Principios fundamentais

* Economy of mechanism
* Fail-safe defaults
* Complete mediation
* Open design
* Separation of privilege
* Least privilege
* Least common mechanism
* Psychological acceptability
* Work factor
* Compromisse recording

Economia nos mecanismos

* Keep it simple:
  + Um sistema deve ter apenas as funcionalidades (ex. serviços a correr) necessários
  + De todos os mecanismos de segurança equivalentes
  + Dvem ser adotados os mais simples
* Facilita a implementação, usabilidade, validação, etc

Protecao por omissão (fail safe defaults)

* A configuração de qualquer sistema deve, por omissão (default), impor um nível de protecao conservador
* Ex. um novo utilizador deve, por omissão, ter o mínimo de permissões
* “fail closed”: por oposição a reverter para uma posição permissiva

Desenho aberto (open design)

* A arquitectura de segurança e os detalhes de funcionamento de um sistema devem ser públicos:
  + Não deve basear-se em segurança de “security through obscurity”
  + Isto é, assumir que o atacante sabe tudo sobre o sistema
* Os segredos são parâmetros do sistema, que podem ser alterados:
  + Chaves criptográficas, passwords, etc.
* Racional:
  + Permite o escrutínio: mais provável sabermos que estamos vulneráveis
  + Podemos mudar os segredos sem mudar o sistema

Defesa em profundidade

* Já vimos antes: todos os mecanismos de segurança podem falhar
* Não podemos depositar toda a nossa confiança num so mecanismo:
  + Ex. blindar perímetro e assumir que noa existem ameacas internas
* Inerente a qualquer sistema que faca qualquer coisa de útil para o exterior

Privilegio mínimo (least privilege)

* Cada utilizador, compartimento, programa, etc
  + Deve ter apenas os privilégios/permissões essenciais para desempenhar a sua função
  + Racional: need-to-know
    - Contrariar este prinicipio amplifica sem necessidade o impacto de uma brecha local na segurança
  + Ex: correr todos os serviços como “root”

Separação de privilégios (separation of privilege)

* As funcionalidades/utilização de recursos devem ser compartimentadas:
  + Cada compartimento deve estar isolado dos outros
  + Todos os compartimentos devem encarar os restantes como estando num outro domínio de confiança
* Essencial combinar com defesa em profundidade e privilegio mínimo:
  + Comprometer um compartimento tem consequencias localizadas
  + Um compartimento tem apenas acesso aos recursos de que necessita

Mediação completa (complete mediation) (mesmo cena q controlo de acessos)

* Um sistema gere recursos: ficheiros, dispositivo hardware, memoria, etc
* Mediação completa implica:
  + Para todos os recursos definir uma politica de protecao
  + Todos os acessos a todos os recursos são validados relativamente a essa politica
* Ex: memoria virtual
  + Todos os processos acedem a um espaco de memoria virtual
  + Todos os acessos a memoria virtual são mediados pelo sistema operativo
* Veremos mais a frente formas diversas de controlo de acessos = mediação

Controlo de acessos

* Família de mecanismos que aplica os seguintes prinicipios:
  + Privilegio mínimo: atribuir privilégios estritamente necessários as entidades que interagem com o sistema
  + Mediação total: garantir que todos os acessos a um recurso são efetuados por entidades com privilégios adequados
* Quando tratamos de controlo de acessos, tratamos de:
  + Ator (entidade que realiza a acao)
  + Recurso (sobre o qual se realiza a acao)
  + Operação (tipo de acao realizada)
* Combinação de recurso/operação chama-se permissão

Matriz de acessos

* Descreve todos os possíveis acessos
  + Ator, recurso, operação
  + Ex. ler, escrever, executar
* Vantagem: clareza, eficiência
* Problema: tamanho

Lista de controlo de acessos (ACL)

* Para cada recurso, as permissões dos atores sobre esses recursos (omissão = ausência de permissão, intuição = lista de convidados)
* Vantagens:
  + Faz-nos pensar sobre como proteger cada recurso individualmente => armazenamento associado ao próprio recurso
  + Permite garantir facilmente que um recurso apenas pode ser acedido por um numero limitado de atores (isolamento)
* Desvantagens:
  + Não e possível determinar as permissões que um ator tem ser ver todos os recursos (ex. remover um ator)
  + Não há agregação de perfis de permissões

Lista de permissões (capabilities)

* Para cada ator, as operações que pode realizar sobre cada recurso (omissão = ausência de permissão, intuição = passe de transportes)
* Vantagens:
  + Faz-nos pensar sobre como lidar com cada ator individualmente => armazenamento associado ao ator
  + Permite garantir facilmente que um ator apenas acede a um numero limitado de recursos
* Desvantagens:
  + Não e possível determinar todos os atores que podem aceder a um recurso sem percorrer todos os atores
  + Não há agregação de perfis de permissões

Role based access control

* Modelo dos dois tipos de relações que permitem separar a gestão de recursos da gestão de atores:
  + Ligação de perfis (roles) a conjuntos de permissões (agregação, menos espaco)
  + Ligação de atores a perfis
* As relações entre perfis e permissões são geralmente muito estáveis:
  + São administrados por quem gere recursos => semelhante a ACL
* Os perfis dos atores podem ser mais dinâmicos:
  + Diagram

    Description automatically generatedSão administrados por quem gere atores (ex. utilizadores) => semelhante a lista de permissões

Attribute-based access control

* O RBAC é um caso particular de ABAC:
  + Atores e recursos teem atributos associados:
  + Matriz de acesoss descreve permissões com base nos atributos:
    - Para aceder a recurso com atributo A o ator deve possuir atributo B
  + Permite politicas mais expressivas, como por exemplo ter em conta o contexto gerografico e temporal, ou sistema hierarquicos

Sistemas operativos

* Interface entre utilizadores e hardware de um computador:
  + Gestão do acesso a recursos por parte das aplicações:
    - Armazenamento, processador, memoria, IO, rede, etc
    - Partilha destes recursos por vários utilizadores e aplicações
    - Software que trata operações de baixo nível e oferecem abstracoes convenientes para o desenvolvimento de aplicações
  + Muito complexos e gerem aspetos críticos para a segurança
* Aspetos de segurança:
  + Múltiplos utilizadores com diferentes níveis de acesso:
    - Administradores, utilizidores frequentes, utilizadores esporádicos, etc
    - Diferentes necessidades e privilégios relativamente a recursos a utilizar
    - O SO tem de garantir que estes requisitos são cumpridos e, ao mesmo tempo, garantir que não existem comportamentos abusivos
    - Os utilizadores são sempre potenciais ameacas
    - Os recursos são sempre ativos a proteger
  + multiplas aplicações/serviços a executar em simultâneo:
    - as aplicações são também potenciais ameacas
    - devem estar protegidas de interferência de outras => isolamento
    - esta protecao aplica-se mesmo qunado não estão a correr:
      * a informação que armazenam como estado esta geralmentos em recursos partilhados
  + SO modernos visam garantir:
    - Isolamento virtual entre utilizadores, aplicações e processos
    - Apesar de estarem a partilhar recursos disponíveis no sistema
    - Através de mecanismos de acesso mediado
  + Administrar um sistema operativo => configurar estes mecanismos
    - Devem aplicar-se os princípios fundamentais: privilegio mínimo , separação de privilégios, etc => boas praticas

Kernel

* Parte do SO que desempenha operações mais criticas
  + Processador está em kernel mode e todas as operações são permitidas
  + Os processadores permitem geralmente definir vários níveis de privilegio
  + Em muitos casos usam-se apenas dois: kernel mode e user mode
  + Código em user mode não tem acesso direto aos recursos do sistema
  + Qualquer troca de informação entre os dois níveis faz parte de uma superfície de ataque
* Esta protegido dos processos em user mode:
  + Tem espaco de memoria gerido de forma independente
  + Processador garante que apenas código que corre em modo kernel pode executar um conjunto de instruções privilegiadas
  + Qualquer processo em user mode tem de aceder aos recursos do sistema usando system calls
  + Parte do código das system calls executa em kernel mode

Confinamento

* Pontos de entrada em system calls são críticos:
  + Para causar danos, um processo em user mode tem de o fazer através de uma system cal
  + Isto implica implementar sistemas de monitorização e controlo de chamadas ao sistema
  + Reference monitor:
    - Sempre presente (se terminar, processos monitorizados tb terminam)
    - Tem de ser simples para poder ser analisado e validado mais facilmente que o sistema todo

System calls

* Controlo de processos
* Acesso a ficheiros
* Gestão de dispositivos
* Configuração do sistema
* Comunicacoes
* Protecao
* Para entrar num modo de funcionamento com mais privilégios, o código user mode deve:
  + Preparar argumentos e identificar um ponto permitido para acesso a kernel mode
  + Executar uma instrução especial que passa o controlo para kernel
* Existe um numero limitado deste tipo de pontos de entrada:
  + Registos específicos para parâmetros, que são tipicamente apontadores para memoria de processos em user mode
  + O processamento desta informação é da total responsabilidade do kernel
* Um numero limitado de pontos de entrada => superfície de ataque bem definida

Processos

* Kernel defina a nocao de processo: uma instancia de um programa que esta a executar
* Programas comecam por estar guardados em armazenamento não volátil
* Para serem executado, teem de ser carregados para a memoria e receber um identificador como processo
* cada processo deve executar num contexto em que tem acesso a um conjunto de recursos, que devem estar disponíveis independentemente dos outros processos
* a fronteira entre processos é uma fronteira de confiança: os processos teem de estar confinados/isolados entre si
* o kernel garante acesso a recursos:
  + atribui uma fracao razoável de tempo de processador a cada processo
  + atribui um espaco de memoria sobre o qual o processo pode trabalhar
  + concede acesso a outros recursos através de system calls
* nos SO multi-utilizador, existe um conjunto de processos que interagem com o utilizador humano:
  + quando o utilizador lanca uma aplicação, o SO ve um dos processos que interage com o user a criar um novo processos => forking
  + os SO gerem uma hierarquia de processos, em que tipicamente os descendentes herdam privilégios dos seus criadores
* muitas vezes, necessário comunicar entre processos => system calls:
  + através de sistema de ficheiros
  + memoria partilhada
  + mensagens assíncronas
  + mensagens síncronas
* daemons/services:
  + processos não visíveis pelo utilizador
  + geralmente arracandos antes dos processos que interagem com o user
  + executam tipicamente com privilégios superiores aos dos utilizadores e sobrevivem as suas sessões

modelo de confiança

* confiança depositada nos processos lancados é indutiva:
  + código armazenado no computador apos uma instalação é “confiável”
  + processo de boot utiliza este código para colocar kernel em memoria e passar-lhe o controlo, criando um estado “confiável”
  + kernel lanca processos com permissões que garantem que nenhum novo processo pode alterar o estado de confiança
  + processos de hibernação preservm estado de confiança
  + administradores podem alterar software instalado no sistema e o sistema de permissões, mas garantem que qualquer atualizacao preserva o estado de confiança
* o que significa “confiável”
  + sistema faz exatamente aquilo que foi especificado
  + ex. não transmite informação para o exterior sem permissão

modelo de ameacas

* ataques de todos os níveis do boot
  + BIOS corrompida
  + Ficheiros de hibernação corrompidos/roubados
  + Bootloader corrompido
  + Cold boot attacks
* Vulnerabilidade que afetam a implementação dos mecanismos de arranque fazem um bypass completo ao modelo de confiança (o ataque é a ancora)

Medidas de mitigação

* Maioria dos problemas ocorre atraves de erros de administração
* Monitorização é uma forma de mitigação comum para detetar quebras neste modelo:
  + Logs de eventos permitem detetar comportamentos suspeitos, como o crash repetido de um processo que esta a tentar explorar uma vulnerabilidade
  + Aplicações de monitorização de processos permitem visualizar os processos que estão a executar, os recursos que utilizam e ficheiros de código que originam
* A mediação de instalação/execução de código com base em assinaturas digitais permanece um entrave a introdução de código malicioso num sistema
* Mecanismos para garantir isolamento entre processos
  + Impedem que um utilizador com pouco privilégios possa utilizar o sistema para alem do que lhe é permitido
  + Impedem que um processo que tenha uma vulnerabilidade não abra uma porta para corromper o sistema todo (defesa em profundidade)

Memoria

* Regra fundamental:
  + Um processo não pode aceder ao espaco de memoria de outro processo
  + Confidencialidade, integridade e controlo de fluxo do kernel tem de ser protegida de todos os processos que executam em user mode
* Como se garante:
  + Em run-time os acessos são mediados por um conjunto de mecanismos de hardware e software geridos pelo kernel
  + As partes de memoria virtual que estão em disco pode ser alvos de ataque off-line se um adversário puder aceder a essa informação => disco

Tradução de endereços

* Tradução de endereços necessária a implementação de mecanismos de memoria virtual cumpre dois propósitos:
  + Isolamento: cada processo acede a uma zona da memoria que não existe na realidade, e q da visão /acesso limitados aos recursos
  + Eficiência: esconde mecanismos de otimizacao (caching, paging, etc)
* Memoria virtual esta dividida em paginas pequenas
* Sistema tem de armazenar para cada pagina a sua localização física
* Page-table: arvore esparsa com informação nas folhas
* Aceder a uma page-table é penalizador
* Translation lookaside buffer (TLB):
  + cache de paginas traduzidas recentemente
  + informação para controlo de acessos em cada pagina
  + read/write/execute
* como lidar com chamadas de sistemas?
  + Kernel mapping
    - No espaco de memoria de cada processo em user mode, existe um subespaço que de facto é memoria do kernel (para que estejas a ser gerida no mesmo contexto, e quando há sys cal podemos passar para kernel mode sem ter um overhead enorme)
    - Quando se muda a memoria do processo de user mode, as partes da memoria que estão a ser utilizadas em kernel mode são “mais ou menos” constantes

Defesa em profundidade

* Que permissões deve ter o kernel?
  + Todas? NAOOOOO
* Defesa em profundidade:
  + Nem o kernel deve ser capaz de violar a regra W^X
  + Impedir o kernel de escrever em partes da memoria do user é uma forma de impedir fugas de informação/código malicioso no caso de kernel corrompido

Sistemas de ficheiros

* Atores: utilizadores, processos
* Recursos: ficheiros e pastas
* Acoes/acessos:
  + r/w/x para ficheiros: read, write, execute
  + r/w/x para pastas: listar conteúdo, criar conteúdo adicional, entrar na pasta
  + cada user pertence a um grupo: permite uma forma de RBAC (roll back access control)
* cada recurso tem um owner e um grupo
  + owner (ACL com uma única entrada) (access control listing)
  + membros do grupo associado ao recurso (RBAC rígido)
  + todos os outros utilizadores (RBAC rígido)
* superuser:
  + antigamente era “root”
  + hoje em dia “sudo”
* alteração de permissões:
  + tudo e permitido ao superuser
  + permissões de um recurso podem ser alteradas pelo owner
  + owner de um recurso pode ser alterada pelo superuser “SO!!!”
  + grupo de um recurso pode ser alterado por owner e superuser
* owner altera permissões => discretionary access control
* apenas administrador => mandatory access control
* permissoes de processos:
  + cada processo tem um “effective user id”
    - determina as permissões que o processo tem
    - em geral, é o user id que lancou o processo
    - existem excecoes (quando mudo password)
* como funciona o login:
  + um processo é lancado com privilégios máximos
  + esse processo autentica o user
  + alterar os próprios uid e gid para os associados ao user
  + lanca o processo de Shell
  + login executa “drop privileges”
* o reverso, “elevate privileges”, deve ser impossível (e o passwd?)
* o bit “setuid” associado a um ficheiro:
  + permite fixar o utilizador associado a um processo ao owner do executável (e não o user que executa)
* tudo é um ficheiro:

exemplo de utilização: Android

* problema: restringir acesso de aplicações a recursos
* solução: cada aplicação ter o sseu próprio utilizador
* problema: múltiplos utilizadores?
  + Solução ad-hoc: u1\_a23

Privilégios de processos

* Quando executamos processo, tipicamente executa com UID do utilizador que lancou
* Alguns processos são executados com UID do owner do ficheiro executável
* Um processo tem 3 UID:
  + Effective user id (EUID): determina as permissoes
  + Real user id (RUID): utilizador que lancou o processo
  + Saved user id (SUID): utilizado em transacoes, lembra o anterior
* O utilizador root pode usar a system call “setuid(x)” para alterar os 3 UIDs para x
* Utilizadores comuns so podem mudar EUID para RUID ou SUID
* Isto permite um processo reduzir os próprios privilégios:
  + Quando o Apache cria um processo para atender um utilizador reduz os privilégios do processo
* System calls “seteuid(x)” altera apenas o EUID
* Utilização típica:
  + Baixar privilégios => executar código => restaurar privilégios
* Conclusao:
  + Sistema de controlo de acessos é basicamente uma implementação de Acess Control Lists com algum batching
  + Vantagem => simples e funciona na pratica
  + Desvantagem => pouco robusto e pouco flexível
    - Uma falha num processo tipo passwd ou ssh tem consequências catastróficas
    - Root utilizado para muita coisa => erros de administração

Confinamento (preludio)

Executar código não confiável

* Executar código não confiável numa plataforma confiavel:
  + Código proveniente de fontes externa, tipo internet
  + Código antigo que sabemos não estar a altura das exigências atuais
  + Analise forense de malware, honeypots (codigo vulneravel que criamos de propósito para estar misturado com sistemas a serio de propósito para o atacante ir la)
* Objetivo: se “portar mal” => destruir aquilo

Confinamento (air gap)

* Solução: garantir que o código potencialmente maliciosa não pode afetar o resto do sistema
* Pode ser implementado a muitos níveis, começando pelo próprio hardware
* Quando o confinamento é efetuado ao nível do HW => airgap
* Desvantagem: difícil de gerir

Confinamento (maquinas virtuais)

* Hypervisors permitem partilhar hardware:
  + Garantem que as acoes em SO1 não afetam S02, e vice-versa

Confinamento (SFI + SCI)

* Software fault isolation: nome generic par aisolamento de processos que partilham o memso espaco de endereçamento
* System call interposition: nome generico para mediacao de todas as system calls, concentrando os pontos de acesso a operações privilegiadas num numero pequeno de pontos que podem ser monitorizados
* Nos sistemas de ficheiros há dois mecanismos SFI/SCI:
  + Isolamento de memoria: virtualizar o espaco de endereçamento e monitorizar os acessos nos mecanismos de tradução de endereços
  + Separação kernel vs userland: fornecer um numero limitado de system calls ao user e mapeando sub-sistemas nos mecanismos de manipulação de ficheiros

Confinamento (sandboxing)

* Confinamento dentro de uma aplicação
* Ex. browsers:
  + Internamente criam um ambiente de execução isolado para código proveniente de fontes externas

Confinamento: implementação

* Componente central é chamado reference monitor
* Faz mediação de todos os pedidos de acesso aos recursos
  + Implemente politica de protecao de recursos/isolamento
* Tem de ser sempre invocado => todas as aplicações são mediadas
* Tem de ser omnipresente
  + Qunado morre reference monitor, morre tudo
* Tem de ser simples o suficiente para poder ser analisado

Jails

* Pasta atual passa a ser raiz do filesystem
* System calls que acedem a ficheiros são interceptadas e as paths recebem um prefixo correspondente à pasta actual
* Consequências => as aplicações não conseguem aceder a recursos fora dessa pasta
* É critico que o user dentro da jail não se consiga tornar root
* Caso contrario, existem muitas maneira de fugir da jail:
  + Criar dispositivo para aceder ao disco em bruto
  + Enviar sinais a processos que não estão na jail
  + Reiniciar sistema