



Introdução

Segurança não é "preto e branco" → Segurança é "é provado seguro"

Vulnerabilidade: falha de design ou programação fraca que pode permitir um atacante explorar software para um propósito malicioso — uma instância ou falha que viola uma política de segurança implícita ou explícita

Exploit: peça de software, pedaço de dados ou sequência de comandos que tira vantagem de uma vulnerabilidade para tentar causar comportamento não pretendido ou não antecipado — utilização maliciosa de uma vulnerabilidade

Vírus: exploit automatizado

Proteção: AV; IDS; FW; ...

Ameaça: ator ou agente que é uma fonte de perigo, capaz de violar a confidencialidade, disponibilidade ou integridade dos ativos de informação e da política de segurança OU classe de exploits

Proteção: políticas; forense; ...

A engenharia de software seguro é a proteção contra vulnerabilidades!

SOFTWARE SEGURO > SOFTWARE DE SEGURANÇA

Segurança de Software: capacidade de prevenir funcionalidade não intencional em todas as camadas da pilha e em todas as partes do sistema

Se se faz X, mitigate Y, o que vale a pena deixar a Z!

Ciclo de Vida de Desenvolvimento

Confidencialidade: o sistema não deve divulgar qualquer informação presuposta para ser oculta

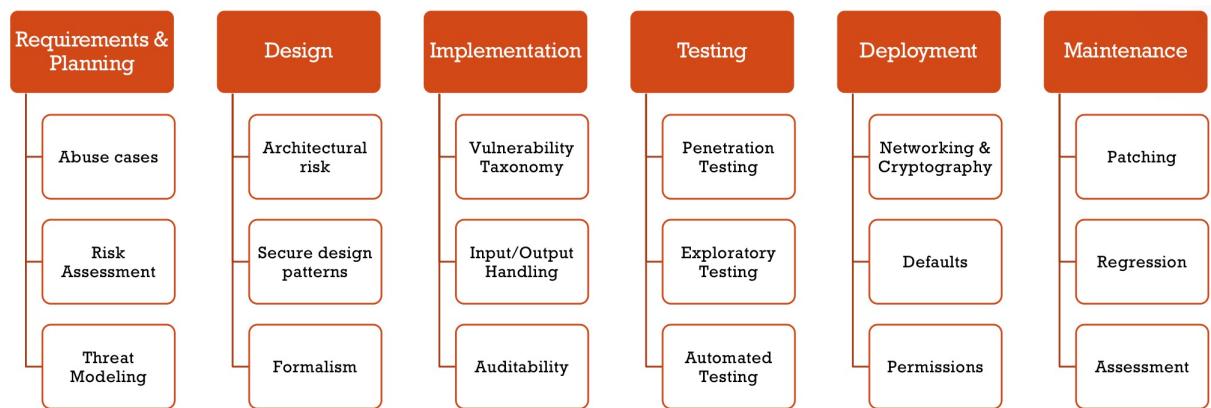
Integridade: o sistema não deve permitir que ativos sejam subvertidos por utilizadores não autorizados — o que está no sistema (dados armazenados e funcionalidade executada) deve ser confiável

Disponibilidade: o sistema deve ser capaz de estar disponível e operacional para os utilizadores

Auditabilidade/Não-Repúdio: um ator não deve ser capaz de negar ou disputar as suas ações

PRINCÍPIOS DE DESIGN

1. Defesa em Profundidade
2. Privilégio Mínimo
3. Falhar de Forma Segura
4. Segurança por Obscuridade
5. Frameworks são Opcionais
6. Detetar e Registrar
7. Não Confiar Inicialmente
8. Seguro Por Defeito
9. Manter Simples



Misuse & Abuse Cases

Requisito: o que o sistema deve fazer ou não fazer e quem interage com o sistema (atores) — descrever como é que o ambiente circundante foi alterado como um resultado do sistema

Segurança: propriedade do software

- Os documentos de requisitos não devem ser só uma lista de funcionalidades!
- As vulnerabilidades estão em áreas de demasiada funcionalidade!

Use Case: ator + pré-condições + fluxo principal que descreve o cenário principal

Misuse/Abuse Case: cenário de um use case no qual um ator compromete o sistema — fluxo de eventos, mas com uso malicioso, que define o dano causado ao sistema

Como desenhar? Questionar as ameaças do sistema e focar-se naquilo que o ator PODE fazer em vez do que VAI fazer!

Misuse: não intencional — "se for repetido intencionalmente, seria abuse"
Abuse: intencional — o ator procura aktivamente por vulnerabilidades

Requisito de Segurança: forma generalizada de misuse & abuse cases
ANTI-REQUISITOS

Avaliação do Risco de Segurança

Porquê estudar risco? Muitos resultados possíveis, mas nem todos prováveis

Risco: $P(\text{ocorrência}) \times \text{impacto}$ - o risco associado com um evento é a probabilidade desse evento ocorrer vezes a magnitude do impacto do evento

Risco de Segurança: $P(\text{exploit}) \times \text{valor do ativo}$

$\alpha P(\text{vulnerabilidade})$ [] → probabilidade de um exploit ocorrer no sistema aumentada pelo número de vulnerabilidades

Ativo: recurso (tangível ou intangível) do sistema que tem valor em confidencialidade, integridade e disponibilidade

- Ativos
- [1] Específicos do Domínio
 - [2] Independentes do Domínio
 - [3] Propriedades Intangíveis

- Os ativos existem em todos os sistemas de software em produção, devendo ser identificados nos estágios de requisitos e design
- A análise de riscos ajuda a refinar e descobrir novos abuse cases!

Abuse & Misuse Cases: ênfase no domínio - "what if?"

Avaliação de Risco: ênfase nos riscos - "what might?"

Protection Poker: quantificar os riscos para os priorizar - "Story points"

- 1. Identificar ativos
- 2. Calibrar valores dos ativos
- 3. Calibrar facilidade de ataque

Segurança Aplicacional

DevOps: combinação do desenvolvimento de software e operações para aumentar a capacidade de uma organização em entregar aplicações e serviços a alta vel.

SecDevOps: processo que visa colocar a segurança como primário passo no ciclo de vida de desenvolvimento de software

Objetivos:

1. minimizar riscos de desenvolvimento, nem afasay o processo
2. apoiar as equipas de segurança, nem revisões/aprovações manuais

Teste de Segurança Aplicacional: processo de tornar aplicações mais resistentes a ameaças de segurança ao identificar fráquezas e vulnerabilidades no código

Bugs
Design: a máquina de estados conceptual não cumpre os objetivos impostos
Funcionalidade: o código tem más transições mas só entre estados válidos
Implementação: o código introduz novos estados não representados

Segurança "Shift Left": esforço de uma equipa de DevOps em garantir segurança aplicacional nos estágios iniciais do ciclo de vida de desenvolvimento de SW

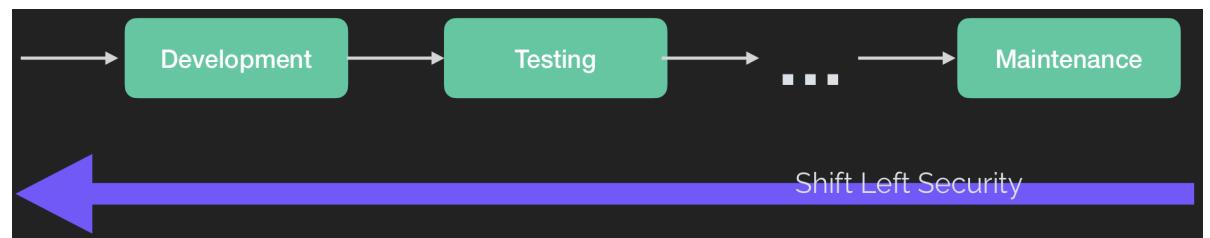
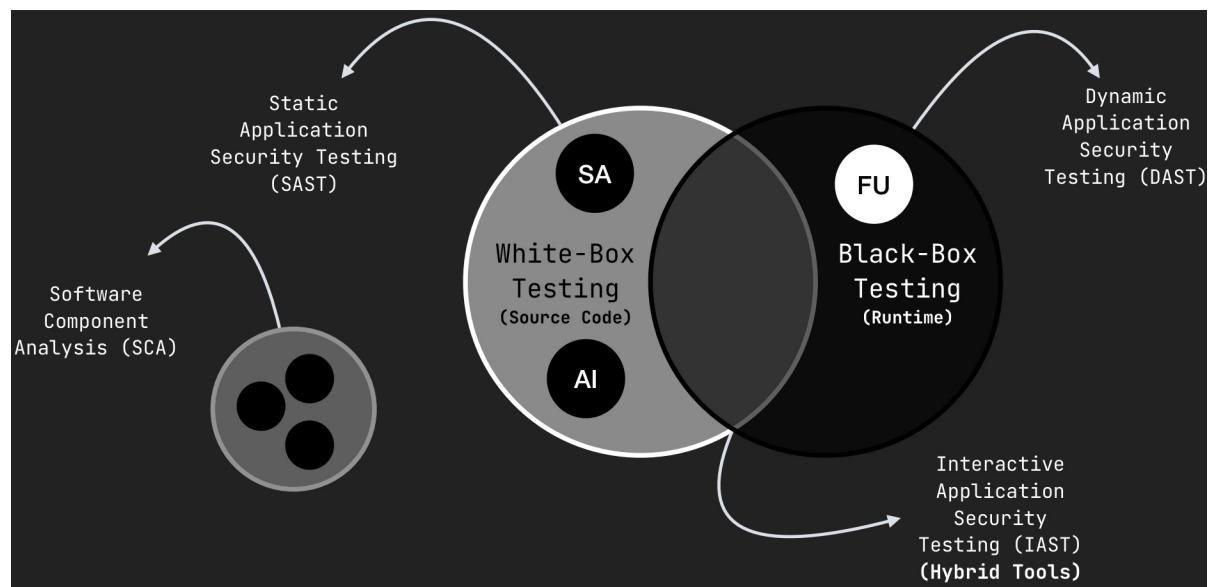
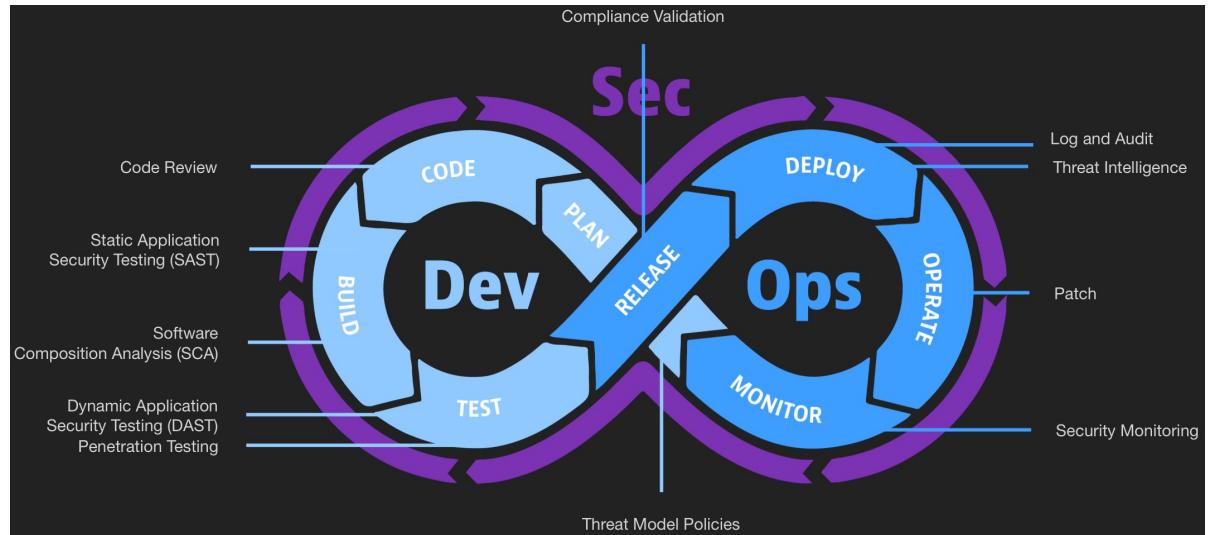
Análise Estática: "scan" do código-fonte sem o executar - testa se uma certa propriedade se mantém e procura onde é violada - dependente da linguagem

Análise Dinâmica: analisa o comportamento de um programa ao executar o seu código - teste "black-box" ao atacar uma aplicação em execução

Puzzing: encontrar bugs num programa ao alimentá-lo com dados aleatórios, corrompidos ou inesperados - inputs aleatórios exploram o espaço de estados

1. Baseado em Mutações: muitas aleatoriamente casos de teste
2. Baseado em Geração: gerar casos de teste com base no input
3. Baseado em Cobertura: guiar por medições de cobertura de código

SCA: metodologia para gerir componentes "open-source"



Planeamento de Testes

1. Analisar: Enumerar; Priorizar; Discutir
2. Gerir
3. Mitigar Riscos
4. Registrar Riscos

Planeamento de Testes "Top-Down": 
REQUISITOS → DESIGN
Objectivos → Riscos → Indicadores → Testes
foco nas vulnerabilidades

Objectivos → Riscos: os riscos de alto-nível mapiam-se em 1+ objetivos

Riscos → Indicadores: como saber se um risco se vai tornar num problema?

Indicadores → Testes: como garantir que um indicador é satisfeito ou evitado?

- :- vinculado a objetivos específicos
- :- incompleto dentro de categorias

Planeamento de Testes "Bottom-Up":

1. Escrever testes
2. Agrupar os testes em categorias
3. Rever as categorias como um grupo
4. Adicionar mais testes a cada categoria

- :- liberdade para escrever os melhores testes imediatamente
- :- fácil de faltar/escapar coisas

Testes de Penetração

Teste de Penetração: Tentar explorar tanto quanto possível - cenário real

A11&CK: Taxonomia de táticas e técnicas para conhecimento de testes de penetração de propósito geral

CAPEC: "Common Attack Pattern Enumeration and Classification" - dicionário de padrões de ataque organizado por mecanismos e domínios

A11&CK

1. Pre-A11&CK: construir capacidades e pesquisa inicial
2. Acesso Inicial: tentar acesso à rede
 - ↳ compromisso guiado; ações de hardware; "spearphishing"
3. Descoberta: compreender o ambiente
 - ↳ Scan de serviços e redes; descoberta de contas
4. Escalamento de Privilégios: obter permissões mais elevadas
 - ↳ injecção de processos; return / ret2gic
5. Evasão a Defesas: evadir detecções
6. Acesso a Credenciais: roubar nomes de contas e palavras-passe
 - ↳ forsa bruta; "dump" de credenciais; contas válidas
7. Recolha: obter dados de interesse do objetivo
8. Execução: correr código malicioso
 - ↳ interface de linha de comandos; execução de serviços
9. Persistência: manter acessos
 - ↳ "bootkit"; tarefa agendada; criar conta; componente de firmware
10. Movimento Lateral: mover pelo ambiente - "pass-the-hash"; cópia de ficheiros
11. Comando e Controle: comunicar com sistemas comprometidos para os controlar
12. Exfiltração: roubar dados
13. Impacto: manipular, interromper ou destruir os sistemas e dados

Modelação de Ameaças

Spoofing - viola autenticação

Tampering - viola integridade

Repudiation - viola integridade do histórico

Information Disclosure - viola confidencialidade

Denial of Service - viola disponibilidade

Elevation of Privilege - viola autorização

→ Discutir riscos de segurança quando a arquitetura estiver quase definida!

Modelação de Ameaças: ferramenta de análise de risco de arquitetura

1. Definir um diagrama de fluxo de dados

a. interatores externos

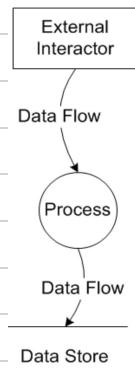
b. processos

c. armazenamento de dados

d. fluxo de dados

2. Definir fronteiras de confiança

3. Mapear STRIDE para cada elemento e relação



• Os modelos de ameaças não especificamente sobre modelar segurança e não arquitetura - baseiam-se em fluxos de dados e zoneiam / caracterizam os dados que vão ser transferidos de uma parte do sistema para outra

"Distrustful Decomposition": quando os programas não executados, o código tem permissões, mas os programas têm funções totalmente separadas que não se devem impactarumas às outras - o sistema deve ser decomposto em múltiplos processos com permissões separadas em que cada processo desconfia do outro e comunicam por Comunicação Inter-Procесс (IPC)

Programação Defensiva

• Uma pequena alteração no código pode originar uma grande mudança na análise de risco!

1. É muito fácil escrever código inseguro
2. Devem conhecer-se as APIs
3. A manutenção também conta

• "A complexidade é inimiga da segurança!"

- 1. Estrutural
- 2. Cognitiva
- 3. Inputs

• Compreender a história do que é comum e possível

Validação de Input: blacklist + whitelist

Sanitização de Input: converter input para não ser interpretado como código

- O sistema deve lidar com exceções — manutenção e complexidade
- A composição é preferível a herança! Valores de tipos imutáveis são preferíveis
- A concorrência é um risco! As variáveis globais estáticas são perigosas!
- Um objeto sanitizado não é confiável!
- A organização da memória é importante! O tamanho dos caracteres é importante!
- Os dados flutuantes não devem permanecer em memória mais do que necessário!
- As variáveis de ambiente não devem ser confiáveis! UTF8/U1F16!
- As chamadas nativas devem ser tratadas como entidades externas! HTTP
- Os ficheiros de configuração são importantes!

Inspeções de Código

Testes "White-Box": com conhecimento íntimo das especificações e do design

Inspeções de Código: revisões técnicas de código com a segurança em mente

Teste: técnica de procura por falhas

Inspeção: técnica de procura por falhas

O que testar? mitigações; código; design; unitários; integração; vulnerabilidades

Quem testa? autor; pessoas com leitura, mas objetividade; experientes em segurança

O que procurar? complexidade; erros; oportunidades

O que priorizar? o que provavelmente tem vulnerabilidades!

Cobertura de Código: o que não foi testado

⇒ revela o esquecido; simples de executar

⇒ não significa bem testado

Análise Estática: o que as ferramentas digem ser vulnerável

⇒ fácil; rápido; conhecimento; contexto

⇒ falsos positivos; orientada para código; não específico do domínio

Precisão: o que o histórico diga ser vulnerável

Fraguças: onde as fraguças podem resultar em vulnerabilidades

Componentes de Engenharia de Software Seguro

Política: afirmação do que é e não é permitido

Mecanismo: procedimento, ferramenta ou método de forçar uma política

Mecanismo de Segurança: implementa funções que ajudam a prevenir, detectar, responder e recuperar de ataques de segurança
↓
CRIPTOGRAFIA

Ex: autenticação; controlo de acessos; confidencialidade; integridade; não-repúcio

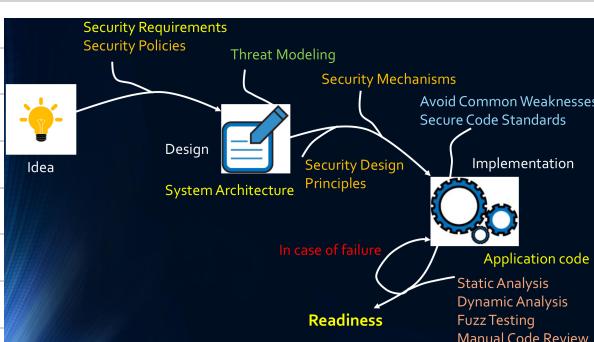
Serviço de Segurança: disponibilização de funções de segurança

Específico: cifra; armadura; controlo de acessos; integridade; autenticação; ...

Pervasivo: funcionalidade confiável; etiquetagem de segurança; deteção de eventos; ...

Criptografia: prática e estudo de técnicas para acesso seguro, comunicações e armazenamento, na presença de adversários

Foga de Segurança Criptográfica: estimativa do número de operações necessárias pelo algoritmo mais bem-conhecido para quebrar a criptografia no processo concreto considerado



Segurança de Software

Qualidade do Software: fiabilidade - frequência de bugs durante o uso normal
Segurança de Software: CIA - inputs improváveis durante o uso normal

→ Adversário; Ataque; Mitigação; Risco; Política; Ativo; Ameaça; Vulnerabilidade

Common Weakness Enumeration (CWE): tipos; linguagens; exemplos - Taxonomia
Common Vulnerability Exposure (CVE): vulnerabilidades em casos reais

CAPEC: ataques aplicacionais - árvore/hierarquia de mecanismos de ataque
ATT&CK: sistemas de IT - táticas, técnicas e procedimentos de atacantes

→ Memória; Input; Concorrência; Acessos; Web; Criptografia

↓ Os programas lidam com dados de inputs e produzem outputs!
NÃO CONFIÁVEL

↓ Todo o input tem de ser verificado!

1. Canonicalização: redução do input à sua forma equivalente mais simples conhecida
2. Normalização: conversão para uma forma padrão (não necessariamente a mais simples)
3. Sanitização: processo de garantir que os dados não violam a política de segurança
3. Validação: processo de verificação de inputs para garantir pertença ao domínio REGEX

↓ "Race Condition": quando dois fluxos de execução concorrentes acedem a um recurso partilhado e o resultado depende da ordem de acesso

↓ "Time-of-Check to Time-of-Use (TOCTOU)": quando existe uma janela temporal entre a verificação de um recurso para uso e o seu uso efetivo - indissociável

A deserialização de objetos/dados não confiáveis pode levar a "exploits"!
→ constrói novos objetos nem invoca qualquer construtor

XML External Entity (XXE): XML é utilizado para transportar dados para uma aplicação web - num documento XML, é possível incluir um DTD externo e definir entidades em ficheiros externos ou URLs

Server-Side Request Forgery (SSRF): quando uma aplicação web permite aos utilizadores fazer pedidos a URLs arbitrários, no código do servidor

Boas Práticas de Programação Segura:

1. Validar input
2. Atentar aos avisos da compilador
3. Projatar pelas políticas de segurança
4. Manter simples
5. Negar por defeito
6. Adherir ao princípio de privilégio mínimo
7. Sanitizar dados enviados para outros sistemas
8. Praticar cipher em profundidade
9. Usar técnicas efetivas de garantia de qualidade
10. Adoptar um padrão de computação segura
11. Definir requisitos de segurança
12. Modelar ameaças



Autenticação

Entidade: ator que necessita de uma distinção entre instâncias diferentes

Identidade: representação única de uma entidade

Policy Enforcement Point (PEP): aplica a política de acessos para o recurso

Policy Decision Point (PDP): autentica o utilizador e consulta as políticas

Autenticação: vínculo de uma identidade a uma entidade

1. Registo: recolha de informação sobre a entidade a autenticar

2. Identificação: interação com o sistema para especificar o identificador do utilizador

3. Verificação: validação das alegações

Informação de Autenticação: informação possível para prova de identidade

Informação Complementar: informação usada para validar a informação de autenticação

Função de Complementação: $A \rightarrow C$ - hash

Função de Autenticação: $A \times C \rightarrow 1/F$ - verifica a identidade

Função de Seleção: altera a informação de autenticação

Fatores de Autenticação:

- 1. Algo que o utilizador sabe
- 2. Algo que o utilizador tem
- 3. Algo que o utilizador é
- 4. Algo que o utilizador faz

Salt: valor aleatório para ser combinado com uma palavra-passe que previne que palavras-passe duplicadas sejam visíveis no ficheiro de palavras-passe para aumentar a dificuldade de ataques de dicionário

Pimenta: valor aleatório comum a todos os utilizadores

Ataque de Dicionário: tentar cada palavra e variantes óbvias num grande dicionário contra as hashes no ficheiro de palavras-passe - fácil se salt conhecido

Ataque de "Rainbow Table": pré-computar uma tabela de valores de hash para todos os sais possíveis

Boas Práticas: educação; geração; verificação reativa/proativa

Fórmula de Anderson: $P > \frac{1 \times G}{N}$

tempo → teste por tempo
Universe → Universo

Smartcard: pode gerar uma palavra-passe dinâmica, usar desafio-resposta e PIN

Paranoid Authenticated Connection Establishment (PACE): garante que o chip do cartão não pode ser lido sem controlo de acesso explícito — PIN ou código

"One Time Paranoid" (OTP): palavra-passe que só é usada uma vez — necessita de uma sincronização inicial com o autenticador HOTP TOIP

Autenticação Biométrica: baseada em características físicas estáticas ou dinâmicas

1. Registo: extração e armazenamento das características biométricas
2. Verificação: correspondência entre a característica fornecida e a armazenada
3. Identificação: comparação e fornecimento do identificador

Autenticação Remota: protocolo desafio-resposta — $f(n, h(p))$

Ataques:

1. Cliente: mascarar-se como utilizador legítimo — limites; palavras-passe fortes
2. Anfíbio: tentativa de obter o ficheiro de palavras-passe — hash; sal
3. Eavesdropping: observação da comunicação e repetição — negredos; desafios; MFA
4. Repetição: reenvio de resposta — desafio-resposta; OTP
5. Caído de Troia: mascarar-se de aplicação autêntica — ambiente seguro
6. Denial-of-Service: tentativa de desativação de serviço — rapidez; limites

Controlo de Acessos

Controlo de Acessos: especificação de quem ou o que pode ter acesso a um recurso específico do sistema e o tipo de acesso permitido (leitura, escrita,...)

Autenticação: identificação dos utilizadores ou outras entidades no sistema

Autorização: concessão de permissões a uma entidade do sistema para aceder a um recurso e como — quem é confiável para executar a operação

Auditoria: exame das actividades dos utilizadores para testar e verificar políticas, adequação de acessos e falhas de segurança

Administração de Segurança: manutenção da base de dados de autorização que especifica que tipo de acesso a quais recursos cada utilizador é permitido

Função de Autorização: determina o acesso a recursos — quem pode aceder e quem

Sujeito: entidade autenticada que pode aceder a objetos

1. Proprietário

2. Grupo

3. Mundo

Objeto: recurso/ativo de acesso controlado

Direitos de Acesso: forma como um sujeito acede a um objeto — operações permitidas

Controlo de Acessos em UNIX: cada utilizador tem um identificador único e é membro de um grupo primário — 12 bits de proteção por objeto



Controlo de Acessos Discricionário: para cada objeto protegido, existe uma lista de sujeitos (permitidos ou não) que afirma que operações podem ser realizadas

Matriç de Acessos

→ espessa
de composta

LISITAS DE CONTROLO DE ACESOS

OBJETOS

SUJEITOS CAPACIDADES

DIREITOS DE ACESSO

Controlo de Acessos Mandatório: decide baseado na responsabilidade de segurança dos recursos e a "clearance" de segurança dos sujeitos

- Há uma autoridade central que manda a política
- A informação pertence à autoridade central e não aos utilizadores
- É atribuída uma etiqueta de responsabilidade aos sujeitos e objetos

Segurança Multi-Nível: sistema hierárquico de etiquetas - autoridade central

↓
ETIQUETA

1. Confidencialidade/Sensibilidade - nível de confiança da autoridade no sujeito
2. Comportamentos - baseados na necessidade de conhecer/saber

$L(X) \subseteq L(Y)$: $L(X)$ não é mais restritivo do que $L(Y)$

$$\hookrightarrow L_1 \subseteq L_2 \leftrightarrow S_1 \leq S_2 \wedge C_1 \leq C_2$$

- S pode ler O se e só se $L(O) \subseteq L(S)$ - "no read up"
→ S pode escrever O se e só se $L(S) \subseteq L(O)$ - "no write down"

Controlo de Acessos Baseado em Funções: os acessos são concedidos com base no papel do sujeito - há um mapeamento entre sujeitos e papéis que têm

Controlo de Acessos Baseado em Atributos: o acesso é concedido com base nos atributos atribuídos aos sujeitos e aos objetos