críticos na segurança do software. E tendo em conta a reduzida disponibilidade e elevada despesa deste tipo de trabalhadores, os testes de penetração tendem a ser realizados apenas depois de serem feitos outros tipos de testes mais baratos.

6 Gerir problemas de segurança encontrados

Um dos principais objetivos do desenvolvimento seguro é a identificação de falhas no *design* ou implementação do *software* que quando exploradas por atacantes podem comprometer a aplicação ou a empresa a um nível de risco.

As categorias de teste descritas anteriormente levam a descobertas relacionadas com a segurança da aplicação, e devem ser identificados de maneira a ser possível prevenir, mitigar ou aceitar o respetivo risco associado. No caso de uma descoberta ser prevenida o risco é eliminado, já no caso da mitigação o risco é reduzido mas não completamente eliminado.

De forma a garantir que algo é realizado tendo em conta cada descoberta, tudo isto deve ser gravado num sistema que permita a diversas equipas na organização ter essa informação disponível.

6.1 Definir gravidade

É necessário existir uma definição clara do que é a gravidade para garantir que existe uma consistência na forma como é tratada a segurança e o seu potencial impacto.

Inicialmente, deve-se implementar uma escala de gravidade (de **Muito Baixo** a **Muito Alto** por exemplo), e definir qual o critério para atribuir a cada categoria.

6.2 Aceitar Risco

Quando um problema identificado não consegue ser resolvido ou completamente mitigado é necessário uma aprovação de que é possível lançar o produto com o risco inerente.

Deve existir uma identificação de quem pode submeter, rever e aprovar os pedidos de aprovação.

O objetivo desta estrutura é obter uma consistência que ajude todos os envolvidos a tomar uma decisão que envolva o menor número de risco para a aplicação final.

A aceitação do risco deve ser documentada juntamente com o nível de gravidade associado, bem como um plano de prevenção ou expiração.

7 Divulgação e resposta às vulnerabilidades

É necessário ter preparado um processo de reposta às vulnerabilidades que possam ser encontradas depois do lançamento do *software* ou serviço, que é importante para manter os *stakeholders* informados dos problemas encontrados. O objetivo é manter os clientes informados e com os *updates* necessários para resolver as vulnerabilidades.

Existem dois *standards* ISO que fornecem passos detalhados de como implementar um plano de resposta.

- ISO/IEC 29147
- ISO/IEC 30111

7.1 Definir Políticas Internas e Externas

Uma política de resposta e divulgação de vulnerabilidades deve considerar uma série de orientações relativamente à resposta interna e externa de vulnerabilidades dentro de cada organização.

O processo de divulgação será certamente diferente em casos em que a vulnerabilidade tenha sido divulgada internamente e externamente e, como tal, a documentação da polícia deverá ser diferente para cada caso.

7.2 Definir Funções e Responsabilidades

Grandes organizações tendem a estabelecer uma equipa de resposta a incidentes (Product Security In cident Response Teams *(PSIRT)*) em que é definida um processo de resposta às vulnerabilidades, enquanto que empresas mais pequenas terão de definir esta equipa a uma escala mais pequena e com base no número de produtos e incidentes.

7.3 Garantir que os repórteres de vulnerabilidades sabem quem contactar

De forma a facilitar uma correta resposta a uma potencial vulnerabilidade, é importante que os investigadores de segurança, clientes e *stakeholders* consigam perceber rapidamente para onde contactar uma vulnerabilidade, portanto este contacto deve estar acessível, por exemplo num *website* da empresa. Esta informação sobre as vulnerabilidades deve ser mantida sempre confidencial.

7.4 Gerir repórteres de vulnerabilidades

As vulnerabilidades podem ser reportadas por repórteres de vulnerabilidades e por utilizadores e, portanto, aquando da recepção de uma marcação, deve-se ter preparada a forma como a empresa vai reagir.

7.5 Monitorizar e Gerir vulnerabilidades de componentes de terceiros

É extremamente importante manter um processo para monitorizar e gerir vulnerabilidades encontradas em componentes de terceiros, de forma a não comprometer o *software*.

7.6 Resolver a vulnerabilidade

Logo após a identificação da vulnerabilidade, a equipa que mantém o código onde se encontra deve fazer a sua identificação, determinando qual o nível de gravidade de forma a estabelecer prioridades na sua resolução.

Outras considerações deverão ser tidas em conta relativamente à urgência, tal como o potencial impacto e a probabilidade de ser explorada.

Para além da resolução deste problema, a equipa de desenvolvimento deve considerar a natureza do erro, tendo em vista a possibilidade de este existir noutras partes do código em que foi usada a mesma classe.

7.7 Divulgação da vulnerabilidade

Quando a solução estiver disponível, deve ser comunicada aos clientes através de consultores e devem notificar, ao mesmo tempo, todas as versões do produto afetadas.

Este processo de divulgação pode ser diferente quando a vulnerabilidade está a ser ativamente atacada, sendo que, neste caso, poderá ser necessário lançar um aviso público mesmo antes de ser encontrada uma solução para a vulnerabilidade.

A divulgação deve conter um balanço entre dar dados suficientes para que os utilizadores se protejam e não dar dados que possam ser utilizados por partes maliciosas.

A informação a ser divulgada deve ser a seguinte:

- Produtos e versões afetadas
- Nível de gravidade
- CVE
- Descrição e potencial impacto
- Detalhes da solução
- Créditos para quem descobriu a vulnerabilidade

7.8 *Feedback* do ciclo de vida do desenvolvimento seguro

De forma a evitar vulnerabilidades semelhantes em produtos novos, deve-se efectuar um *loop* contínuo de análises desde a raiz do problema encontrado.

8 Planeamento e implementação de práticas seguras de desenvolvimento

8.1 Cultura da empresa

Enquanto que em algumas empresas, os trabalhadores respondem melhor a pedidos dos CEOs e gestores de topo, noutras respondem melhor a pedidos de uma equipa de engenharia e, como tal, a cultura da empresa deve ser analisada de forma a perceber qual a forma mais eficaz.

8.2 Experiência da empresa

Cada um dos trabalhadores da empresa deve estar ciente da importância da segurança e deve ser-lhe dada formação sobre os detalhes técnicos necessários. Para cada uma das práticas de desenvolvimento seguro implementadas, deve ser considerado o nível de experiência dentro da organização, sendo que deve manter sempre a quantidade necessária de especialistas em cada área.

8.3 Ciclo de vida e modelo do desenvolvimento do produto

Apesar de práticas de segurança executadas numa ordem sequencial terem grandes ganhos a nível de segurança, outro tipo de *triggers* tais como tempo ou resposta às alterações ao ambiente de resposta devem também ser considerados.

De forma a facilitar alterações a maior escala, o processo e as ferramentas de segurança devem ser integradas no trabalho dos engenheiros, o que permite reduzir a fricção com os mesmos e permite também um mais rápido avanço no desenvolvimento.

8.4 Âmbito do desenvolvimento inicial

Ter um bom conhecimento de como o processo de planeamento do projeto funciona e qual a cultura da empresa ajuda o gestor do programa de desenvolvimento seguro a fazer melhores escolhas relativamente à implementação e adoção de novas práticas.

8.5 Propostas de valor

Software de segurança é uma área em que é bastante difícil efetuar medições, sendo que algumas mais comuns incluem modelos de custo de qualidade, nível de gravidade e de *exploitability* e ainda o custo de ataques ao produto.

9 Exemplos de ferramentas de segurança de *software*

Posteriormente à publicação alvo de estudo, apresenta-se as seguintes ferramentas de segurança de *software*.

- codiscope
- continuumsecurity
- OSSEC
- sonatype
- servicenow
- Checkmarx
- BLACKDUCK
- LogRhythm
- Qualys
- WhiteSource
- tripwire
- SNORT
- VERACODE
- Signal Sciences

Algumas destas ferramentas são *frameworks* que detetam vulnerabilidades e apresentam as respetivas possíveis alterações a serem feitas e outras são sistemas de *logs* e monitorização de tráfego.

10 Considerações finais

A criação de *software* abrange o desenvolvimento e implementação de métodos e processos para garantir que o *software* funcione conforme o planeado e livre de defeitos de *design* e falhas de implementação. Nesse sentido, o documento alvo de estudo provê de um esforço para auxiliar a aprimorar ou iniciar práticas fundamentais para o desenvolvimento seguro de *software*

À medida que as ameaças e os métodos de ataque continuam a evoluir, também os processos, técnicas e ferramentas para desenvolver *software* seguro. Posto isto, o documento alvo de estudo aborda mais elementos relativos ao ciclo de vida de desenvolvimento seguro de *software*, não sendo um guião exaustivo sobre o tema. Pelo contrário, destina-se a fornecer um conjunto básico de práticas de desenvolvimento seguras.

11 Bibliografia

- XeniaLabs, "The Ultimate List of Software Security Tools", disponível em https://xebialabs.com/the-ultimate-devops-tool-chest/security/?fbclid=IwAR2rx4jHC5KuP1o8ltAoQBvl4\discretionary{-}{}{}KEmNQ6GMx3Z5he05uKeyHTjpDm36oYIlE (acedido a 21 de Março de 2020)
- Dzone, "7 Free Security Tools That All Developers Will Want in Their Toolbox", disponível em https://dzone.com/articles/7-free-security-tools-that-all-developers-will-wan?fbclid=IwAR3E-9ruwzu9skAXe1x42bG37b53Toi8JPyyTUvJU_aAGa87IqwZaB-Yg0A (acedido a 21 de Março de 2020)
- Microsoft, "MIcrosoft Security development lifecicly Practices", disponível em https://www.microsoft.com/en-us/securityengineering/sdl/practices?fbclid=IwAR0H0_PIANS9DyKLC_hbjS7rMyuZAXuxd0mgUM2xDkp2aAk6a77EyRKdXSE (acedido a 21 de Março de 2020)
- CyBOK, "SECURE SOFTWARE LIFECYCLE", disponível em https://www.cybok.org/media/downloads/Secure_Software_Lifecycle_KA_-_draft_for_review_April_2019.pdf (acedido a 21 de Março de 2020)
- IBM, "Security in Development", disponível em https://www.redbooks.ibm.com/redpapers/pdfs/redp4641.pdf (acedido a 21 de Março de 2020)
- SAFECode, "Fundamental Practices for Secure Software Development", disponível em https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf (acedido a 21 de Março de 2020)