Machine Translated by Google

WEB SECURITY TESTING GUIDE

VERSION 4.2

Elie Saad
Rick Mitchell

owasp.org

Machine Translated by Google

Introdução

O Projeto de Teste OWASP

O Projeto de Teste OWASP está em desenvolvimento há muitos anos. O objetivo do projeto é ajudar as pessoas a entender o **quê, por que, quando, onde** e **como** testar aplicações web. O projeto forneceu uma estrutura de testes completa, e não apenas uma simples lista de verificação ou prescrição de questões que deveriam ser abordadas. Os leitores podem usar esta estrutura como modelo para construir seus próprios programas de testes ou para qualificar os processos de outras pessoas. O Guia de Teste descreve detalhadamente a estrutura geral de testes e as técnicas necessárias para implementar a estrutura na prática.

Escrever o Guia de Teste provou ser uma tarefa difícil. Foi um desafio obter consenso e desenvolver conteúdos que permitissem às pessoas aplicar os conceitos descritos no guia, ao mesmo tempo que lhes permitissem trabalhar no seu próprio ambiente e cultura. Também foi um desafio mudar o foco dos testes de aplicações web, de testes de penetração para testes integrados no ciclo de vida de desenvolvimento de software.

Porém, o grupo está muito satisfeito com os resultados do projeto. Muitos especialistas do setor e profissionais de segurança, alguns dos quais são responsáveis pela segurança de software em algumas das maiores empresas do mundo, estão validando a estrutura de testes. Esta estrutura ajuda as organizações a testar suas aplicações web para construir software confiável e seguro. A estrutura não destaca simplesmente áreas de fraqueza, embora isso seja certamente um subproduto de muitos dos guias e listas de verificação do OWASP. Como tal, decisões difíceis tiveram que ser tomadas sobre a adequação de certas técnicas e tecnologias de teste. O grupo compreende perfeitamente que nem todos concordarão com todas estas decisões. No entanto, o OWASP é capaz de assumir uma posição elevada e mudar a cultura ao longo do tempo através da consciencialização e da educação, com base no consenso e na experiência.

O restante deste guia está organizado da seguinte forma: esta introdução cobre os pré-requisitos para testar aplicações web e o escopo dos testes. Ele também cobre os princípios de testes e técnicas de teste bem-sucedidos, práticas recomendadas para relatórios e casos de negócios para testes de segurança. O Capítulo 3 apresenta o OWASP Testing Framework e explica suas técnicas e tarefas em relação às diversas fases do ciclo de vida de desenvolvimento de software. O Capítulo 4 aborda como testar vulnerabilidades específicas (por exemplo, SQL Injection) por meio de inspeção de código e testes de penetração.

Medindo a segurança: a economia do software inseguro

Um princípio básico da engenharia de software é resumido em uma citação de Controlling Software Projects: Management, Medição e estimativas por Tom DeMarco:

> Você não pode controlar o que não pode medir.

Os testes de segurança não são diferentes. Infelizmente, medir a segurança é um processo notoriamente difícil.

Um aspecto que deve ser enfatizado é que as medições de segurança tratam tanto de questões técnicas específicas (por exemplo, quão prevalente é uma determinada vulnerabilidade) quanto de como essas questões afetam a economia do software. A maioria dos técnicos compreenderá pelo menos os problemas básicos ou poderá ter uma compreensão mais profunda das vulnerabilidades. Infelizmente, poucos são capazes de traduzir esse conhecimento técnico em termos monetários e quantificar o custo potencial das vulnerabilidades para o negócio do proprietário da aplicação. Até que isso aconteça, os CIOs não serão capazes de desenvolver um retorno preciso sobre o investimento em segurança e, subsequentemente, atribuir orçamentos apropriados para segurança de software.

Embora estimar o custo de software inseguro possa parecer uma tarefa difícil, tem havido uma quantidade significativa de trabalho nessa direção. Em 2018, o Consórcio para Qualidade de Software de TI resumiu:

> …o custo do software de má qualidade nos EUA em 2018 foi de aproximadamente 2,84 biliões de dólares…

A estrutura descrita neste documento incentiva as pessoas a medir a segurança durante todo o processo de desenvolvimento. Eles podem então relacionar o custo do software inseguro ao impacto que ele tem nos negócios e, consequentemente,

Machine Translated by Google

desenvolver processos de negócios apropriados e atribuir recursos para gerenciar o risco. Lembre-se de que medir e testar aplicações web é ainda mais crítico do que para outros softwares, uma vez que as aplicações web são expostas a milhões de usuários através da Internet.

O que é teste?

Muitas coisas precisam ser testadas durante o ciclo de vida de desenvolvimento de uma aplicação web, mas o que realmente significa testar? O Dicionário Oxford de Inglês define “teste” como:

> **teste** (substantivo): um procedimento destinado a estabelecer a qualidade, desempenho ou confiabilidade de algo, especialmente antes de ser amplamente utilizado.

Para os fins deste documento, teste é um processo de comparação do estado de um sistema ou aplicativo em relação a um conjunto de critérios. Na indústria de segurança, as pessoas frequentemente testam um conjunto de critérios mentais que não são bem definidos nem completos. Como resultado disso, muitos estrangeiros consideram os testes de segurança uma arte negra. O objetivo deste documento é mudar essa percepção e tornar mais fácil para pessoas sem conhecimento profundo de segurança fazer a diferença nos testes.

Por que realizar testes?

Este documento foi elaborado para ajudar as organizações a compreender o que compreende um programa de testes e a identificar as etapas que precisam ser realizadas para construir e operar um programa moderno de testes de aplicações web. O guia oferece uma visão ampla dos elementos necessários para criar um programa abrangente de segurança de aplicações web. Este guia pode ser usado como referência e metodologia para ajudar a determinar a lacuna entre as práticas existentes e as melhores práticas do setor. Este guia permite que as organizações se comparem com seus pares do setor, para compreender a magnitude dos recursos necessários para testar e manter software ou para se preparar para uma auditoria. Este capítulo não entra em detalhes técnicos sobre como testar um aplicativo, pois a intenção é fornecer uma estrutura organizacional de segurança típica. Os detalhes técnicos sobre como testar uma aplicação, como parte de um teste de penetração ou revisão de código, serão abordados nas partes restantes deste documento.

Quando testar?

A maioria das pessoas hoje não testa software até que ele já tenha sido criado e esteja na fase de implantação de seu ciclo de vida (ou seja, o código tenha sido criado e instanciado em um aplicativo Web funcional). Esta é geralmente uma prática muito ineficaz e de custo proibitivo. Um dos melhores métodos para evitar o aparecimento de bugs de segurança em aplicações de produção é melhorar o Ciclo de Vida de Desenvolvimento de Software (SDLC), incluindo segurança em cada uma de suas fases. Um SDLC é uma estrutura imposta ao desenvolvimento de artefatos de software. Se um SDLC não estiver sendo usado em seu ambiente, é hora de escolher um! A figura a seguir mostra um modelo SDLC genérico, bem como o custo crescente (estimado) para corrigir bugs de segurança em tal modelo.

Machine Translated by Google

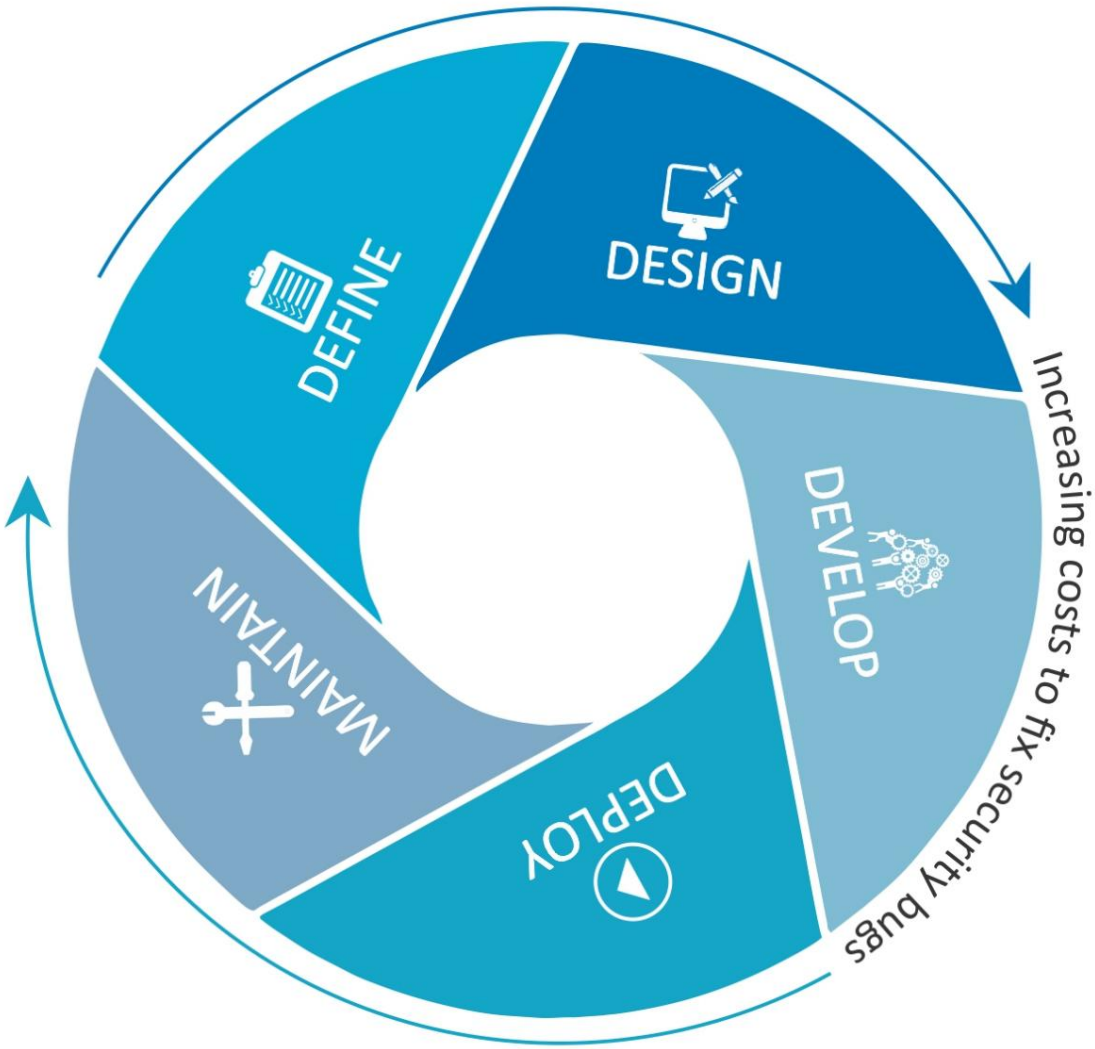

Figura 2-1: Modelo SDLC genérico

As empresas devem inspecionar seu SDLC geral para garantir que a segurança seja parte integrante do processo de desenvolvimento.
Os SDLCs devem incluir testes de segurança para garantir que a segurança seja adequadamente coberta e que os controles sejam eficazes durante todo o processo de desenvolvimento.

O que testar?

Pode ser útil pensar no desenvolvimento de software como uma combinação de pessoas, processos e tecnologia. Se estes são os fatores que “criam” o software, então é lógico que estes são os fatores que devem ser testados. Hoje, a maioria das pessoas geralmente testa a tecnologia ou o próprio software.

Um programa de testes eficaz deve ter componentes que testem o seguinte:

- **Pessoas** – para garantir que haja educação e sensibilização adequadas;
- **Processo** – para garantir que existem políticas e padrões adequados e que as pessoas sabem como seguir essas políticas;
- **Tecnologia** – para garantir que o processo tenha sido eficaz na sua implementação.

Machine Translated by Google

A menos que seja adoptada uma abordagem holística, testar apenas a implementação técnica de uma aplicação não revelará vulnerabilidades de gestão ou operacionais que possam estar presentes. Ao testar pessoas, políticas e processos, uma organização pode detectar problemas que mais tarde se manifestariam em defeitos na tecnologia, erradicando assim os bugs antecipadamente e identificando as causas raízes dos defeitos. Da mesma forma, testar apenas alguns dos problemas técnicos que podem estar presentes num sistema resultará numa avaliação da postura de segurança incompleta e imprecisa.

Denis Verdon, chefe de segurança da informação da Fidelity National Financial, apresentou uma excelente analogia para esse equívoco na Conferência OWASP AppSec 2004 em Nova York:

> Se os carros fossem construídos como aplicações... os testes de segurança assumiriam apenas o impacto frontal. Os carros não seriam testados quanto à estabilidade em manobras de emergência, eficácia dos freios, impacto lateral e resistência ao roubo.

Como fazer referência a cenários WSTG

Cada cenário possui um identificador no formato WSTG-<category>-<number> , onde: 'categoria' é uma string maiúscula de 4 caracteres que identifica o tipo de teste ou fraqueza, e 'número' é um valor numérico preenchido com zeros de 01 a 99. Por exemplo: WSTG-INFO-02 é o segundo teste de coleta de informações.

Os identificadores podem mudar entre versões, portanto é preferível que outros documentos, relatórios ou ferramentas usem o formato: WSTG-<versão>-<categoria>-<número> , onde: 'versão' é a tag da versão com pontuação removida. Por exemplo: WSTG-v42-INFO-02 seria entendido como significando especificamente o segundo teste de coleta de informações da versão 4.2.

Se os identificadores forem usados sem incluir o elemento <version> então deve-se presumir que eles se referem ao conteúdo mais recente do Web Security Testing Guide. Obviamente, à medida que o guia cresce e muda, isso se torna problemático, e é por isso que os escritores ou desenvolvedores devem incluir o elemento version.

Vinculando

A vinculação aos cenários do Guia de testes de segurança da Web deve ser feita usando links com versão não estável ou mais recente , o que definitivamente mudará com o tempo. Entretanto, é intenção da equipe do projeto que os links versionados não sejam alterados. Por exemplo: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server.html . Nota: o elemento v42 refere-se à versão 4.2.

Feedback e comentários

Tal como acontece com todos os projetos OWASP, agradecemos comentários e feedback. Gostamos especialmente de saber que nosso trabalho está sendo utilizado e que é eficaz e preciso.

Princípios de Teste Existem alguns

equívocos comuns ao desenvolver uma metodologia de teste para encontrar bugs de segurança em software.
Este capítulo aborda alguns dos princípios básicos que os profissionais devem levar em consideração ao realizar testes de segurança em software.

Não há bala de prata

Embora seja tentador pensar que um scanner de segurança ou firewall de aplicativo fornecerá muitas defesas contra ataques ou identificará uma infinidade de problemas, na realidade não existe solução mágica para o problema de software inseguro. O software de avaliação de segurança de aplicativos, embora útil como uma primeira passagem para encontrar resultados mais fáceis de alcançar, geralmente é imaturo e ineficaz na avaliação aprofundada ou no fornecimento de cobertura de teste adequada. Lembre-se de que segurança é um processo e não um produto.

Pense estrategicamente, não taticamente Os

profissionais de segurança perceberam a falácia do modelo patch-and-penetrate que era difundido na segurança da informação durante a década de 1990. O modelo patch-and-penetrate envolve a correção de um bug relatado, mas sem a investigação adequada da causa raiz. Este modelo está normalmente associado à janela de vulnerabilidade, também designada por janela de exposição, apresentada na figura abaixo. A evolução das vulnerabilidades em softwares comuns utilizados em todo o mundo mostrou a ineficácia deste modelo. Para obter mais informações sobre janelas de exposição, consulte Schneier em Segurança.

Machine Translated by Google

Estudos de vulnerabilidade, como o Relatório de ameaças à segurança da Internet da Symantec mostraram que, com o tempo de reação dos invasores em todo o mundo, a janela típica de vulnerabilidade não fornece tempo suficiente para a instalação do patch, uma vez que o tempo entre a descoberta de uma vulnerabilidade e o desenvolvimento e lançamento de um ataque automatizado contra ela está diminuindo a cada ano.

Existem várias suposições incorretas no modelo patch-and-penetrate. Muitos usuários acreditam que os patches interferem nas operações normais ou podem interromper os aplicativos existentes. Também é incorreto presumir que todos os usuários estão cientes dos patches recém-lançados. Conseqüentemente, nem todos os usuários de um produto aplicarão patches, seja porque acham que os patches podem interferir no funcionamento do software ou porque não têm conhecimento sobre a existência do patch.

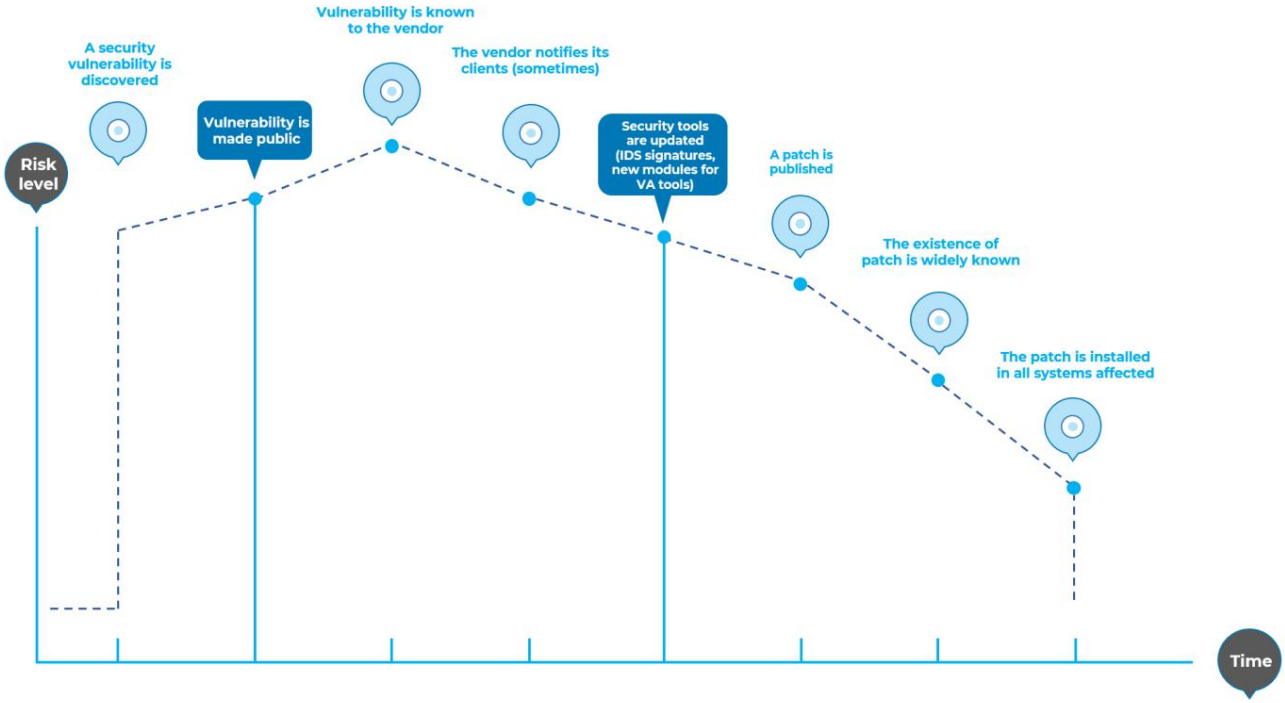

Figura 2-2: Janela de Vulnerabilidade

É essencial incorporar segurança ao Ciclo de Vida de Desenvolvimento de Software (SDLC) para evitar problemas de segurança recorrentes em um aplicativo. Os desenvolvedores podem incorporar segurança ao SDLC desenvolvendo padrões, políticas e diretrizes que se ajustem e funcionem dentro da metodologia de desenvolvimento. A modelagem de ameaças e outras técnicas devem ser usadas para ajudar a atribuir recursos apropriados às partes de um sistema que correm maior risco.

O SDLC é rei

O SDLC é um processo bem conhecido dos desenvolvedores. Ao integrar a segurança em cada fase do SDLC, permite uma abordagem holística à segurança das aplicações que aproveita os procedimentos já em vigor na organização. Esteja ciente de que, embora os nomes das diversas fases possam mudar dependendo do modelo SDLC usado por uma organização, cada fase conceitual do arquétipo SDLC será usada para desenvolver o aplicativo (ou seja, definir, projetar, desenvolver, implantar, manter). Cada fase tem considerações de segurança que devem tornar-se parte do processo existente, para garantir um programa de segurança abrangente e com boa relação custo-benefício.

Existem várias estruturas SDLC seguras que fornecem conselhos descritivos e prescritivos. Se uma pessoa segue aconselhamento descritivo ou prescritivo depende da maturidade do processo SDLC. Essencialmente, o aconselhamento prescritivo mostra como o SDLC seguro deve funcionar, e o aconselhamento descritivo mostra como ele é usado no mundo real. Ambos têm o seu lugar. Por exemplo, se você não sabe por onde começar, uma estrutura prescritiva pode fornecer um menu de possíveis controles de segurança que podem ser aplicados no SDLC. O aconselhamento descritivo pode então ajudar a orientar o processo de decisão, apresentando o que funcionou bem para outras organizações. SDLCs seguros descritivos incluem BSIMM; e os SDLCs seguros prescritivos incluem o Open Software Assurance Maturity Model (OpenSAMM) da OWASP e a ISO/IEC 27034 Partes 1-7, todas publicadas (exceto a parte 4).

Teste cedo e teste frequentemente

Machine Translated by Google

Quando um bug é detectado precocemente no SDLC, ele pode ser resolvido mais rapidamente e com menor custo. Nesse aspecto, um bug de segurança não é diferente de um bug funcional ou baseado em desempenho. Um passo fundamental para tornar isso possível é educar as equipes de desenvolvimento e controle de qualidade sobre problemas comuns de segurança e as formas de detectá-los e evitá-los. Embora novas bibliotecas, ferramentas ou linguagens possam ajudar a projetar programas com menos bugs de segurança, novas ameaças surgem constantemente e os desenvolvedores devem estar cientes das ameaças que afetam o software que estão desenvolvendo. A educação em testes de segurança também ajuda os desenvolvedores a adquirirem a mentalidade apropriada para testar um aplicativo da perspectiva de um invasor. Isto permite que cada organização considere as questões de segurança como parte das suas responsabilidades existentes.

Automação de testes

Em metodologias de desenvolvimento modernas, como (mas não limitadas a): ágil, devops/devsecops ou desenvolvimento rápido de aplicativos (RAD), deve-se considerar a integração de testes de segurança em fluxos de trabalho de integração contínua/implantação contínua (CI/CD), a fim de manter informações/análises básicas de segurança e identificar pontos fracos do tipo “mais fáceis de alcançar”. Isso pode ser feito aproveitando testes dinâmicos de segurança de aplicativos (DAST), testes estáticos de segurança de aplicativos (SAST) e análise de composição de software (SCA) ou ferramentas de rastreamento de dependência durante fluxos de trabalho de lançamento automatizados padrão ou regularmente programados.

Entenda o escopo da segurança

É importante saber quanta segurança um determinado projeto exigirá. Os bens a serem protegidos devem receber uma classificação que indique como devem ser tratados (por exemplo, confidencial, secreto, ultrassecreto). As discussões devem ocorrer com o conselho jurídico para garantir que quaisquer requisitos de segurança específicos serão atendidos. Nos EUA, os requisitos podem vir de regulamentações federais, como a Lei Gramm-Leach-Bliley, ou de leis estaduais, como o California SB-1386.
Para organizações sediadas em países da UE, podem ser aplicadas regulamentações específicas do país e diretivas da UE. Por exemplo, a Directiva 96/46/CE4 e Regulamento (UE) 2016/679 (Regulamento Geral de Proteção de Dados) tornar obrigatório o tratamento dos dados pessoais nas aplicações com o devido cuidado, qualquer que seja a aplicação.

Desenvolva a mentalidade certa

Testar com sucesso um aplicativo em busca de vulnerabilidades de segurança exige pensar “fora da caixa”. Os casos de uso normais testarão o comportamento normal do aplicativo quando um usuário o estiver usando da maneira esperada. Bons testes de segurança exigem ir além do esperado e pensar como um invasor que está tentando invadir a aplicação.
O pensamento criativo pode ajudar a determinar quais dados inesperados podem causar falhas em um aplicativo de maneira insegura.
Também pode ajudar a encontrar suposições feitas por desenvolvedores da Web que nem sempre são verdadeiras e como essas suposições podem ser subvertidas. Uma razão pela qual as ferramentas automatizadas fazem um mau trabalho nos testes de vulnerabilidades é que as ferramentas automatizadas não pensam de forma criativa. O pensamento criativo deve ser feito caso a caso, pois a maioria das aplicações web estão sendo desenvolvidas de forma única (mesmo quando se utilizam frameworks comuns).

Entenda o assunto

Uma das primeiras iniciativas importantes em qualquer bom programa de segurança deve ser exigir documentação precisa da aplicação. A arquitetura, os diagramas de fluxo de dados, os casos de uso, etc. devem ser registrados em documentos formais e disponibilizados para revisão. As especificações técnicas e os documentos de aplicação devem incluir informações que listem não apenas os casos de uso desejados, mas também quaisquer casos de uso especificamente não permitidos. Finalmente, é bom ter pelo menos uma infra-estrutura de segurança básica que permita a monitorização e tendências de ataques contra as aplicações e a rede de uma organização (por exemplo, sistemas de detecção de intrusões).

Use as ferramentas certas

Embora já tenhamos afirmado que não existe uma ferramenta mágica, as ferramentas desempenham um papel crítico no programa de segurança geral. Há uma variedade de ferramentas comerciais e de código aberto que podem automatizar muitas tarefas rotineiras de segurança. Essas ferramentas podem simplificar e acelerar o processo de segurança, auxiliando o pessoal de segurança em suas tarefas. Porém, é importante entender exatamente o que essas ferramentas podem ou não fazer para que não sejam vendidas em excesso ou usadas incorretamente.

O diabo está nos detalhes

É fundamental não realizar uma análise superficial de segurança de um aplicativo e considerá-la completa. Isto irá incutir uma falsa sensação de confiança que pode ser tão perigosa quanto não ter feito uma revisão de segurança. É vital analisar cuidadosamente as conclusões e eliminar quaisquer falsos positivos que possam permanecer no relatório. Relatar uma descoberta de segurança incorreta muitas vezes pode prejudicar a mensagem válida do restante de um relatório de segurança. Deve-se ter cuidado para verificar

Machine Translated by Google

que todas as seções possíveis da lógica do aplicativo foram testadas e que todos os cenários de casos de uso foram explorados em busca de possíveis vulnerabilidades.

Use o código-fonte quando disponível

Embora os resultados dos testes de penetração de caixa preta possam ser impressionantes e úteis para demonstrar como as vulnerabilidades são expostas em um ambiente de produção, eles não são a maneira mais eficaz ou eficiente de proteger um aplicativo. É difícil para o teste dinâmico testar toda a base de código, principalmente se existirem muitas instruções condicionais aninhadas. Se o código-fonte do aplicativo estiver disponível, ele deverá ser fornecido à equipe de segurança para auxiliá-los durante a revisão. É possível descobrir vulnerabilidades na origem do aplicativo que seriam perdidas durante um envolvimento de caixa preta.

Desenvolva métricas

Uma parte importante de um bom programa de segurança é a capacidade de determinar se as coisas estão melhorando. É importante acompanhar os resultados dos testes e desenvolver métricas que revelem as tendências de segurança de aplicativos dentro da organização.

Boas métricas mostrarão:

- Se for necessária mais educação e formação;
- Se houver um mecanismo de segurança específico que não seja claramente compreendido pela equipe de desenvolvimento;
- Se o número total de problemas relacionados à segurança encontrados estiver diminuindo.

Métricas consistentes que podem ser geradas de forma automatizada a partir do código-fonte disponível também ajudarão a organização a avaliar a eficácia dos mecanismos introduzidos para reduzir bugs de segurança no desenvolvimento de software. As métricas não são facilmente desenvolvidas, portanto, usar um padrão como o fornecido pelo IEEE é um bom ponto de partida.

Documente os resultados do teste

Para concluir o processo de teste, é importante produzir um registro formal de quais ações de teste foram realizadas, por quem, quando foram realizadas e detalhes dos resultados do teste. É aconselhável chegar a um acordo sobre um formato aceitável para o relatório que seja útil para todas as partes interessadas, que pode incluir desenvolvedores, gerenciamento de projetos, proprietários de empresas, departamento de TI, auditoria e conformidade.

O relatório deve identificar claramente ao proprietário da empresa onde existem riscos materiais e fazê-lo de forma suficiente para obter o seu apoio para ações de mitigação subsequentes. O relatório também deve ser claro para o desenvolvedor ao identificar a função exata que é afetada pela vulnerabilidade e recomendações associadas para resolver problemas em uma linguagem que o desenvolvedor entenda. O relatório também deve permitir que outro testador de segurança reproduza os resultados. Escrever o relatório não deve ser excessivamente oneroso para o testador de segurança. Os testadores de segurança geralmente não são conhecidos por suas habilidades de redação criativa, e chegar a um acordo sobre um relatório complexo pode levar a casos em que os resultados dos testes não sejam devidamente documentados. Usar um modelo de relatório de teste de segurança pode economizar tempo e garantir que os resultados sejam documentados de maneira precisa e consistente e em um formato adequado ao público.

Técnicas de teste explicadas Esta seção apresenta uma

visão geral de alto nível de diversas técnicas de teste que podem ser empregadas ao construir um programa de teste. Não apresenta metodologias específicas para essas técnicas, pois essas informações são abordadas no Capítulo 3.
Esta seção foi incluída para contextualizar a estrutura apresentada no próximo capítulo e para destacar as vantagens ou desvantagens de algumas das técnicas que devem ser consideradas. Em particular, cobriremos:

- Inspeções e revisões manuais
- Modelagem de ameaças
- Revisão de código
- Teste de penetração

Inspeções e revisões manuais

Machine Translated by Google

Visão geral

As inspeções manuais são revisões humanas que normalmente testam as implicações de segurança de pessoas, políticas e processos. As inspeções manuais também podem incluir a inspeção de decisões tecnológicas, como projetos arquitetônicos. Geralmente são conduzidos por meio da análise de documentação ou da realização de entrevistas com os projetistas ou proprietários do sistema.

Embora o conceito de inspeções manuais e revisões humanas seja simples, elas podem estar entre as técnicas mais poderosas e eficazes disponíveis. Ao perguntar a alguém como algo funciona e por que foi implementado de uma maneira específica, o testador pode determinar rapidamente se alguma preocupação de segurança provavelmente será evidente. Inspeções e revisões manuais são uma das poucas maneiras de testar o próprio processo do ciclo de vida de desenvolvimento de software e de garantir que haja uma política ou conjunto de habilidades adequado em vigor.

Tal como acontece com muitas coisas na vida, ao realizar inspeções e revisões manuais, recomenda-se que seja adotado um modelo de confiança, mas verificação. Nem tudo o que o testador mostra ou conta será preciso. As revisões manuais são particularmente boas para testar se as pessoas compreendem o processo de segurança, se foram informadas sobre as políticas e se possuem as habilidades adequadas para projetar ou implementar aplicações seguras.

Outras atividades, incluindo a revisão manual da documentação, políticas de codificação segura, requisitos de segurança e projetos arquitetônicos, devem ser realizadas por meio de inspeções manuais.

Vantagens

- Não requer tecnologia de suporte
- Pode ser aplicado em diversas situações
- Flexível
- Promove o trabalho em equipe
- No início do SDLC

Desvantagens

- Pode ser demorado
- Material de apoio nem sempre disponível
- Requer pensamento e habilidade humanos significativos para ser eficaz

Modelagem de ameaças

Visão geral

A modelagem de ameaças tornou-se uma técnica popular para ajudar os projetistas de sistemas a pensar sobre as ameaças à segurança que seus sistemas e aplicativos podem enfrentar. Portanto, a modelagem de ameaças pode ser vista como uma avaliação de risco para aplicações. Ele permite que o projetista desenvolva estratégias de mitigação para vulnerabilidades potenciais e os ajuda a concentrar seus recursos e atenção inevitavelmente limitados nas partes do sistema que mais necessitam deles. Recomenda-se que todas as aplicações tenham um modelo de ameaça desenvolvido e documentado. Os modelos de ameaças devem ser criados o mais cedo possível no SDLC e devem ser revistos à medida que a aplicação evolui e o desenvolvimento avança.

Para desenvolver um modelo de ameaça, recomendamos adotar uma abordagem simples que siga o NIST 800-30 padrão para avaliação de risco. Esta abordagem envolve:

- Decompondo o aplicativo – use um processo de inspeção manual para entender como o aplicativo funciona, seus ativos, funcionalidade e conectividade.
- Definição e classificação dos ativos – classifique os ativos em ativos tangíveis e intangíveis e classifique-os de acordo com a importância do negócio.
- Explorar vulnerabilidades potenciais – sejam técnicas, operacionais ou gerenciais.
- Explorar ameaças potenciais – desenvolva uma visão realista de vetores de ataque potenciais a partir da perspectiva de um invasor usando cenários de ameaças ou árvores de ataque.
- Criar estratégias de mitigação – desenvolver controlos de mitigação para cada uma das ameaças consideradas realistas.

Machine Translated by Google

A saída de um modelo de ameaça em si pode variar, mas normalmente é uma coleção de listas e diagramas. Vários projetos de código aberto e produtos comerciais suportam metodologias de modelagem de ameaças de aplicativos que podem ser usadas como referência para testar aplicativos em busca de possíveis falhas de segurança no design do aplicativo. Não existe uma maneira certa ou errada de desenvolver modelos de ameaças e realizar avaliações de risco de informações em aplicativos.

Vantagens

- Visão prática do invasor sobre o sistema
- Flexível
- No início do SDLC

Desvantagens

- Bons modelos de ameaças não significam automaticamente um bom software

Revisão do código-fonte

Visão geral

A revisão do código-fonte é o processo de verificação manual do código-fonte de um aplicativo da web em busca de problemas de segurança. Muitas vulnerabilidades graves de segurança não podem ser detectadas com qualquer outra forma de análise ou teste. Como diz o ditado popular “se você quer saber o que realmente está acontecendo, vá direto à fonte”. Quase todos os especialistas em segurança concordam que não há substituto para realmente olhar o código. Todas as informações para identificar problemas de segurança estão no código, em algum lugar. Ao contrário do teste de software fechado, como sistemas operacionais, ao testar aplicativos da Web (especialmente se tiverem sido desenvolvidos internamente), o código-fonte deve ser disponibilizado para fins de teste.

Muitos problemas de segurança não intencionais, mas significativos, são extremamente difíceis de descobrir com outras formas de análise ou teste, como testes de penetração. Isso torna a análise do código-fonte a técnica preferida para testes técnicos. Com o código-fonte, um testador pode determinar com precisão o que está acontecendo (ou deveria estar acontecendo) e eliminar as suposições dos testes de caixa preta.

Exemplos de problemas que são particularmente propícios a serem encontrados através de revisões de código-fonte incluem problemas de simultaneidade, lógica de negócios falha, problemas de controle de acesso e fraquezas criptográficas, bem como backdoors, cavalos de Tróia, ovos de Páscoa, bombas-relógio, bombas lógicas e outras formas de Código malicioso. Esses problemas muitas vezes se manifestam como as vulnerabilidades mais prejudiciais em aplicações web. A análise do código-fonte também pode ser extremamente eficiente para encontrar problemas de implementação, como locais onde a validação de entrada não foi executada ou onde procedimentos de controle de falha aberta podem estar presentes. Os procedimentos operacionais também precisam ser revisados, uma vez que o código-fonte a ser implantado pode não ser o mesmo que está sendo analisado aqui. Discurso do Prêmio Turing de Ken Thompson descreve uma possível manifestação deste problema.

Vantagens

- Completude e eficácia
- Precisão
- Rápido (para revisores competentes)

Desvantagens

- Requer desenvolvedores altamente qualificados e conscientes da segurança
- Pode perder problemas em bibliotecas compiladas
- Não é possível detectar erros de tempo de execução facilmente
- O código-fonte realmente implantado pode ser diferente daquele que está sendo analisado

Para obter mais informações sobre revisão de código, consulte o projeto de revisão de código OWASP.

Teste de penetração

Visão geral

Machine Translated by Google

O teste de penetração tem sido uma técnica comum usada para testar a segurança da rede há décadas. Também é comumente conhecido como teste de caixa preta ou hacking ético. O teste de penetração é essencialmente a “arte” de testar um sistema ou aplicativo remotamente para encontrar vulnerabilidades de segurança, sem conhecer o funcionamento interno do próprio alvo. Normalmente, a equipe de teste de penetração consegue acessar um aplicativo como se fossem usuários. O testador age como um invasor e tenta encontrar e explorar vulnerabilidades. Em muitos casos, o testador receberá uma ou mais contas válidas no sistema.

Embora os testes de penetração tenham provado ser eficazes na segurança de redes, a técnica não se traduz naturalmente em aplicações. Quando os testes de penetração são realizados em redes e sistemas operacionais, a maior parte do trabalho envolvido consiste em encontrar e depois explorar vulnerabilidades conhecidas em tecnologias específicas. Como os aplicativos da web são quase exclusivamente personalizados, os testes de penetração na área de aplicativos da web se assemelham mais à pesquisa pura. Algumas ferramentas automatizadas de testes de penetração foram desenvolvidas, mas considerando a natureza personalizada das aplicações web, a sua eficácia por si só pode ser fraca.

Muitas pessoas usam testes de penetração de aplicativos da web como principal técnica de teste de segurança. Embora certamente tenha o seu lugar em um programa de testes, não acreditamos que deva ser considerada a principal ou única técnica de testes. Como Gary McGraw escreveu em Software Penetration Testing, “Na prática, um teste de penetração só pode identificar uma pequena amostra representativa de todos os possíveis riscos de segurança num sistema.” No entanto, testes de penetração focados (isto é, testes que tentam explorar vulnerabilidades conhecidas detectadas em revisões anteriores) podem ser úteis para detectar se algumas vulnerabilidades específicas estão realmente corrigidas no código-fonte implantado.

Vantagens

- Pode ser rápido (e, portanto, barato)
- Requer um conjunto de habilidades relativamente menor do que a revisão do código-fonte
- Testa o código que está realmente sendo exposto

Desvantagens

- Tarde demais no SDLC
- Apenas testes de impacto frontal

A Necessidade de uma Abordagem Equilibrada

Com tantas técnicas e abordagens para testar a segurança de aplicações web, pode ser difícil entender quais técnicas usar ou quando usá-las. A experiência mostra que não há resposta certa ou errada para a questão de quais técnicas exatamente devem ser usadas para construir uma estrutura de teste. Na verdade, todas as técnicas devem ser utilizadas para testar todas as áreas que precisam ser testadas.

Embora esteja claro que não existe uma técnica única que possa ser executada para cobrir eficazmente todos os testes de segurança e garantir que todos os problemas foram resolvidos, muitas empresas adotam apenas uma abordagem. A única abordagem usada tem sido historicamente o teste de penetração. Os testes de penetração, embora úteis, não podem resolver com eficácia muitos dos problemas que precisam ser testados. É simplesmente “tarde demais” no SDLC.

A abordagem correta é uma abordagem equilibrada que inclui diversas técnicas, desde revisões manuais até testes técnicos, até testes integrados de CI/CD. Uma abordagem equilibrada deve abranger testes em todas as fases do SDLC. Esta abordagem aproveita as técnicas mais adequadas disponíveis, dependendo da fase atual do SDLC.

É claro que há momentos e circunstâncias em que apenas uma técnica é possível. Por exemplo, considere um teste de uma aplicação web que já foi criada, mas onde a parte do teste não tem acesso ao código-fonte. Neste caso, o teste de penetração é claramente melhor do que nenhum teste. No entanto, as partes envolvidas no teste devem ser encorajadas a desafiar suposições, tais como não ter acesso ao código-fonte, e a explorar a possibilidade de testes mais completos.

Uma abordagem equilibrada varia dependendo de muitos fatores, como a maturidade do processo de testes e a cultura corporativa. Recomenda-se que uma estrutura de teste balanceada seja semelhante às representações mostradas na Figura 3 e na Figura 4. A figura a seguir mostra uma representação proporcional típica sobreposta ao SLDC. Em

Machine Translated by Google

de acordo com a investigação e a experiência, é essencial que as empresas coloquem maior ênfase nas fases iniciais de desenvolvimento.

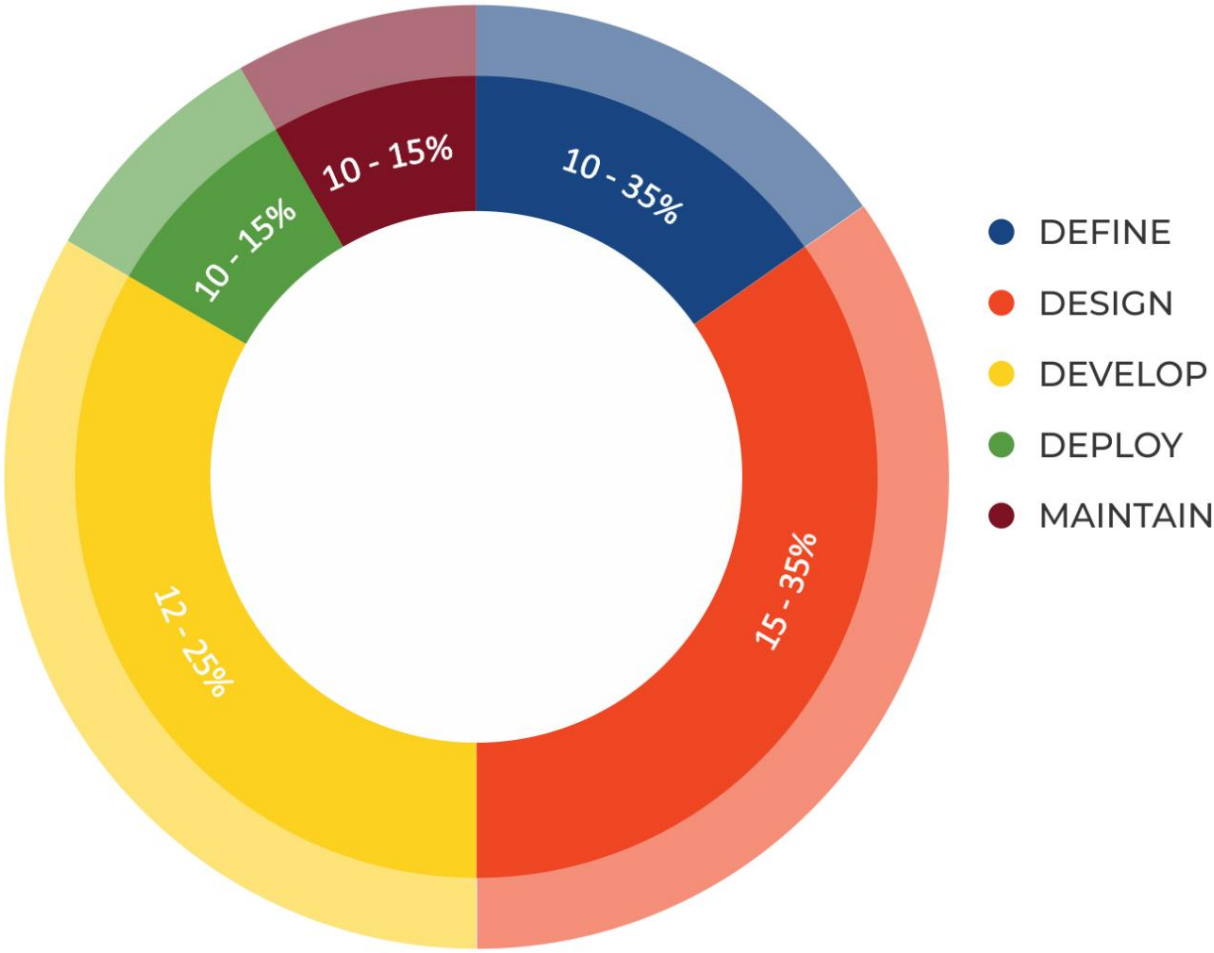

Figura 2-3: Proporção de Esforço de Teste em SDLC

A figura a seguir mostra uma representação proporcional típica sobreposta às técnicas de teste.

Machine Translated by Google

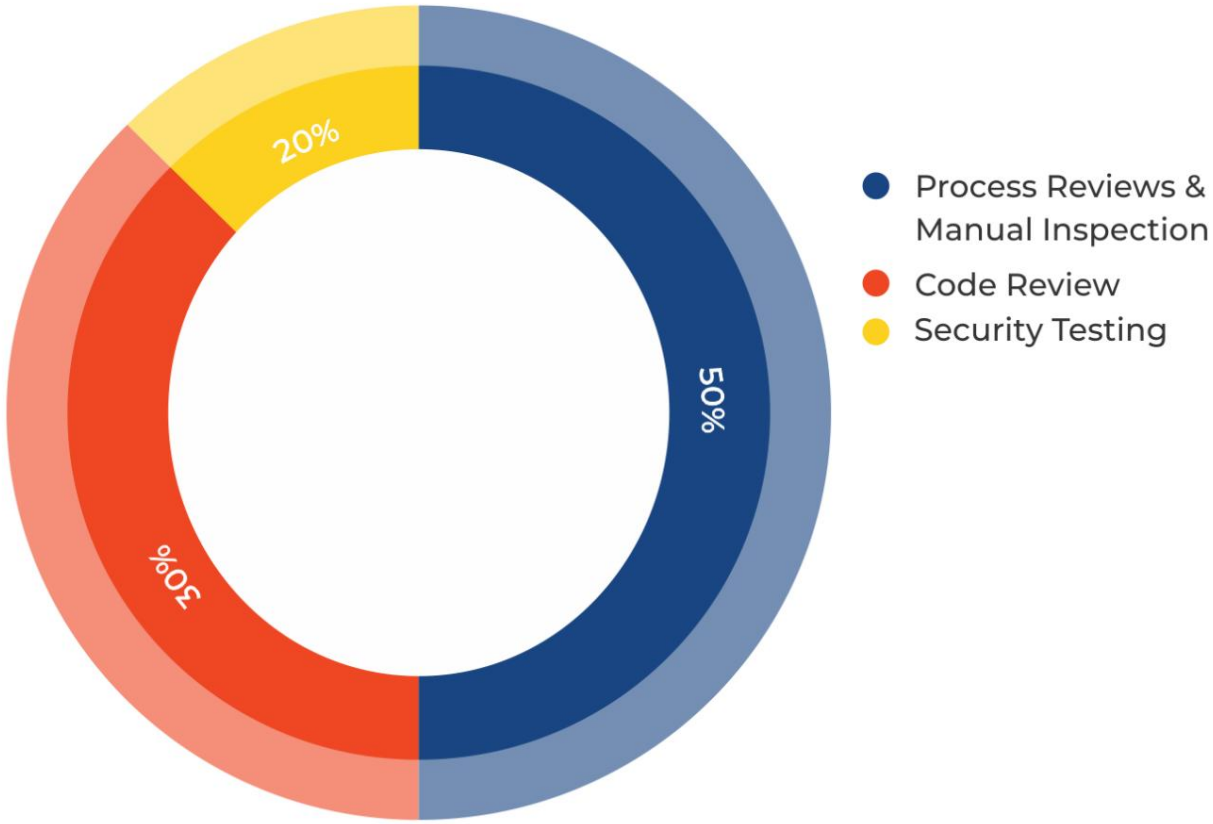

Figura 2-4: Proporção de esforço de teste de acordo com a técnica de teste

Uma observação sobre scanners de aplicativos da Web

Muitas organizações começaram a usar scanners automatizados de aplicativos da web. Embora tenham, sem dúvida, um lugar num programa de testes, algumas questões fundamentais precisam ser destacadas sobre por que se acredita que a automação de testes de caixa preta não é (nem nunca será) completamente eficaz. No entanto, destacar estas questões não deve desencorajar o uso de scanners de aplicações web. Em vez disso, o objetivo é garantir que as limitações sejam compreendidas e que as estruturas de teste sejam planejadas de forma adequada.

É útil compreender a eficácia e as limitações das ferramentas automatizadas de detecção de vulnerabilidades. Para esse fim, o OWASP Benchmark Project é um conjunto de testes projetado para avaliar a velocidade, cobertura e precisão de ferramentas e serviços automatizados de detecção de vulnerabilidades de software. O benchmarking pode ajudar a testar as capacidades destas ferramentas automatizadas e ajudar a tornar explícita a sua utilidade.

Os exemplos a seguir mostram por que os testes automatizados de caixa preta podem não ser eficazes.

Exemplo 1: Parâmetros Mágicos

Imagine um aplicativo da web simples que aceita um par nome-valor “mágico” e depois o valor. Para simplificar, a solicitação GET pode ser: http://www.host/application?magic=value

Para simplificar ainda mais o exemplo, os valores neste caso só podem ser caracteres ASCII a – z (maiúsculas ou minúsculas) e números inteiros 0 – 9.

Os designers deste aplicativo criaram um backdoor administrativo durante os testes, mas o ofuscaram para evitar que um observador casual o descobrisse. Ao enviar o valor sf8g7sfjdsurtsdieerwqredsgnfg8d (30 caracteres), o usuário será então logado e será apresentada uma tela administrativa com total controle da aplicação. A solicitação HTTP agora é: http://www.host/application?magic=sf8g7sfjdsurtsdieerwqredsgnfg8d

Dado que todos os outros parâmetros eram campos simples de dois e três caracteres, não é possível começar a adivinhar combinações com aproximadamente 28 caracteres. Um scanner de aplicativos da web precisará aplicar força bruta (ou adivinhar) em todo o

Machine Translated by Google

espaço-chave de 30 caracteres. Isso representa até 30^28 permutações, ou trilhões de solicitações HTTP. Isso é um elétron em um palheiro digital.

O código para este exemplo de verificação de parâmetro mágico pode ser semelhante ao seguinte:

```
public void doPost( solicitação HttpServletRequest, resposta HttpServletResponse) {
    String mágica = "sf8g7sfjdsurtsdieerwqredsgnfg8d"; boolean
    admin = magic.equals( request.getParameter("magic")); if (admin)
    doAdmin(solicitação, resposta); senão ... //
    processamento normal
}
```

Ao observar o código, a vulnerabilidade praticamente salta da página como um problema potencial.

Exemplo 2: Criptografia incorreta A criptografia

é amplamente utilizada em aplicações web. Imagine que um desenvolvedor decidiu escrever um algoritmo de criptografia simples para conectar um usuário do site A ao site B automaticamente. Em sua sabedoria, o desenvolvedor decide que se um usuário estiver logado no site A, ele irá gerar uma chave usando uma função hash MD5 que compreende: Hash { nome de usuário: data
}

Quando um usuário é passado para o site B, ele enviará a chave na string de consulta para o site B em um redirecionamento HTTP. O site B calcula o hash de forma independente e o compara com o hash transmitido na solicitação. Se corresponderem, o site B conecta o usuário como o usuário que afirma ser.

À medida que o esquema é explicado, as inadequações podem ser resolvidas. Qualquer pessoa que descubra o esquema (ou saiba como ele funciona, ou baixe as informações do Bugtraq) pode fazer login como qualquer usuário. A inspeção manual, como uma revisão ou inspeção de código, teria descoberto esse problema de segurança rapidamente. Um scanner de aplicação web de caixa preta não teria descoberto a vulnerabilidade. Teria visto um hash de 128 bits que mudava com cada usuário e, pela natureza das funções hash, não mudava de forma previsível.

Uma observação sobre ferramentas estáticas de revisão de código-fonte

Muitas organizações começaram a usar scanners de código-fonte estáticos. Embora tenham, sem dúvida, um lugar num programa de testes abrangente, é necessário destacar algumas questões fundamentais sobre por que esta abordagem não é eficaz quando usada isoladamente. A análise estática do código-fonte por si só não consegue identificar problemas devido a falhas no design, uma vez que não consegue compreender o contexto no qual o código é construído. As ferramentas de análise de código-fonte são úteis para determinar problemas de segurança devido a erros de codificação; no entanto, é necessário um esforço manual significativo para validar as descobertas.

Derivando Requisitos de Teste de Segurança

Para ter um programa de testes bem-sucedido, é preciso saber quais são os objetivos dos testes. Esses objetivos são especificados pelos requisitos de segurança. Esta seção discute detalhadamente como documentar requisitos para testes de segurança, derivando-os de padrões e regulamentos aplicáveis, de requisitos de aplicação positivos (especificando o que a aplicação deve fazer) e de requisitos de aplicação negativos (especificando o que a aplicação não deve fazer). . Ele também discute como os requisitos de segurança orientam efetivamente os testes de segurança durante o SDLC e como os dados dos testes de segurança podem ser usados para gerenciar com eficácia os riscos de segurança de software.

Objetivos do teste Um dos

objetivos dos testes de segurança é validar se os controles de segurança funcionam conforme esperado. Isto é documentado através de requisitos de segurança que descrevem a funcionalidade do controle de segurança. Em alto nível, isso significa comprovar a confidencialidade, integridade e disponibilidade dos dados, bem como do serviço. O outro objetivo é validar se os controles de segurança são implementados com poucas ou nenhuma vulnerabilidade. Estas são vulnerabilidades comuns, como o OWASP Top Ten, bem como vulnerabilidades que foram previamente identificadas com avaliações de segurança durante o SDLC, como modelagem de ameaças, análise de código-fonte e teste de penetração.

Documentação de requisitos de segurança

Machine Translated by Google

A primeira etapa na documentação dos requisitos de segurança é compreender os requisitos de negócios . Um documento de requisitos de negócios pode fornecer informações iniciais de alto nível sobre a funcionalidade esperada do aplicativo. Por exemplo, o objectivo principal de uma aplicação pode ser fornecer serviços financeiros a clientes ou permitir a compra de bens a partir de um catálogo on-line. Uma seção de segurança dos requisitos de negócios deve destacar a necessidade de proteger os dados do cliente, bem como de cumprir a documentação de segurança aplicável, como regulamentos, padrões e políticas.

Uma lista de verificação geral dos regulamentos, padrões e políticas aplicáveis é uma boa análise preliminar de conformidade de segurança para aplicativos da web. Por exemplo, as regulamentações de conformidade podem ser identificadas verificando informações sobre o setor empresarial e o país ou estado onde o aplicativo irá operar. Algumas destas diretrizes e regulamentos de conformidade podem traduzir-se em requisitos técnicos específicos para controlos de segurança. Por exemplo, no caso de aplicações financeiras, a conformidade com a Ferramenta e Documentação de Avaliação de Segurança Cibernética do Conselho Federal de Exame de Instituições Financeiras (FFIEC) exige que as instituições financeiras implementem aplicações que mitiguem riscos de autenticação fraca com controles de segurança multicamadas e autenticação multifatorial.

Os padrões de segurança aplicáveis da indústria também devem ser incluídos na lista de verificação de requisitos gerais de segurança. Por exemplo, no caso de aplicativos que lidam com dados de cartão de crédito de clientes, a conformidade com o PCI Security Standards Council Data Security Standard (DSS) proíbe o armazenamento de PINs e dados CVV2 e exige que o comerciante proteja os dados da tarja magnética no armazenamento e na transmissão com criptografia e exibição por mascaramento. Esses requisitos de segurança do PCI DSS poderiam ser validados por meio da análise do código-fonte.

Outra seção da lista de verificação precisa impor requisitos gerais para conformidade com os padrões e políticas de segurança da informação da organização. Da perspectiva dos requisitos funcionais, os requisitos para o controle de segurança precisam ser mapeados para uma seção específica dos padrões de segurança da informação. Um exemplo de tal requisito pode ser: “uma complexidade de senha de dez caracteres alfanuméricos deve ser imposta pelos controles de autenticação usados pelo aplicativo”. Quando os requisitos de segurança são mapeados para regras de conformidade, um teste de segurança pode validar a exposição dos riscos de conformidade. Se forem encontradas violações dos padrões e políticas de segurança da informação, estas resultarão num risco que pode ser documentado e que a empresa terá de gerir ou resolver. Como esses requisitos de conformidade de segurança são aplicáveis, eles precisam ser bem documentados e validados com testes de segurança.

Validação de Requisitos de Segurança

Do ponto de vista da funcionalidade, a validação dos requisitos de segurança é o objetivo principal dos testes de segurança. Do ponto de vista da gestão de riscos, a validação dos requisitos de segurança é o objetivo das avaliações de segurança da informação. Em alto nível, o principal objetivo das avaliações de segurança da informação é a identificação de lacunas nos controles de segurança, como a falta de autenticação básica, autorização ou controles de criptografia. Examinado mais detalhadamente, o objetivo da avaliação de segurança é a análise de risco, como a identificação de potenciais fraquezas nos controles de segurança que garantem a confidencialidade, integridade e disponibilidade dos dados. Por exemplo, quando a aplicação trata de informações de identificação pessoal (PII) e dados sensíveis, o requisito de segurança a ser validado é o cumprimento da política de segurança da informação da empresa que exige a criptografia de tais dados em trânsito e no armazenamento. Supondo que a criptografia seja usada para proteger os dados, os algoritmos de criptografia e os comprimentos das chaves precisam estar em conformidade com os padrões de criptografia da organização. Isso pode exigir que apenas determinados algoritmos e comprimentos de chave sejam usados. Por exemplo, um requisito de segurança que pode ser testado é verificar se apenas as cifras permitidas são usadas (por exemplo, SHA-256, RSA, AES) com comprimentos de chave mínimos permitidos (por exemplo, mais de 128 bits para simétricos e mais de 1024 para assimétricos). criptografia).

Do ponto de vista da avaliação de segurança, os requisitos de segurança podem ser validados em diferentes fases do SDLC usando diferentes artefatos e metodologias de teste. Por exemplo, a modelagem de ameaças concentra-se na identificação de falhas de segurança durante o projeto; análises e revisões seguras de código concentram-se na identificação de problemas de segurança no código-fonte durante o desenvolvimento; e o teste de penetração concentra-se na identificação de vulnerabilidades no aplicativo durante o teste ou validação.

Os problemas de segurança identificados no início do SDLC podem ser documentados em um plano de teste para que possam ser validados posteriormente com testes de segurança. Ao combinar os resultados de diferentes técnicas de teste, é possível obter melhores casos de teste de segurança e aumentar o nível de garantia dos requisitos de segurança. Por exemplo, distinguir as vulnerabilidades verdadeiras das inexploráveis é possível quando os resultados dos testes de penetração e da análise do código-fonte são combinados.
Considerando o teste de segurança para uma vulnerabilidade de injeção de SQL, por exemplo, um teste de caixa preta pode envolver primeiro uma varredura de

Machine Translated by Google

o aplicativo para identificar a vulnerabilidade. A primeira evidência de uma potencial vulnerabilidade de injeção SQL que pode ser validada é a geração de uma exceção SQL. Uma validação adicional da vulnerabilidade SQL pode envolver a injeção manual de vetores de ataque para modificar a gramática da consulta SQL para uma exploração de divulgação de informações. Isso pode envolver muitas análises de tentativa e erro antes que a consulta maliciosa seja executada. Supondo que o testador possua o código-fonte, ele poderá aprender diretamente com a análise do código-fonte como construir o vetor de ataque SQL que explorará a vulnerabilidade com êxito (por exemplo, executar uma consulta maliciosa retornando dados confidenciais a um usuário não autorizado). Isso pode agilizar a validação da vulnerabilidade do SQL.

Taxonomias de ameaças e contramedidas

Uma classificação de ameaça e contramedida , que leva em consideração as causas raízes das vulnerabilidades, é o fator crítico para verificar se os controles de segurança são projetados, codificados e construídos para mitigar o impacto da exposição de tais vulnerabilidades. No caso de aplicações web, a exposição dos controles de segurança a vulnerabilidades comuns, como o OWASP Top Ten, pode ser um bom ponto de partida para derivar requisitos gerais de segurança. A lista de verificação do guia de testes OWASP é um recurso útil para orientar os testadores através de vulnerabilidades específicas e testes de validação.

O foco de uma categorização de ameaças e contramedidas é definir os requisitos de segurança em termos das ameaças e da causa raiz da vulnerabilidade. Uma ameaça pode ser categorizada usando STRIDE, um acrônimo para Spoofing, Adulteração, Repúdio, Divulgação de informações, Negação de serviço e Elevação de privilégio. A causa raiz pode ser categorizada como falha de segurança no design, um bug de segurança na codificação ou um problema devido a uma configuração insegura. Por exemplo, a causa raiz da vulnerabilidade de autenticação fraca pode ser a falta de autenticação mútua quando os dados cruzam um limite de confiança entre as camadas cliente e servidor do aplicativo. Um requisito de segurança que captura a ameaça de não repúdio durante uma revisão do projeto de arquitetura permite a documentação do requisito para a contramedida (por exemplo, autenticação mútua) que pode ser validada posteriormente com testes de segurança.

Uma categorização de ameaças e contramedidas para vulnerabilidades também pode ser usada para documentar requisitos de segurança para codificação segura, como padrões de codificação segura. Um exemplo de erro de codificação comum em controles de autenticação consiste na aplicação de uma função hash para criptografar uma senha, sem aplicar uma semente ao valor. Do ponto de vista da codificação segura, esta é uma vulnerabilidade que afeta a criptografia usada para autenticação com uma causa raiz de vulnerabilidade em um erro de codificação. Como a causa raiz é a codificação insegura, o requisito de segurança pode ser documentado em padrões de codificação segura e validado por meio de revisões seguras de código durante a fase de desenvolvimento do SDLC.

Testes de segurança e análise de riscos

Os requisitos de segurança precisam levar em consideração a gravidade das vulnerabilidades para apoiar uma estratégia de mitigação de riscos . Supondo que a organização mantenha um repositório de vulnerabilidades encontradas em aplicativos (ou seja, uma base de conhecimento de vulnerabilidades), os problemas de segurança podem ser relatados por tipo, problema, mitigação, causa raiz e mapeados para os aplicativos onde são encontrados. Essa base de conhecimento de vulnerabilidade também pode ser usada para estabelecer métricas para analisar a eficácia dos testes de segurança em todo o SDLC.

Por exemplo, considere um problema de validação de entrada, como uma injeção de SQL, que foi identificado por meio de análise de código-fonte e relatado com uma causa raiz de erro de codificação e um tipo de vulnerabilidade de validação de entrada. A exposição de tal vulnerabilidade pode ser avaliada através de um teste de penetração, investigando campos de entrada com vários vetores de ataque de injeção SQL. Este teste pode validar se os caracteres especiais são filtrados antes de atingir o banco de dados e mitigar a vulnerabilidade. Ao combinar os resultados da análise do código-fonte e do teste de penetração, é possível determinar a probabilidade e a exposição da vulnerabilidade e calcular a classificação de risco da vulnerabilidade. Ao reportar classificações de risco de vulnerabilidade nas conclusões (por exemplo, relatório de teste) é possível decidir sobre a estratégia de mitigação. Por exemplo, vulnerabilidades de alto e médio risco podem ser priorizadas para correção, enquanto vulnerabilidades de baixo risco podem ser corrigidas em versões futuras.

Ao considerar os cenários de ameaça de exploração de vulnerabilidades comuns, é possível identificar riscos potenciais para os quais o controle de segurança do aplicativo precisa ser testado. Por exemplo, as dez principais vulnerabilidades do OWASP podem ser mapeadas para ataques como phishing, violações de privacidade, roubo de identidade, comprometimento do sistema, alteração ou destruição de dados, perdas financeiras e perda de reputação. Tais questões devem ser documentadas como parte dos cenários de ameaça. Pensando em termos de ameaças e vulnerabilidades, é possível elaborar uma bateria de testes que simulem tais cenários de ataque. Idealmente, a base de conhecimento de vulnerabilidades da organização pode ser usada para derivar casos de teste orientados a riscos de segurança para validar os cenários de ataque mais prováveis. Por exemplo, se o roubo de identidade for considerado de alto risco, cenários de teste negativos

Machine Translated by Google

devem validar a mitigação dos impactos decorrentes da exploração de vulnerabilidades em autenticação, controles criptográficos, validação de entrada e controles de autorização.

Derivando Requisitos de Teste Funcionais e Não Funcionais

Requisitos de segurança funcional

Do ponto de vista dos requisitos de segurança funcional, os padrões, políticas e regulamentos aplicáveis determinam tanto a necessidade de um tipo de controle de segurança quanto a funcionalidade de controle. Esses requisitos também são chamados de “requisitos positivos”, pois estabelecem a funcionalidade esperada que pode ser validada por meio de testes de segurança.
Exemplos de requisitos positivos são: “a aplicação bloqueará o usuário após seis tentativas malsucedidas de login” ou “as senhas precisam ter no mínimo dez caracteres alfanuméricos”. A validação de requisitos positivos consiste em afirmar a funcionalidade esperada e pode ser testada recriando as condições de teste e executando o teste de acordo com entradas predefinidas. Os resultados são então mostrados como uma condição de falha ou aprovação.

Para validar os requisitos de segurança com testes de segurança, os requisitos de segurança precisam ser orientados por funções. Eles precisam destacar a funcionalidade esperada (o quê) e implicar a implementação (o como). Exemplos de requisitos de design de segurança de alto nível para autenticação podem ser:

- Proteja as credenciais do usuário ou segredos compartilhados em trânsito e armazenamento.
- Mascare quaisquer dados confidenciais em exibição (por exemplo, senhas, contas).
- Bloqueie a conta do usuário após um certo número de tentativas de login malsucedidas.
- Não mostre erros de validação específicos ao usuário como resultado de uma falha no logon.
- Permita apenas senhas alfanuméricas, que incluam caracteres especiais e tenham no mínimo dez caracteres de comprimento, para limitar a superfície de ataque.
- Permita a funcionalidade de alteração de senha apenas para usuários autenticados, validando a senha antiga, a nova senha e a resposta do usuário à pergunta de desafio, para evitar a força bruta de uma senha por meio da alteração de senha.
- O formulário de redefinição de senha deve validar o nome de usuário e o e-mail cadastrado do usuário antes de enviar a senha temporária ao usuário por e-mail. A senha temporária emitida deve ser uma senha de uso único. Um link para a página da web de redefinição de senha será enviado ao usuário. A página web de redefinição de senha deve validar a senha temporária do usuário, a nova senha, bem como a resposta do usuário à pergunta de desafio.

Requisitos de segurança orientados a riscos

Os testes de segurança também devem ser orientados ao risco. Eles precisam validar o aplicativo quanto a comportamentos inesperados ou requisitos negativos.

Exemplos de requisitos negativos são:

- A aplicação não deve permitir que os dados sejam alterados ou destruídos.
- O aplicativo não deve ser comprometido ou utilizado indevidamente para transações financeiras não autorizadas por um mal-intencionado do utilizador.

Os requisitos negativos são mais difíceis de testar porque não há comportamento esperado a ser observado. Procurar um comportamento esperado que atenda aos requisitos acima pode exigir que um analista de ameaças apresente condições, causas e efeitos de entrada imprevisíveis, de forma irrealista. Portanto, os testes de segurança precisam ser orientados pela análise de riscos e pela modelagem de ameaças.
A chave é documentar os cenários de ameaça e a funcionalidade da contramedida como um fator para mitigar uma ameaça.

Por exemplo, no caso de controlos de autenticação, os seguintes requisitos de segurança podem ser documentados do ponto de vista de ameaças e contramedidas:

- Criptografe dados de autenticação em armazenamento e trânsito para reduzir o risco de divulgação de informações e ataques ao protocolo de autenticação.
- Criptografe senhas usando criptografia não reversível, como usar um resumo (por exemplo, HASH) e uma semente para evitar ataques de dicionário.

Machine Translated by Google

- Bloqueie contas após atingir um limite de falha de logon e aplique a complexidade da senha para mitigar o risco de ataques de força bruta a senhas.
- Exibir mensagens de erro genéricas na validação de credenciais para reduzir o risco de coleta de contas ou enumeração.
- Autentique mutuamente cliente e servidor para evitar ataques de não repúdio e Manipulator In the Middle (MiTM).

Ferramentas de modelagem de ameaças, como árvores de ameaças e bibliotecas de ataques, podem ser úteis para derivar cenários de testes negativos. Uma árvore de ameaças assumirá um ataque raiz (por exemplo, o invasor pode ser capaz de ler as mensagens de outros usuários) e identificará diferentes explorações de controles de segurança (por exemplo, falha na validação de dados devido a uma vulnerabilidade de injeção de SQL) e contramedidas necessárias (por exemplo, implementar dados validação e consultas parametrizadas) que poderiam ser validadas para serem eficazes na mitigação de tais ataques.

Derivando requisitos de teste de segurança por meio de casos de uso e uso indevido

Um pré-requisito para descrever a funcionalidade do aplicativo é entender o que o aplicativo deve fazer e como. Isso pode ser feito descrevendo casos de uso. Os casos de uso, na forma gráfica comumente usada na engenharia de software, mostram as interações dos atores e suas relações. Eles ajudam a identificar os atores da aplicação, seus relacionamentos, a sequência de ações pretendida para cada cenário, ações alternativas, requisitos especiais, pré-condições e pós-condições.

Semelhante aos casos de uso, os casos de uso indevido ou abuso descrevem cenários de uso não intencional e malicioso do aplicativo. Esses casos de uso indevido fornecem uma maneira de descrever cenários de como um invasor poderia fazer uso indevido e abusar do aplicativo. Ao percorrer as etapas individuais em um cenário de uso e pensar em como ele pode ser explorado de forma maliciosa, podem ser descobertas possíveis falhas ou aspectos do aplicativo que não estão bem definidos. A chave é descrever todos os cenários possíveis ou, pelo menos, os mais críticos de uso e uso indevido.

Os cenários de uso indevido permitem a análise da aplicação do ponto de vista do invasor e contribuem para identificar potenciais vulnerabilidades e as contramedidas que precisam ser implementadas para mitigar o impacto causado pela potencial exposição a tais vulnerabilidades. Dados todos os casos de uso e abuso, é importante analisá-los para determinar quais são os mais críticos e que precisam ser documentados nos requisitos de segurança. A identificação dos casos mais críticos de uso indevido e abuso orienta a documentação dos requisitos de segurança e os controles necessários onde os riscos de segurança devem ser mitigados.

Para derivar requisitos de segurança de casos de uso e uso indevido, é importante definir os cenários funcionais e os cenários negativos e colocá-los em forma gráfica. O exemplo a seguir é uma metodologia passo a passo para o caso de derivação de requisitos de segurança para autenticação.

Etapa 1: Descreva o Cenário Funcional

O usuário se autentica fornecendo um nome de usuário e uma senha. O aplicativo concede acesso aos usuários com base na autenticação das credenciais do usuário pelo aplicativo e fornece erros específicos ao usuário quando a validação falha.

Etapa 2: descreva o cenário negativo

O invasor quebra a autenticação por meio de um ataque de força bruta ou de dicionário de senhas e coleta de vulnerabilidades de contas no aplicativo. Os erros de validação fornecem informações específicas para um invasor que é usado para adivinhar quais contas são contas registradas válidas (nomes de usuário). O invasor então tenta forçar a senha de uma conta válida. Um ataque de força bruta a senhas com comprimento mínimo de quatro dígitos pode ter sucesso com um número limitado de tentativas (ou seja, 10^4).

Etapa 3: Descrever cenários funcionais e negativos com casos de uso e uso indevido

O exemplo gráfico abaixo descreve a derivação dos requisitos de segurança por meio de casos de uso e uso indevido. O cenário funcional consiste nas ações do usuário (inserir um nome de usuário e senha) e nas ações do aplicativo (autenticar o usuário e fornecer uma mensagem de erro se a validação falhar). O caso de uso indevido consiste nas ações do invasor, ou seja, tentar quebrar a autenticação por força bruta da senha por meio de um ataque de dicionário e adivinhar os nomes de usuário válidos a partir de mensagens de erro. Ao representar graficamente as ameaças às ações do usuário (usos indevidos), é possível derivar as contramedidas como as ações do aplicativo que mitigam tais ameaças.

Machine Translated by Google

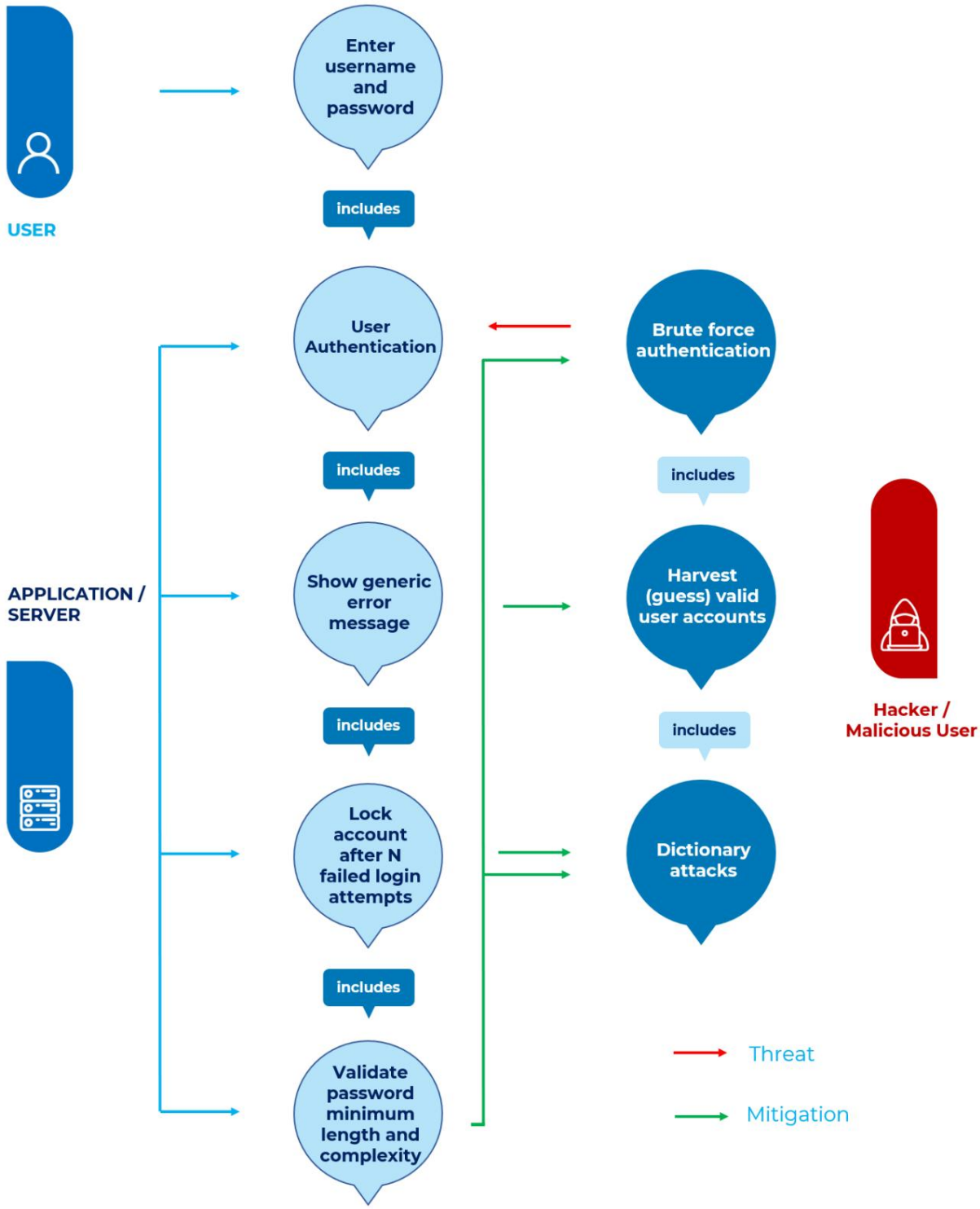

Figura 2-5: Caso de uso e uso indevido

Etapa 4: elicite os requisitos de segurança

Nesse caso, são derivados os seguintes requisitos de segurança para autenticação:

1. Os requisitos de senhas devem estar alinhados com os padrões atuais para complexidade suficiente.
2. As contas devem ser bloqueadas após cinco tentativas malsucedidas de login.
3. As mensagens de erro de login devem ser genéricas.

Esses requisitos de segurança precisam ser documentados e testados.

Testes de segurança integrados em fluxos de trabalho de desenvolvimento e teste

Testes de segurança no fluxo de trabalho de desenvolvimento

Machine Translated by Google

Os testes de segurança durante a fase de desenvolvimento do SDLC representam a primeira oportunidade para os desenvolvedores garantirem que os componentes de software individuais que desenvolveram sejam testados em termos de segurança antes de serem integrados a outros componentes ou incorporados ao aplicativo. Os componentes de software podem consistir em artefatos de software, como funções, métodos e classes, bem como interfaces de programação de aplicativos, bibliotecas e arquivos executáveis. Para testes de segurança, os desenvolvedores podem confiar nos resultados da análise do código-fonte para verificar estaticamente se o código-fonte desenvolvido não inclui vulnerabilidades potenciais e está em conformidade com os padrões de codificação segura. Os testes de unidade de segurança podem verificar ainda mais dinamicamente (ou seja, em tempo de execução) se os componentes funcionam conforme o esperado. Antes de integrar alterações de código novas e existentes na construção do aplicativo, os resultados da análise estática e dinâmica devem ser revisados e validados.

A validação do código-fonte antes da integração na construção de aplicativos geralmente é de responsabilidade do desenvolvedor sênior. Os desenvolvedores seniores costumam ser especialistas no assunto de segurança de software e sua função é liderar a revisão segura do código. Eles devem tomar decisões sobre aceitar o código a ser lançado na construção do aplicativo ou exigir mais alterações e testes. Esse fluxo de trabalho seguro de revisão de código pode ser aplicado por meio de aceitação formal, bem como por meio de uma verificação em uma ferramenta de gerenciamento de fluxo de trabalho. Por exemplo, assumindo o fluxo de trabalho típico de gerenciamento de defeitos usado para bugs funcionais, os bugs de segurança que foram corrigidos por um desenvolvedor podem ser relatados em um sistema de gerenciamento de defeitos ou alterações. O mestre de construção pode então analisar os resultados dos testes relatados pelos desenvolvedores na ferramenta e conceder aprovações para verificar as alterações de código na construção do aplicativo.

Teste de segurança no fluxo de trabalho de teste

Depois que os componentes e as alterações de código forem testados pelos desenvolvedores e verificados na construção do aplicativo, a próxima etapa mais provável no fluxo de trabalho do processo de desenvolvimento de software será realizar testes no aplicativo como uma entidade inteira. Este nível de teste é geralmente chamado de teste integrado e teste de nível de sistema. Quando os testes de segurança fazem parte dessas atividades de teste, eles podem ser usados para validar tanto a funcionalidade de segurança do aplicativo como um todo, quanto a exposição a vulnerabilidades no nível do aplicativo. Esses testes de segurança no aplicativo incluem testes de caixa branca, como análise de código-fonte, e testes de caixa preta, como testes de penetração. Os testes também podem incluir testes de caixa cinza, nos quais se presume que o testador tenha algum conhecimento parcial sobre o aplicativo. Por exemplo, com algum conhecimento sobre o gerenciamento de sessão do aplicativo, o testador pode entender melhor se as funções de logout e tempo limite estão devidamente protegidas.

O alvo dos testes de segurança é o sistema completo vulnerável a ataques. Durante esta fase, é possível que os testadores de segurança determinem se as vulnerabilidades podem ser exploradas. Isso inclui vulnerabilidades comuns de aplicativos da Web, bem como problemas de segurança que foram identificados anteriormente no SDLC com outras atividades, como modelagem de ameaças, análise de código-fonte e revisões seguras de código.

Normalmente, os engenheiros de teste, e não os desenvolvedores de software, realizam testes de segurança quando o aplicativo está no escopo de testes do sistema de integração. Os engenheiros de teste têm conhecimento de segurança de vulnerabilidades de aplicativos da web, técnicas de teste de caixa preta e caixa branca e possuem a validação dos requisitos de segurança nesta fase. Para realizar testes de segurança, é um pré-requisito que os casos de teste de segurança sejam documentados nas diretrizes e procedimentos de testes de segurança.

Um engenheiro de testes que valide a segurança da aplicação no ambiente de sistema integrado poderá liberar a aplicação para testes no ambiente operacional (por exemplo, testes de aceitação do usuário). Nesta fase do SDLC (ou seja, validação), os testes funcionais da aplicação são geralmente de responsabilidade dos testadores de controle de qualidade, enquanto hackers white-hat ou consultores de segurança são geralmente responsáveis pelos testes de segurança. Algumas organizações contam com sua própria equipe especializada de hackers éticos para realizar tais testes quando uma avaliação de terceiros não é necessária (como para fins de auditoria).

Como esses testes podem, às vezes, ser a última linha de defesa para corrigir vulnerabilidades antes que o aplicativo seja liberado para produção, é importante que os problemas sejam resolvidos conforme recomendado pela equipe de testes. As recomendações podem incluir alterações de código, design ou configuração. Neste nível, os auditores de segurança e os responsáveis pela segurança da informação discutem as questões de segurança reportadas e analisam os riscos potenciais de acordo com os procedimentos de gestão de risco da informação. Tais procedimentos podem exigir que a equipe de desenvolvimento corrija todas as vulnerabilidades de alto risco antes que o aplicativo possa ser implantado, a menos que tais riscos sejam reconhecidos e aceitos.

Testes de segurança do desenvolvedor

Machine Translated by Google

Testes de segurança na fase de codificação: testes unitários

Do ponto de vista do desenvolvedor, o principal objetivo dos testes de segurança é validar se o código está sendo desenvolvido em conformidade com os requisitos dos padrões de codificação segura. Os próprios artefatos de codificação dos desenvolvedores (como funções, métodos, classes, APIs e bibliotecas) precisam ser validados funcionalmente antes de serem integrados à construção do aplicativo.

Os requisitos de segurança que os desenvolvedores devem seguir devem ser documentados em padrões de codificação seguros e validados com análises estáticas e dinâmicas. Se a atividade de teste de unidade seguir uma revisão de código segura, os testes de unidade poderão validar se as alterações de código exigidas pelas revisões de código seguras foram implementadas corretamente. Tanto as revisões seguras de código quanto a análise do código-fonte por meio de ferramentas de análise de código-fonte podem ajudar os desenvolvedores a identificar problemas de segurança no código-fonte à medida que ele é desenvolvido. Ao usar testes unitários e análises dinâmicas (por exemplo, depuração), os desenvolvedores podem validar a funcionalidade de segurança dos componentes, bem como verificar se as contramedidas desenvolvidas atenuam quaisquer riscos de segurança previamente identificados por meio de modelagem de ameaças e análise de código-fonte.

Uma boa prática para desenvolvedores é criar casos de teste de segurança como um conjunto genérico de testes de segurança que faz parte da estrutura de teste de unidade existente. Um conjunto genérico de testes de segurança pode ser derivado de casos de uso e uso indevido previamente definidos para funções, métodos e classes de teste de segurança. Um conjunto genérico de testes de segurança pode incluir casos de teste de segurança para validar requisitos positivos e negativos para controles de segurança, como:

- Identidade, autenticação e controle de acesso
- Validação e codificação de entrada
- Criptografia
- Gerenciamento de usuários e sessões
- Tratamento de erros e exceções
- Auditoria e registro

Os desenvolvedores capacitados com uma ferramenta de análise de código-fonte integrada em seu IDE, padrões de codificação seguros e uma estrutura de teste de unidade de segurança podem avaliar e verificar a segurança dos componentes de software que estão sendo desenvolvidos.
Os casos de teste de segurança podem ser executados para identificar possíveis problemas de segurança que tenham causas raiz no código-fonte: além da validação de entrada e saída de parâmetros que entram e saem dos componentes, esses problemas incluem verificações de autenticação e autorização feitas pelo componente, proteção dos dados dentro do componente, tratamento seguro de exceções e erros e auditoria e registro seguros. Estruturas de teste unitário como JUnit, NUnit e CUnit podem ser adaptadas para verificar os requisitos de teste de segurança. No caso de testes funcionais de segurança, os testes de nível unitário podem testar a funcionalidade dos controles de segurança no nível do componente de software, como funções, métodos ou classes. Por exemplo, um caso de teste poderia validar a validação de entrada e saída (por exemplo, saneamento de variáveis) e verificações de limites para variáveis, afirmando a funcionalidade esperada do componente.

Os cenários de ameaças identificados com casos de uso e uso indevido podem ser usados para documentar os procedimentos para testar componentes de software. No caso de componentes de autenticação, por exemplo, os testes de unidade de segurança podem afirmar a funcionalidade de definir um bloqueio de conta, bem como o fato de que os parâmetros de entrada do usuário não podem ser abusados para contornar o bloqueio de conta (por exemplo, definindo o contador de bloqueio de conta para um número negativo).

No nível do componente, os testes unitários de segurança podem validar asserções positivas e negativas, como erros e tratamento de exceções. As exceções devem ser capturadas sem deixar o sistema em um estado inseguro, como uma potencial negação de serviço causada por recursos que não estão sendo desalocados (por exemplo, identificadores de conexão não fechados dentro de um bloco de instrução final), bem como uma potencial elevação de privilégios (por exemplo, , privilégios mais altos adquiridos antes da exceção ser lançada e não redefinidos para o nível anterior antes de sair da função). O tratamento seguro de erros pode validar a divulgação potencial de informações por meio de mensagens de erro informativas e rastreamentos de pilha.

Os casos de teste de segurança em nível de unidade podem ser desenvolvidos por um engenheiro de segurança que seja especialista no assunto em segurança de software e também seja responsável por validar se os problemas de segurança no código-fonte foram corrigidos e podem ser verificados na construção do sistema integrado. Normalmente, o gerente das compilações do aplicativo também garante que bibliotecas e arquivos executáveis de terceiros sejam avaliados em termos de segurança quanto a possíveis vulnerabilidades antes de serem integrados na compilação do aplicativo.

Machine Translated by Google

Cenários de ameaças para vulnerabilidades comuns que têm causas raiz em codificação insegura também podem ser documentados no guia de testes de segurança do desenvolvedor. Quando uma correção é implementada para um defeito de codificação identificado com a análise do código-fonte, por exemplo, os casos de teste de segurança podem verificar se a implementação da alteração do código segue os requisitos de codificação segura documentados nos padrões de codificação segura.

A análise do código-fonte e os testes unitários podem validar que a alteração do código mitiga a vulnerabilidade exposta pelo defeito de codificação identificado anteriormente. Os resultados da análise automatizada de código seguro também podem ser usados como portas de check-in automático para controle de versão, por exemplo, artefatos de software não podem ser verificados na construção com problemas de codificação de gravidade alta ou média.

Testes de segurança de testadores funcionais

Testes de segurança durante a fase de integração e validação: testes de sistema integrado e testes de operação

O principal objetivo dos testes de sistemas integrados é validar o conceito de “defesa em profundidade”, ou seja, que a implementação de controles de segurança proporciona segurança em diferentes camadas. Por exemplo, a falta de validação de entrada ao chamar um componente integrado à aplicação costuma ser um fator que pode ser testado com testes de integração.

O ambiente de teste do sistema de integração também é o primeiro ambiente onde os testadores podem simular cenários reais de ataque que podem ser potencialmente executados por um usuário externo ou interno mal-intencionado do aplicativo. Os testes de segurança neste nível podem validar se as vulnerabilidades são reais e podem ser exploradas por invasores. Por exemplo, uma vulnerabilidade potencial encontrada no código-fonte pode ser classificada como de alto risco devido à exposição a potenciais usuários mal-intencionados, bem como devido ao impacto potencial (por exemplo, acesso a informações confidenciais).

Cenários reais de ataque podem ser testados tanto com técnicas de teste manuais quanto com ferramentas de teste de penetração. Os testes de segurança desse tipo também são chamados de testes de hacking ético. Do ponto de vista dos testes de segurança, estes são testes orientados ao risco e têm como objetivo testar a aplicação no ambiente operacional. O destino é a construção do aplicativo que representa a versão do aplicativo que está sendo implantada na produção.

Incluir testes de segurança na fase de integração e validação é fundamental para identificar vulnerabilidades devido à integração de componentes, bem como validar a exposição de tais vulnerabilidades. Os testes de segurança de aplicativos exigem um conjunto especializado de habilidades, incluindo conhecimento de software e de segurança, que não são típicos dos engenheiros de segurança. Como resultado, as organizações são frequentemente obrigadas a treinar seus desenvolvedores de software em técnicas de hacking ético e procedimentos e ferramentas de avaliação de segurança. Um cenário realista é desenvolver esses recursos internamente e documentá-los em guias e procedimentos de testes de segurança que levem em consideração o conhecimento do desenvolvedor em testes de segurança. Uma chamada “folha de dicas ou lista de verificação de casos de teste de segurança”, por exemplo, pode fornecer casos de teste simples e vetores de ataque que podem ser usados pelos testadores para validar a exposição a vulnerabilidades comuns, como falsificação, divulgação de informações, buffer overflows, strings de formatação, SQL injeção e injeção XSS, XML, SOAP, problemas de canonização, negação de serviço e código gerenciado e controles ActiveX (por exemplo, .NET). Uma primeira bateria desses testes pode ser realizada manualmente com um conhecimento básico de segurança de software.

O primeiro objetivo dos testes de segurança pode ser a validação de um conjunto de requisitos mínimos de segurança. Esses casos de teste de segurança podem consistir em forçar manualmente o aplicativo a erros e estados excepcionais e coletar conhecimento do comportamento do aplicativo. Por exemplo, as vulnerabilidades de injeção SQL podem ser testadas manualmente injetando vetores de ataque por meio de entrada do usuário e verificando se as exceções SQL são devolvidas ao usuário. A evidência de um erro de exceção SQL pode ser uma manifestação de uma vulnerabilidade que pode ser explorada.

Um teste de segurança mais aprofundado pode exigir o conhecimento do testador sobre técnicas e ferramentas de teste especializadas. Além da análise de código-fonte e testes de penetração, essas técnicas incluem, por exemplo: código-fonte e injeção binária de falhas, análise de propagação de falhas e cobertura de código, testes fuzz e engenharia reversa. O guia de testes de segurança deve fornecer procedimentos e recomendar ferramentas que podem ser usadas pelos testadores de segurança para realizar avaliações de segurança aprofundadas.

O próximo nível de testes de segurança após os testes do sistema de integração é realizar testes de segurança no ambiente de aceitação do usuário. Existem vantagens exclusivas em realizar testes de segurança no ambiente operacional. O ambiente de teste de aceitação do usuário (UAT) é o mais representativo da configuração da versão, com exceção dos dados (por exemplo, dados de teste são usados no lugar de dados reais). Uma característica dos testes de segurança no UAT é testar

Machine Translated by Google

problemas de configuração de segurança. Em alguns casos, estas vulnerabilidades podem representar riscos elevados. Por exemplo, o servidor que hospeda o aplicativo Web pode não estar configurado com privilégios mínimos, certificado SSL válido e configuração segura, serviços essenciais desabilitados e diretório raiz da Web limpo de páginas da Web de teste e administração.

Análise e relatórios de dados de teste de segurança

Metas para métricas e medições de testes de segurança

Definir os objetivos das métricas e medições dos testes de segurança é um pré-requisito para usar dados de testes de segurança para análise de risco e processos de gerenciamento. Por exemplo, uma medição, como o número total de vulnerabilidades encontradas em testes de segurança, pode quantificar a postura de segurança do aplicativo. Essas medições também ajudam a identificar objetivos de segurança para testes de segurança de software, por exemplo, reduzindo o número de vulnerabilidades a um número mínimo aceitável antes que o aplicativo seja implantado em produção.

Outro objetivo gerenciável poderia ser comparar a postura de segurança de aplicativos com uma linha de base para avaliar melhorias nos processos de segurança de aplicativos. Por exemplo, a linha de base das métricas de segurança pode consistir em um aplicativo que foi testado apenas com testes de penetração. Os dados de segurança obtidos de uma aplicação que também foi testada durante a codificação devem mostrar uma melhoria (por exemplo, menos vulnerabilidades) quando comparados com os dados
linha de base.

Nos testes de software tradicionais, o número de defeitos de software, como os bugs encontrados em um aplicativo, pode fornecer uma medida da qualidade do software. Da mesma forma, os testes de segurança podem fornecer uma medida de segurança de software. Do ponto de vista do gerenciamento e do relatório de defeitos, os testes de qualidade e segurança de software podem usar categorizações semelhantes para causas raízes e esforços de correção de defeitos. Do ponto de vista da causa raiz, um defeito de segurança pode ser devido a um erro de design (por exemplo, falhas de segurança) ou a um erro de codificação (por exemplo, bug de segurança). Da perspectiva do esforço necessário para corrigir um defeito, tanto os defeitos de segurança quanto os de qualidade podem ser medidos em termos de horas do desenvolvedor para implementar a correção, as ferramentas e recursos necessários e o custo para implementar a correção.

Uma característica dos dados de teste de segurança, em comparação com os dados de qualidade, é a categorização em termos de ameaça, a exposição da vulnerabilidade e o impacto potencial causado pela vulnerabilidade para determinar o risco. Testar a segurança dos aplicativos consiste em gerenciar riscos técnicos para garantir que as contramedidas do aplicativo atendam a níveis aceitáveis. Por esta razão, os dados dos testes de segurança precisam de apoiar a estratégia de risco de segurança em pontos de verificação críticos durante o SDLC. Por exemplo, vulnerabilidades encontradas no código-fonte com análise de código-fonte representam uma medida inicial de risco.
Uma medida de risco (por exemplo, alto, médio, baixo) para a vulnerabilidade pode ser calculada determinando os fatores de exposição e probabilidade e validando a vulnerabilidade com testes de penetração. As métricas de risco associadas às vulnerabilidades encontradas nos testes de segurança capacitam o gerenciamento de negócios a tomar decisões de gerenciamento de risco, como decidir se os riscos podem ser aceitos, mitigados ou transferidos em diferentes níveis dentro da organização (por exemplo, riscos comerciais e também técnicos).

Ao avaliar a postura de segurança de uma aplicação, é importante levar em consideração alguns fatores, como o tamanho da aplicação que está sendo desenvolvida. Foi comprovado estatisticamente que o tamanho do aplicativo está relacionado ao número de problemas encontrados no aplicativo durante o teste. Como os testes reduzem os problemas, é lógico que aplicativos de tamanho maior sejam testados com mais frequência do que aplicativos de tamanho menor.

Quando os testes de segurança são feitos em várias fases do SDLC, os dados de teste podem comprovar a capacidade dos testes de segurança em detectar vulnerabilidades assim que elas são introduzidas. Os dados de teste também podem comprovar a eficácia da remoção das vulnerabilidades através da implementação de contramedidas em diferentes pontos de verificação do SDLC. Uma medição deste tipo também é definida como “métrica de contenção” e fornece uma medida da capacidade de uma avaliação de segurança realizada em cada fase do processo de desenvolvimento para manter a segurança dentro de cada fase. Estas métricas de contenção também são um fator crítico na redução do custo de correção das vulnerabilidades. É menos dispendioso lidar com vulnerabilidades na mesma fase do SDLC em que são encontradas, em vez de corrigi-las posteriormente em outra fase.

As métricas de teste de segurança podem dar suporte à análise de gerenciamento de riscos, custos e defeitos de segurança quando estão associadas a metas tangíveis e cronometradas, como:

- Reduzindo o número geral de vulnerabilidades em 30%.
- Correção de problemas de segurança dentro de um determinado prazo (por exemplo, antes do lançamento beta).

Machine Translated by Google

Os dados de testes de segurança podem ser absolutos, como o número de vulnerabilidades detectadas durante a revisão manual de código, ou comparativos, como o número de vulnerabilidades detectadas em revisões de código em comparação com testes de penetração. Para responder a questões sobre a qualidade do processo de segurança, é importante determinar uma linha de base para o que pode ser considerado aceitável e bom.

Os dados de teste de segurança também podem apoiar objetivos específicos da análise de segurança. Esses objetivos poderiam ser a conformidade com regulamentações de segurança e padrões de segurança da informação, gerenciamento de processos de segurança, identificação de causas básicas de segurança e melhorias de processos, além de análise de custo-benefício de segurança.

Quando os dados de teste de segurança são relatados, eles devem fornecer métricas para apoiar a análise. O escopo da análise é a interpretação dos dados de teste para encontrar pistas sobre a segurança do software que está sendo produzido, bem como a eficácia do processo.

Alguns exemplos de pistas apoiadas por dados de testes de segurança podem ser:

- As vulnerabilidades são reduzidas a um nível aceitável para liberação?
- Como a qualidade de segurança deste produto se compara a produtos de software similares?
- Todos os requisitos de teste de segurança estão sendo atendidos?
- Quais são as principais causas dos problemas de segurança?
- Quantas são as falhas de segurança em comparação com os bugs de segurança?
- Qual atividade de segurança é mais eficaz para encontrar vulnerabilidades?
- Qual equipe é mais produtiva na correção de defeitos e vulnerabilidades de segurança?
- Qual porcentagem de vulnerabilidades gerais é de alto risco?
- Quais ferramentas são mais eficazes na detecção de vulnerabilidades de segurança?
- Que tipo de testes de segurança são mais eficazes para encontrar vulnerabilidades (por exemplo, testes de caixa branca versus caixa preta)?
- Quantos problemas de segurança são encontrados durante revisões de código seguro?
- Quantos problemas de segurança são encontrados durante revisões de projetos seguros?

Para fazer um bom julgamento usando os dados de teste, é importante ter um bom entendimento do processo de teste, bem como das ferramentas de teste. Uma taxonomia de ferramentas deve ser adotada para decidir quais ferramentas de segurança usar. As ferramentas de segurança podem ser qualificadas como boas para encontrar vulnerabilidades comuns e conhecidas, ao direcionar diferentes artefatos.

É importante observar que problemas de segurança desconhecidos não são testados. O fato de um teste de segurança estar isento de problemas não significa que o software ou aplicativo seja bom.

Mesmo as ferramentas de automação mais sofisticadas não são páreo para um testador de segurança experiente. Confiar apenas nos resultados de testes bem-sucedidos de ferramentas automatizadas dará aos profissionais de segurança uma falsa sensação de segurança. Normalmente, quanto mais experientes os testadores de segurança forem com a metodologia e as ferramentas de teste de segurança, melhores serão os resultados do teste e da análise de segurança. É importante que os gestores que investem em ferramentas de testes de segurança também considerem um investimento na contratação de recursos humanos qualificados, bem como na formação em testes de segurança.

Requisitos de Relatório

A postura de segurança de uma aplicação pode ser caracterizada do ponto de vista do efeito, como o número de vulnerabilidades e a classificação de risco das vulnerabilidades, bem como do ponto de vista da causa ou origem, como erros de codificação, falhas arquitetônicas e problemas de configuração.

As vulnerabilidades podem ser classificadas de acordo com diferentes critérios. A métrica de gravidade de vulnerabilidade mais comumente usada é o Common Vulnerability Scoring System (CVSS), padrão mantido pelo Fórum de Equipes de Segurança e Resposta a Incidentes (FIRST).

Ao relatar dados de testes de segurança, a prática recomendada é incluir as seguintes informações:

- uma categorização de cada vulnerabilidade por tipo;
- a ameaça à segurança a que cada questão está exposta;

Machine Translated by Google

- a causa raiz de cada problema de segurança, como bug ou falha;
- cada técnica de teste usada para encontrar os problemas;
- a remediação, ou contramedida, para cada vulnerabilidade; e a classificação de
- gravidade de cada vulnerabilidade (por exemplo, pontuação alta, média, baixa ou CVSS).

Ao descrever qual é a ameaça à segurança, será possível compreender se e porque é que o controlo de mitigação é ineficaz na mitigação da ameaça.

Relatar a causa raiz do problema pode ajudar a identificar o que precisa ser corrigido. No caso de testes de caixa branca, por exemplo, a causa raiz da vulnerabilidade na segurança do software será o código-fonte ofensivo.

Depois que os problemas forem relatados, também é importante fornecer orientação ao desenvolvedor de software sobre como testar novamente e encontrar a vulnerabilidade. Isso pode envolver o uso de uma técnica de teste de caixa branca (por exemplo, revisão de código de segurança com um analisador de código estático) para descobrir se o código é vulnerável. Se uma vulnerabilidade puder ser encontrada através de um teste de penetração de caixa preta, o relatório de teste também precisará fornecer informações sobre como validar a exposição da vulnerabilidade ao front-end (por exemplo, cliente).

As informações sobre como corrigir a vulnerabilidade devem ser detalhadas o suficiente para que um desenvolvedor implemente uma correção. Deve fornecer exemplos de codificação segura, alterações de configuração e referências adequadas.

Por fim, a classificação de gravidade contribui para o cálculo da classificação de risco e ajuda a priorizar o esforço de remediação.
Normalmente, atribuir uma classificação de risco à vulnerabilidade envolve uma análise de risco externa baseada em fatores como impacto e exposição.

Casos de negócios

Para que as métricas de teste de segurança sejam úteis, elas precisam fornecer valor às partes interessadas nos dados de teste de segurança da organização. As partes interessadas podem incluir gerentes de projeto, desenvolvedores, escritórios de segurança da informação, auditores e diretores de informação. O valor pode ser em termos do caso de negócio que cada parte interessada do projeto possui, em termos de papel e responsabilidade.

Os desenvolvedores de software analisam os dados dos testes de segurança para mostrar que o software é codificado de forma segura e eficiente. Isso permite que eles defendam o uso de ferramentas de análise de código-fonte, sigam padrões de codificação seguros e participem de treinamento em segurança de software.

Os gerentes de projeto procuram dados que lhes permitam gerenciar e utilizar com sucesso atividades e recursos de testes de segurança de acordo com o plano do projeto. Para os gerentes de projeto, os dados dos testes de segurança podem mostrar que os projetos estão dentro do cronograma e avançando dentro das datas de entrega, e estão melhorando durante os testes.

Os dados de teste de segurança também ajudam no caso de negócios para testes de segurança se a iniciativa vier de oficiais de segurança da informação (ISOs). Por exemplo, pode fornecer evidências de que os testes de segurança durante o SDLC não afetam a entrega do projeto, mas reduzem a carga de trabalho geral necessária para resolver vulnerabilidades posteriormente na produção.

Para os auditores de conformidade, as métricas de teste de segurança fornecem um nível de garantia de segurança de software e confiança de que a conformidade com os padrões de segurança é abordada por meio dos processos de revisão de segurança dentro da organização.

Finalmente, os Chief Information Officers (CIOs) e os Chief Information Security Officers (CISOs), responsáveis pelo orçamento que precisa ser alocado em recursos de segurança, procuram a derivação de uma análise de custo-benefício a partir dos dados de testes de segurança. Isto permite-lhes tomar decisões informadas sobre em que atividades e ferramentas de segurança investir. Uma das métricas que apoia essa análise é o Retorno do Investimento (ROI) em segurança. Para derivar tais métricas a partir de dados de testes de segurança, é importante quantificar o diferencial entre o risco, devido à exposição de vulnerabilidades, e a eficácia dos testes de segurança na mitigação do risco de segurança e, em seguida, considerar esta lacuna com o custo da segurança. atividade de teste ou as ferramentas de teste adotadas.

Referências

- Pesquisa de 2002 do Instituto Nacional de Padrões dos EUA (NIST) sobre o custo de software inseguro para a economia dos EUA devido a testes de software inadequados