Princípios de projeto de software seguro

SEG786203 – CST em Análise e Desenvolvimento de Sistemas

Prof. Emerson Ribeiro de Mello

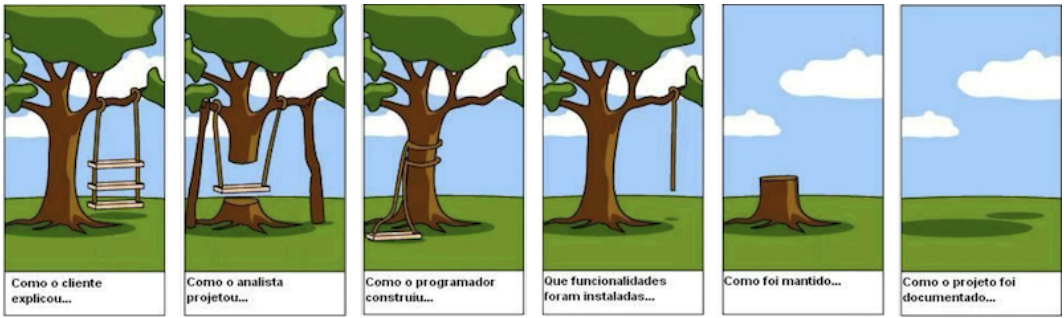
mello@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Campus São José

Licenciamento

Segurança de software

- Foco no desenvolvimento e uso de software que preserva de forma confiável as propriedades básicas de segurança das informações
- A segurança de um software depende de como os requisitos atendem às necessidades que o software deve atender e de como o software é projetado, implementado, testado, implantado, mantido e documentado



Princípios de projeto e implementação de mencanismos de segurança

- Os princípios de projeto de software seguro são diretrizes fundamentais visando a simplicidade e restrições (Bishop, 2003)
- A **simplicidade** faz com que o projeto e os mecanismos sejam mais fáceis de entender e reduz potencial inconsistências com as políticas de segurança
- **Restrições** são necessárias para minimizar o poder de uma entidade, limitando o que ela pode fazer

Princípios de projeto de software seguro

The Protection of Information in Computer Systems

Saltzer e Schroeder (1975) propuseram oito de princípios de projeto, sendo amplamente aceitos e ainda relevantes atualmente

1. Princípio do menor privilégio
2. Princípio da segurança por padrão
3. Princípio da economia de mecanismos
4. Princípio da mediação completa
5. Princípio do design aberto
6. Princípio da separação de privilégios
7. Princípio do menor compartilhamento de mecanismos
8. Princípio da aceitabilidade psicológica

Princípio do menor privilégio

Definição

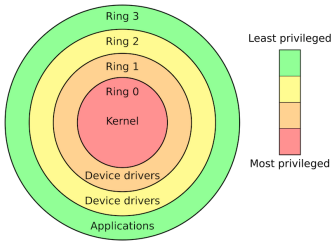
Conceder a um usuário ou processo apenas os privilégios necessários para realizar sua tarefa

- No Linux o *root* tem acesso a todos os recursos do sistema, sendo capaz de realizar qualquer operação sobre qualquer arquivo ou processo. A conta de administrador no Windows tem privilégios semelhantes

Princípio do menor privilégio

Definição

Conceder a um usuário ou processo apenas os privilégios necessários para realizar sua tarefa



Fonte: https://en.wikipedia.org/wiki/Principle_of_least_privilege

- **Anéis de privilégio** (rings) em CPU x86 (Intel) separa o código do sistema operacional em quatro níveis de privilégio

Princípio da segurança por padrão

Definição

A menos que a usuário ou processo seja dado acesso explícito a um objeto, ele deve ser negado e o estado de proteção do sistema deve permanecer inalterado

- O acesso padrão a um recurso é negado por padrão, a menos que haja uma autorização explícita
- Se um usuário não conseguir completar sua tarefa, o sistema deve desfazer qualquer alteração feita e retornar ao estado anterior
  - Assim, mesmo diante de uma falha, o sistema permanece seguro (*fail-secure*)

Princípio da segurança por padrão

```
mello@maq:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), deny (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
3306/tcp                   LIMIT IN    Anywhere
22/tcp                     LIMIT IN    Anywhere
224.0.0.0/24 on eno1       DENY IN     Anywhere
22/tcp (v6)                LIMIT IN    Anywhere (v6)
3306/tcp (v6)              LIMIT IN    Anywhere (v6)
```

- Regras de firewall do *Uncomplicated Firewall* (UFW) no Ubuntu
- Por padrão, qualquer tráfego de entrada é negado

Princípio da economia de mecanismos

Definição

O mecanismo de segurança deve ser o mais simples possível

- Mecanismos complexos geralmente fazem suposições sobre o sistema ou ambiente onde são executados
- Sistemas complexos podem expor um maior número de pontos de entrada que podem ser explorados por atacantes

Princípio da economia de mecanismos

- Cartão EMV (Europay, Mastercard e Visa) possui várias interfaces de comunicação
  - Número em relevo, chip, tarja magnética e NFC
- Cada interface pode apresentar vulnerabilidades distintas, aumentando a superfície de ataque
  - *skimming* para tarja magnética
  - *relay attack* para NFC
  - Cópia do número em relevo

Princípio da mediação completa

Definição

Todos os acessos a objetos devem ser verificados para garantir que eles continuam autorizados

- Quando um usuário ou processo tenta acessar um objeto, o sistema operacional deve mediar o acesso
- Se o usuário ou processo tentar acessar novamente, o sistema deve verificar novamente se o acesso é permitido
- Confiança zero (*zero trust*) é um exemplo de implementação desse princípio, onde cada acesso é verificado independentemente do histórico de acesso
  - Também conhecido como *never trust, always verify*

Princípio do design aberto

Definição

A segurança de um sistema não deve depender da obscuridade do design ou da implementação

- Se a segurança de um sistema depender da ignorância do atacante sobre o design ou implementação, o sistema é inseguro

Princípio do design aberto

Definição

A segurança de um sistema não deve depender da obscuridade do design ou da implementação

- Se a segurança de um sistema depender da ignorância do atacante sobre o design ou implementação, o sistema é inseguro
- O *Content Scramble System* (CSS) era um algoritmo criptográfico usado em DVDs para proteger o conteúdo contra cópia não autorizada
  - Confiaram na obscuridade do algoritmo para proteger o conteúdo, mas foi quebrado em 1999 permitindo a reprodução de DVDs em sistemas sem software licenciado, além de permitir a cópia do conteúdo

Princípio da separação de privilégios

Definição

Um sistema não deve conceder a um usuário ou processo controle sobre todos os aspectos de uma operação

- O acesso a um recurso deve ser dividido em partes menores, cada uma com um nível diferente de privilégio, sendo necessário a combinação de várias partes para completar a operação
- O princípio da separação de privilégios é uma forma de implementar o princípio do menor privilégio
- Exemplo: uma mesma pessoa não pode ser ordenadora e pagadora de uma transação financeira

Princípio do menor compartilhamento de mecanismos

Definição

Mecanismos usados para acessar os recursos não devem ser compartilhados

- Cópias de segurança (*backups*) devem ser armazenadas em locais diferentes dos originais, para evitar que um ataque a um local comprometa o outro
- No envenenamento de *prompt* (*prompt injection*), um usuário mal-intencionado pode inserir comandos ocultos para manipular as respostas do modelo de aprendizado de máquina

Princípio da aceitabilidade psicológica

Definição

Mecanismos de segurança não devem tornar o sistema mais difícil de usar do que seria sem eles

- Quando o Whatsapp ativou a criptografia de ponta a ponta, os usuários não perceberam nenhuma mudança perceptível, pois continuaram a usar o aplicativo da mesma forma
- A autenticação biométrica nos dispositivos móveis fez com que a grande maioria dos usuários a adotasse
- A autenticação multifator, a depender do fator, gera um impacto negativo na experiência do usuário

Teste de software

- Testes de software são uma parte importante para garantir a qualidade de um sistema
  - Testes funcionais, testes estruturais, testes de desempenho, testes de usabilidade, testes de segurança
- Teste de segurança implica em realizar testes específicos para identificar vulnerabilidades e avaliar a segurança de um sistema
  - Engloba testes funcionais, estruturais e de requisitos específicos para segurança

Teste de penetração

Penetration testing* ou *pentest

- Teste de segurança que simula um ataque real a um sistema para identificar vulnerabilidades e avaliar a segurança do sistema
- Pode ser realizado internamente, simulando um ataque de um usuário interno, ou externamente, simulando um ataque de um usuário externo
- Pode ser cego, onde a equipe de segurança tem conhecimento do teste, mas os testadores não têm conhecimento do sistema
- Pode ser duplamente cego, onde a equipe de segurança não tem conhecimento do teste

Teste de Caixa-Branca

White-box testing

- Avalia a segurança do sistema com base no conhecimento interno de seu funcionamento, estrutura e código-fonte
- Tem por objetivo identificar vulnerabilidades e falhas internas no sistema
  - Ex: credenciais de acesso armazenadas em texto claro, falta de validação de entradas, falta de controle de acesso

Teste de Caixa-Preta

Black-box testing

- Avalia a segurança do sistema com base no comportamento observado do sistema
- Simula um ataque real ao sistema, sem conhecimento interno de seu funcionamento, estrutura e código-fonte
  - Ex: tentativa de acesso não autorizado, tentativa de injeção de código, tentativa de negação de serviço

Teste de Caixa-Cinza

Gray-box testing

- Combina elementos de teste de caixa-branca e caixa-preta, com conhecimento parcial do sistema e de seu funcionamento
- Pode ter acesso a informações limitadas sobre o sistema, como uma conta de usuário com privilégios limitados

Teste de Segurança Estático

***Static Application Security Testing*, SAST**

- Análise estática do código-fonte, *bytecode* ou código binário de um aplicativo em busca de vulnerabilidades de segurança
- Ferramentas de SAST podem estar integradas diretamente nas IDEs
  - Exemplo: *Checkmarx*, *Fortify*, *Veracode*, *SonarQube*
- Podem ser invocadas por meio de *pipelines* de integração contínua
  - Ex: *Jenkins*, *GitLab CI/CD*, *GitHub Actions*
- Executado nas fases iniciais do ciclo de vida do desenvolvimento

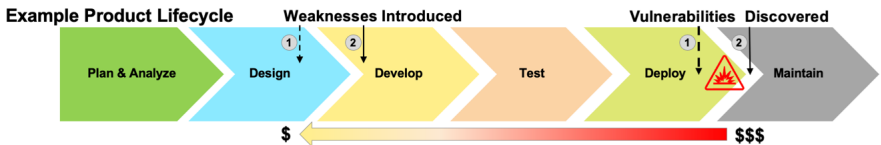
Teste de Segurança Dinâmico

***Dynamic Application Security Testing*, DAST**

- Testa a aplicação que está em execução em busca de vulnerabilidades de segurança
- Pode ser realizado em um ambiente de produção simulado ou em um ambiente de teste
- Ferramentas como *OWASP ZAP* e *Burp Suite* podem ser usadas para realizar testes de DAST em aplicações *web*
- Executado nas fases finais do ciclo de vida do desenvolvimento, o que pode resultar em custos mais elevados para corrigir vulnerabilidades

O que é melhor, SAST ou DAST?

- *Shift left* tem como objetivo encontrar e corrigir vulnerabilidades no início do processo de desenvolvimento
  - Também aprimora a experiência do usuário e pode reduzir custos indiretos
- *Shift right* é a prática de continuar a prática de testes, controle de qualidade e avaliação de desempenho no ambiente de pós-produção
  - Proteção contra riscos de segurança em constante evolução
- SAST e DAST são complementares, pois cada um tem suas vantagens e desvantagens



Fonte: https://cwe.mitre.org/about/

Blue Team e Red Team no desenvolvimento de software seguro

- **Blue Team** e **Red Team** vêm do mundo da cibersegurança ofensiva e defensiva e são amplamente utilizados para testar e fortalecer a segurança de sistemas e aplicações
  - **Grupo de pessoas autorizadas** a simular ataques cibernéticos (*Red Team*) e defender contra esses ataques (*Blue Team*)
- No contexto do desenvolvimento de software seguro, essas equipes desempenham papéis complementares na detecção e mitigação de vulnerabilidades

Red Team – ataque

- Simulam ataques reais para encontrar vulnerabilidades exploráveis
  - Assumem o papel de um atacante mal-intencionado
- Fazem testes de penetração (*pentest*) para identificar falhas no código e na infraestrutura
  - Injeção SQL, *Cross-Site Scripting* (XSS), *Cross-Site Request Forgery* (CSRF), *buffer overflow*
  - Explorar fraquezas em APIs, autenticação e criptografia
  - Avaliar engenharia social para identificar falhas humanas (*phishing*, *spoofing*)
- O objetivo é melhorar a segurança da informação da organização, demonstrando os impactos de ataques bem-sucedidos e o que funciona para os defensores (*Blue Team*) em um ambiente operacional

Blue Team – defesa

- Responsável pela defesa do sistema contra ataques simulados do *Red Team* ou de atacantes reais
  - sobre um período significativo de tempo
  - em um contexto operacional representativo (por exemplo, como parte de um exercício operacional)
  - de acordo com regras estabelecidas e monitoradas com a ajuda de um grupo neutro que arbitra a simulação ou exercício (o *White Team*)
- Implementa medidas de proteção e fortalece a segurança com base nas descobertas do *Red Team*

Trabalho conjunto do Blue Team e Red Team no desenvolvimento de software seguro

1. **Fase de projeto**
   - Blue Team define os requisitos de segurança, seguindo os princípios de projeto de software seguro
2. **Fase de implementação**
   - Red Team realiza testes de segurança manuais e automatizados, enquanto o Blue Team corrige as vulnerabilidades encontradas
3. **Fase de teste**
   - O Red Team realiza ataques simulados, e o Blue Team analisa logs e melhora as defesas
4. **Fase de implantação e manutenção**
   - Blue Team monitora logs, acessos e atividades suspeitas, enquanto o Red Team realiza testes periódicos

Modelagem de ameaças

- Processo para identificar e avaliar ameaças a um sistema, com o objetivo de mitigar riscos de segurança
- Ajuda a identificar vulnerabilidades desde as fases iniciais do desenvolvimento
- Facilita a priorização de riscos e a implementação de controles de segurança eficazes
- Promove uma compreensão profunda do sistema e de suas possíveis falhas
- Sua construção é uma tarefa desafiadora em sistemas grandes ou dinâmicos, além de demandar conhecimento técnico e de negócio

Modelagem de ameaças

Metodologias

- **STRIDE**
  - Foca nas categorias de ameaças: personificação, adulteração, negação de serviço, repúdio, divulgação de informações, elevação de privilégio
- **LINDDUN**
  - Foca em ameaças à privacidade, como não repúdio, rastreamento e identificação

Fonte: Ariel Jacob

Modelagem de ameaças

Exemplo de STRIDE

Cenário

Sistema de pagamento online permite que usuários realizem pagamentos a partir de seus cartões de crédito ou via carteira digital

- **Cadastro e login**
  - Usuários podem criar contas e acessar o sistema com e-mail e senha
- **Adição de cartão de crédito**
  - Os usuários podem salvar seus cartões para pagamentos futuros
- **Processamento de pagamentos**
  - O sistema permite transações entre clientes e lojistas
- **Histórico de transações**
  - Os usuários podem visualizar suas compras anteriores
- **Recuperação de senha**
  - Caso o usuário esqueça a senha, ele pode recuperá-la via e-mail

Identificação de ameaças usando STRIDE

Categoria	Definição	Ameaça
Spoofing	Personificação	*Phishing* para roubo de credenciais
Tampering	Adulteração de dados	Modificar transações em trânsito
Repudiation	Impossibilidade de rastrear ações	Cliente alega que não fez a transação
Information Disclosure	Exposição indevida de dados	Erro na API pode expor informações de cartões de crédito
Denial of Service	Impedir o acesso ao sistema	Grande volume de requisições ao servidor para sobrecarregá-lo
Elevation of Privilege	Elevação indevido de privilégio	*bug* na lógica de autenticação pode permitir que um usuário comum obtenha permissões administrativas

Identificação de ameaças usando STRIDE

Categoria	Definição	Ameaça
Spoofing	Personificação	*Phishing* para roubo de credenciais
Tampering	Adulteração de dados	Modificar transações em trânsito
Repudiation	Impossibilidade de rastrear ações	Cliente alega que não fez a transação
Information Disclosure	Exposição indevida de dados	Erro na API pode expor informações de cartões de crédito
Denial of Service	Impedir o acesso ao sistema	Grande volume de requisições ao servidor para sobrecarregá-lo
Elevation of Privilege	Elevação indevido de privilégio	*bug* na lógica de autenticação pode permitir que um usuário comum obtenha permissões administrativas

- Para cada ameaça, descreva o impacto que ela pode causar ao sistema e uma solução para mitigá-la
  - Ex: ameaça: exposição indevida de dados; impacto: vazamento de dados financeiros; mitigação: usar criptografia com TLS

Ferramentas para modelagem de ameaças

- **OWASP Threat Dragon**
  - Ferramenta *web* de modelagem de ameaças gratuita e de código aberto
  - Integração com o *OWASP Application Security Verification Standard* (ASVS)
  - Integração com o *OWASP Top Ten*
  - https://www.threatdragon.com
- **Microsoft Threat Modeling Tool**
  - Ferramenta desktop gratuita para modelagem de ameaças
  - Integração com o *Microsoft Security Development Lifecycle* (SDL)
  - https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling

Aula baseada em I

ADKINS, Heather *et al.* **Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems**. O'Reilly Media, mar. 2020. ISBN 978-1-492-08313-9.

BISHOP, Matt. **Computer Security: Art and Science**. 1. ed.: Addison-Wesley, 2003. ISBN 0201440997.

HAT, Red. **Shift left e shift right**. Mar. 2024. Disponível em: https://www.redhat.com/pt-br/topics/devops/shift-left-vs-shift-right. Acesso em: 7 fev. 2025.

KELLY, Judy; SASTRE, David. **Security by design: Security principles and threat modeling**. Fev. 2023. Disponível em: https://www.redhat.com/en/blog/security-design-security-principles-and-threat-modeling. Acesso em: 7 fev. 2025.