Índice

Conceitos Base

Criptografia: Técnicas que procuram tornar possível a comunicação secreta entre duas partes sobre um canal aberto.

Cripto-análise: Tenta gorar os objetivos da criptografia.

Criptologia: Criptografia + Criptoanálise

Modelo de comunicação

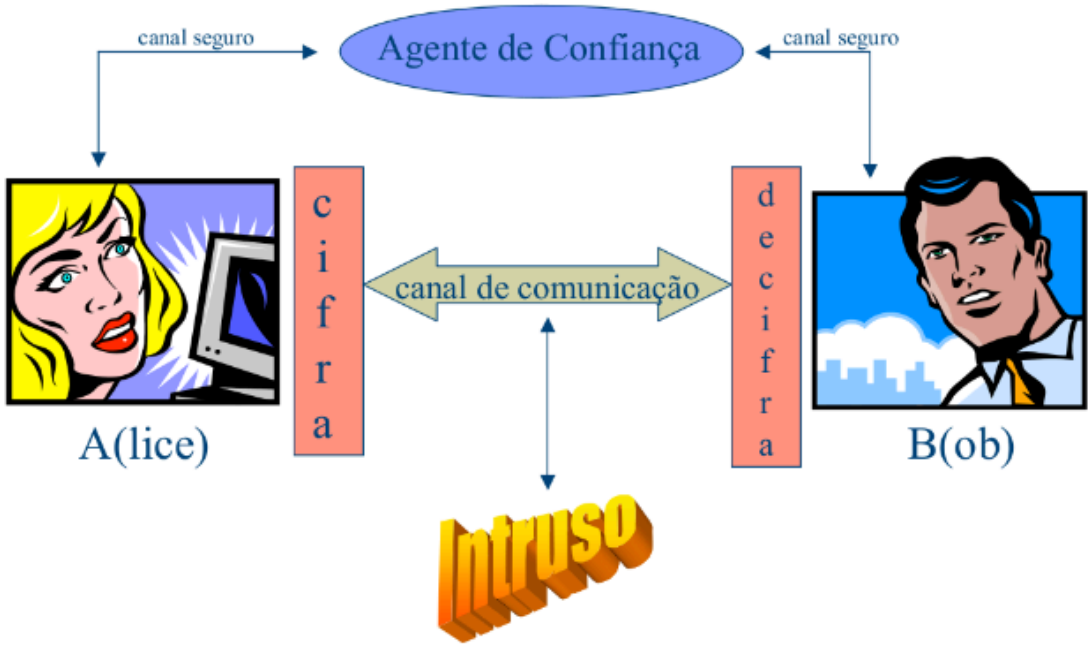


Figure 1: Fluxo do primeiro caso de uso

Princípio de Kerckhoff

Para avaliar a segurança de uma técnica criptográfica devemos assumir que esta é do conhecimento de eventuais inimigos. O corolário diz a segurança de uma cifra é assegurada pela **chave**.

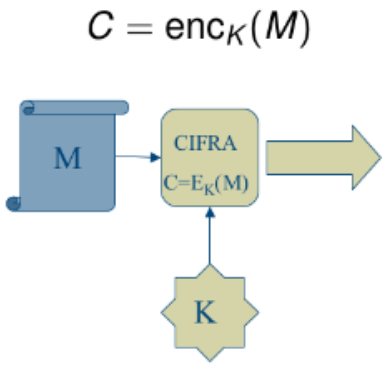


Figure 2: Fluxo do primeiro caso de uso

Terminologia

Alguns termons ligados à criptografia:

- **Texto Limpo:** Mensagem a transmitir.
- **Cifra:** Operação que transforma o texto limpo numa mensagem com significado obscurecido, originando um **criptograma**.
- **Chave**: Parâmetro de segurança da operação de cifra
- **Sistema criptográfico**: Especificação das operações de inicialização, cifra e decifragem.
- **Ataque**: Comprometimento dos objectivos da técnica criptográfica (obtenção do texto limpo sem conhecimento da chave).
- **Intruso/adversário/inimigo/spy:** Entidade que personifica quem pretende comprometer os objectivos da técnica criptográfica.

Existem dois **tipos de ataques**:

- **Passivo:** O dispõe apenas tem a capacidade de escutar o canal de comunicação.
- **Ativo:** Consegue escutar o canal e tem capacidade para manipular a informação que circula no nele.

Existem dois **tipos de segurança**:

- **Segurança Absoluta:** A segurança é estabelecida perante um adversário sem limitações computacionais.
- **Segurança Computacional:** Caso contrário.

Cifra de César

Operação de cifra consiste em realizar um **deslocamento das letras do alfabeto**. Por exemplo, cifrar a mensagem CartagoEstaNoPapo com chave K = 6 resulta em IGWZGMUKYZGTUVGVU.

Como temos um baixo número de possíveis chaves, permite uma busca exaustiva, designado por **ataque por força bruta**. Este tipo de ataques pode ser **ultrapassado, adotando tamanhos maiores para as chaves**. Esta abordagem funciona para adversários limitados computacionalmente.

A **análise de frequência permite ataques mais eficiente**. Alta frequência da letra L sugere chave K = L – A = 11 pois a letra A é muito frequênte em português.

Cifra por substituição mono-alfabética

Generaliza a cifra de César permitindo **deslocamentos diferentes para as diferentes letras do alfabeto**.

A	B	C	.....	W	Y	Z
R	X	K	.....	B	I	F

Figure 3: Fluxo do primeiro caso de uso

Também pode ser **alvo de análise de frequência das letras**.

Cifra de Vigenère (substituição poli-alfabética)

Chave é uma frase. **Cada substituição é um simples deslocamento determinado pelo carácter respectivo da chave** (A = 0, B = 1,...)

Chave:	B	A	C	O	B	A	C	O	B	A	C	O	B	A	C	O	B
Texto limpo	C	I	F	R	A	I	N	D	E	C	I	F	R	A	V	E	L
Criptograma	D	I	H	F	B	I	P	R	F	C	K	J	S	A	X	S	M

Figure 4: Fluxo do primeiro caso de uso

Se o texto **limpo for muito maior do que a chave, os padrões no texto limpo vão-se repercutir no criptograma**.

Para quebrar:
1. Inferir tamanhos prováveis para a chave por análise dos espaçamentos entre padrões repetidos.
2. Partir criptograma em fatias (uma por cada letra da chave)
3. Atacar cada uma delas (cifra de césar) por análise de frequências.

Cifras por Transposição

Altera a posição dos caracteres. Cifrar AINDAOUTRACIFRA resulta em IANADOTURCAIRFA, segundo o seguinte esquema.

2	1	3
A	I	N
D	A	O
U	T	R
A	C	I
F	R	A

Figure 5: Fluxo do primeiro caso de uso

Cifra One-Time-Pad

Requisitos para a **chave**:

- **Cumprimento** da chave é o **mesmo da mensagem** a cifrar;
- A chave é completamente **aleatória**.

Operações de **cifra/decifragem são simplesmente o xor** com a chave. Chaves só podem ser utilizadas numa única operação de cifra.

Propriedades de Segurança

Existem algumas propriédades de segurança, como:

- **Confidencialidade:** Garantir que o conteudo da mensagem só é do conhecimento dos intervenientes legítimos.
- **Integridade:** Garantir que o receptor não aceita mensagens que tenham sido manipuladas.
- **Autenticidade:** Assegurar a origem da mensagem.
- **Não repúdio:** Demonstrar a origem da mensagem.
- **Anonimato:** Não fornecer qualquer informação sobre a origem da mensagem.
- **Indentificação:** Assegurar a identidade do interveniente na comunicação.

Os **protocolos criptográficos** especificam o **modo com que se troca mensagens**, baseada nas técnicas criptográficas utilizadas.

Cifra Sequenciais

Consiste em aproximar a cifra OneTimePad por intermédio de um **gerador de chaves**, que **produz uma sequência de chave a partir de uma chave de comprimento fixo**. Utiliza-se um gerador de chaves que toma uma chave de comprimento fixo (chave inicial) e a expande para uma sequência longa de chaves. O **processamento do texto limpo é realizado símbolo a símbolo** (bit a bit, carácter a carácter, byte a byte).

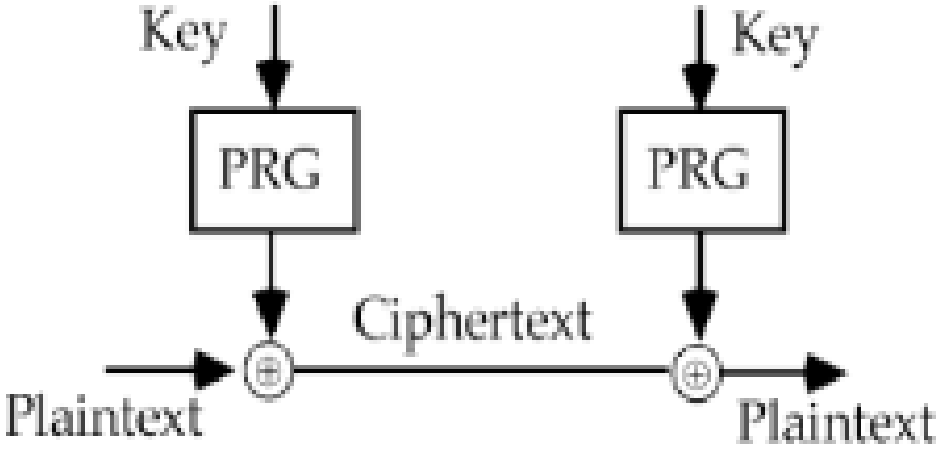


Figure 6: Fluxo do primeiro caso de uso

O período deve ser tão grande quanto possível.

A chave deve ser:
- pseudo-aleatória
- imprevisível

Cifras Síncronas

Em cifras síncronas, a sequência de chave (keystream) é gerada de maneira independente do texto claro (plaintext) ou do texto cifrado (ciphertext). A **keystream é gerada apenas a partir da chave inicial** e, possivelmente, de um vetor de inicialização (IV).

Como a keystream é sincronizada com a posição no texto, **qualquer inserção ou perda de bits** no texto cifrado resultará na perda de sincronismo entre o emissor e o receptor. Isso fará com que a **descriptografia produza dados inválidos a partir do ponto de perda ou inserção**.

Se ocorrer uma **alteração de bits** (bit flip) no texto cifrado, **apenas o bit correspondente no texto claro será afetado após a descriptografia**. Esse comportamento é uma característica das cifras de fluxo síncronas, onde os erros de transmissão não se propagam além do ponto de ocorrência.

Cifras Auto-Sincronizáveis

Em cifras auto-sincronizáveis, **cada bit da sequência de chave (keystream) é calculado a partir dos últimos n bits do texto cifrado (criptograma) e da chave original**. Esse método permite que a cifra se recupere automaticamente de erros de sincronização, pois a geração da keystream se ajusta com base nos bits recebidos.

Para permitir a inicialização e sincronização correta na recepção, **introduz-se um prefixo de n bits aleatórios** no início do texto claro (plaintext). Esse prefixo é **transmitido junto com a mensagem e utilizado pelo receptor para começar a geração correta da keystream**.

As cifras auto-sincronizáveis podem ser vulneráveis a **ataques por repetição**. Neste tipo de ataque, um intruso intercepta uma porção do criptograma (texto cifrado) e a reenvia. Se o criptograma reenviado contém um prefixo ou parte da mensagem que tem um significado útil, o receptor pode interpretá-lo como uma mensagem válida, mesmo que seja uma repetição de uma comunicação anterior.

Sequências Pseudo-Aleatórias

Critérios de Golomb:

- A **diferença no número de 1s e de 0s deve ser tão pequeno** quanto possível.
- Quando particionamos a sequência em sub-sequências de símbolos repetidos (runs), devemos encontrar um número de runs de comprimento l dado por r(l)=2-l*r (se 2-l*r>1), onde r é o número de runs.
- A **auto-correlação deve ser um valor constante** para qualquer desvio diferente de 0 (mod p).

Protecção da Chave e NONCEs

NONCE: Sequência de **bytes gerada aleatoriamente** no processo de cifra, e anexado ao criptograma.

- Número que nunca deve ser repetido.
- **Não se exige que se mantenha em segredo**, ou seja, pode ser enviado juntamente com o criptograma.

Cifras baseadas em PRFs

PRFs(Pseudo-Random Functions): PRFs são funções que, dadas uma entrada e uma chave, produzem uma saída que é indistinguível de uma saída gerada por uma função verdadeiramente aleatória. Em outras palavras, para alguém que não conhece a chave, a saída da PRF parece ser aleatória. **Sem conhecer a chave, não se consegue distinguir entre a PRF e uma função aleatória**.

Cifras sequenciais

- **Salsa20:** Chave de 256 bit, NONCE de 64 bit e contador de 64 bit.
- **Chacha20:** Chave de 256 bit, NONCE de 64 bit e contador de 64 bit.
- **HC-256:** Muito pesada numa fase de pré-processamento, mas rápida na cifra. Chave e IV de 256 bit.

Cifra por blocos

Como funcionam:

1. **Mensagem é partida em blocos** do comprimento requerido.
2. Muitas vezes, torna-se necessário fazer como que a mensagem tenha um **comprimento múltiplo do tamanho do bloco**, tendo o último bloco de ser preenchido (**padding**).

Promovem difusão e cofusão par tornar complicada a relação entre propriedades estatísticas do criptograma face às propriedades do texto limpo.

- **Difusão: Cada bit do texto limpo deve afectar o maior número de bits do criptograma**.
- **Confusão: Cada bit do criptograma deve ser uma função complexa dos bits do texto limpo**.

Cifras Blocos vs. Sequenciais

- Unidade de processamento de cicfras em bloco é um bloco de dados fixo enquanto na sequêncial é uma unidade individual de dados (bit ou byte).
- Cifras por blocos mais complexas.
- As cifras por blocos são consideravelmente mais lentas.
- Cifras sequenciais não promovem a difusão.
- Cifras por blocos protegem a chave, ao contrário das sequenciais em que por cada utilização deve ser utilizada uma chave distinta.

Padding

O Padding é assim uma estratégia para **completar o último bloco de texto-limpo**, sem perder informação sobre comprimento efectivo da mensagem.

Modos de Operação

Uma cifra por blocos de diversos modos:

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher FeedBack (CFB)
- Output FeedBack(OFB)
- Counter Mode (CTR)

Electronic Code Book (ECB)

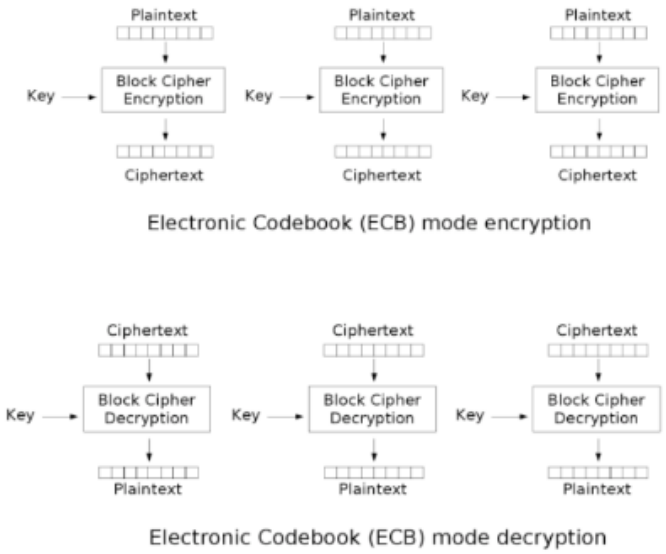


Figure 7: Fluxo do primeiro caso de uso

A repetição de blocos é detectável, **code book attack**, tornando vulnerável a ataques por repetição/substituição.

Só deve ser utilizado para cifrar mensagem de um só bloco.

Um **erro de um bit num bloco do criptograma afecta um só bloco** após a decifragem (mas todo o bloco).

Cipher Block Chaining (CBC)

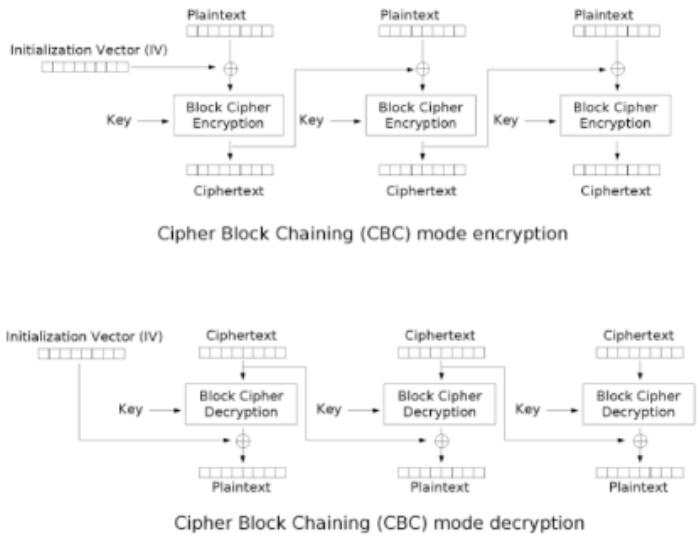


Figure 8: Fluxo do primeiro caso de uso

O **IV** previamente conhecido para iniciar o processo (**enviado em claro** ou, preferencialmente, enviado **cifrado em ECB**). **Se o IV for enviado em claro, o intruso pode alterar bits do primeiro bloco alterando os respectivos bits do IV.**

Um **erro num bloco** do criptograma corrompe dois blocos após a decifragem (mais precisamente, um bloco e um bit).

Encadeamento do processo faz **depender a operação de cifra de um bloco de todos os que o antecedem**.

Cipher FeedBack (CFB)

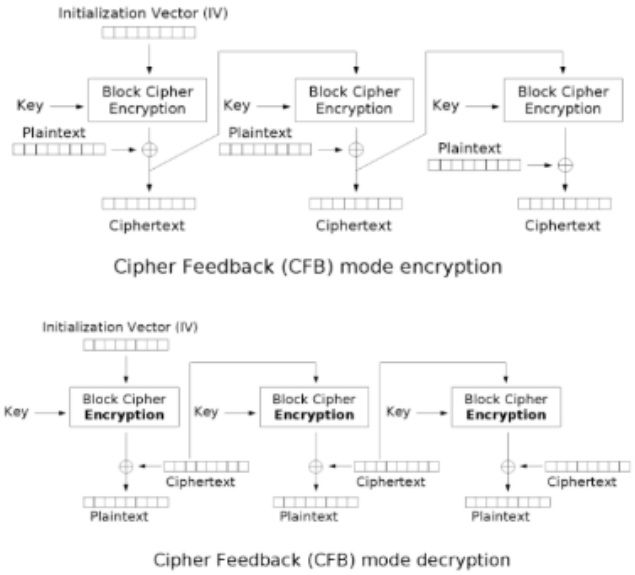


Figure 9: Fluxo do primeiro caso de uso

Número de bits no FeedBack é variável (CFBn - standard prevê n=1, 8 ou 64). A **realimentação transfere os n bits mais significativos para os menos significativos** (com shift dos restantes).

IV deve ser único por cada utilização e pode ser enviado em claro.

Um **erro num bit** do criptograma **afecta o bit respectivo no bloco e todos do bloco seguinte**.

Sequência de chave **depende do IV, chave da cifra e de todo o texto limpo já cifrado**.

Output FeedBack (OFB)

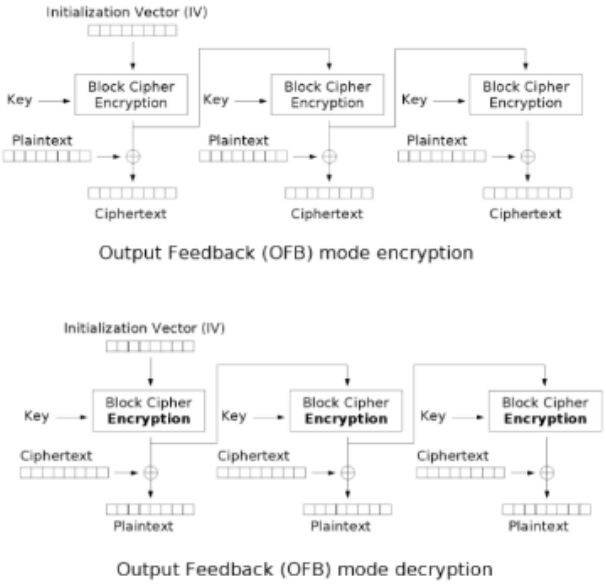


Figure 10: Fluxo do primeiro caso de uso

Implementa uma cifra sequencial síncrona com uma cifra por blocos.

Sequência de chave é obtida **iterando a cifra sobre um bloco inicial (IV)**.

Erros de bits no criptograma **só afectam os respectivos bits na mensagem original**.

Counter Mode (CTR)

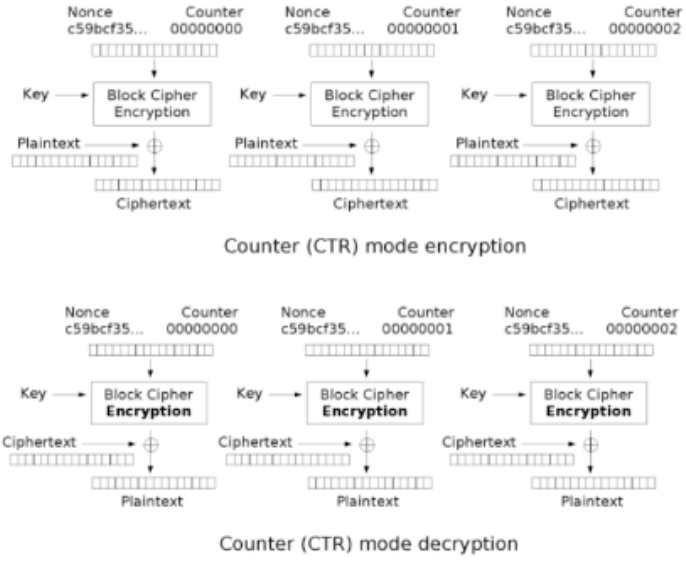


Figure 11: Fluxo do primeiro caso de uso

Simula uma cifra sequencial síncrona, em counter mode.

Nonce (IV) e Counter podem ser conjugados de diferentes formas.

Único requisito para o **Counter é produzir valores distintos para todos os blocos**.

Não impõe dependência entre processamento dos vários blocos.

Authenticated Encryption

Chamada a propostas para modos de operação que ofereçam **garantias de integridade**. Um exemplo é o **ASE**.

O ASE tem tamanho de bloco variável, tamanho da chave variável (e independente do tamanho do bloco) e o **número de rounds dependente do tamanho da chave e do bloco**.

Rounds compostos por 4 camadas:

- **Byte Substitution**

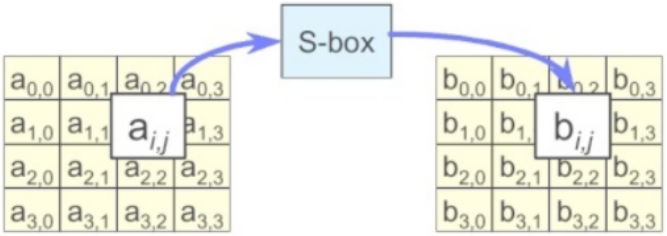
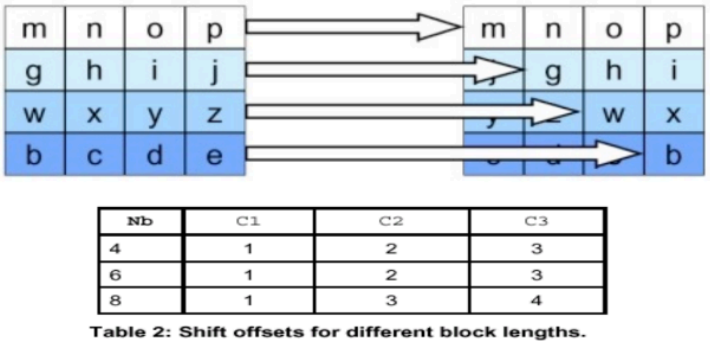


Figure 12: Fluxo do primeiro caso de uso

- **Shift Rows**



Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Table 2: Shift offsets for different block lengths.

Figure 13: Fluxo do primeiro caso de uso

- **Mix Columns**

$c(x) = \text{'03'}x^3 + \text{'01'}x^2 + \text{'01'}x + \text{'02'}$

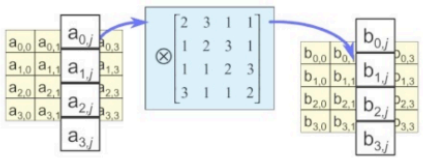


Figure 14: Fluxo do primeiro caso de uso

- **Key Addition**

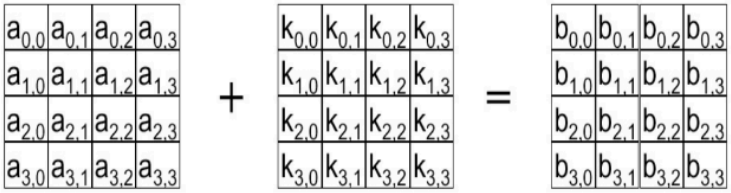


Figure 15: Fluxo do primeiro caso de uso

- **Todas as operações**

Funções sentido único

Não disponham de um algoritmo eficiente que calcule uma sua (pseudo-)inversa (complexidade computacional).

Um exemplo são as **funções de Hash criptográficas**. O objectivo é que mensagens de comprimento arbitrário sejam mapeadas num contra-domínio de tamanho fixo.

Exemplos: MD5, SHA-1, SHA-256, RIPEMD-160, SHA-3

Requisitos das funções de hash são a seguinte hierarquia de propriedades:

- **(First) pre-image resistant:** Dado um valor de hash h, deverá ser **inviável conseguir obter uma mensagem m tal que hash(m)=h**.
- **Second pre-image resistant** Dada uma mensagem m1, deverá ser **inviável obter uma mensagem m2 distinta de m1 tal que hash(m2)=hash(m1)**.
- **Collision resistant:** É **inviável encontrar mensagens distintas m1 e m2 tais que hash(m1)=hash(m2)**.

Birthday attack

Necessitamos de um contra-domínio de tamanho razoável para se conseguir resistência a colisões.

Quantas pessoas se tem (em média) que perguntar a idade numa festa de anos para encontrar duas com o mesmo dia de aniversário?

Testando cerca de sqrt(N) valores aleatórios do domínio dispõe-se de probabilidade superior a ½ de encontrar uma colisão.

Message Authentication Codes (MAC)

As funções de hash, por si só, não garantem nem a integridade nem autenticidade.

No contexto deste código, **MAC (Message Authentication Code)** é um código usado para **verificar a autenticidade e integridade de uma mensagem**. É uma forma de garantir que a mensagem não tenha sido alterada ou adulterada após ter sido criada.

O **HMAC** (Hash-based Message Authentication Code) é uma construção específica de MAC que utiliza uma função de hash criptográfica, como SHA-256, para **gerar o código de autenticação a partir da chave secreta e dos dados da mensagem**. Isso proporciona uma camada adicional de segurança, tornando o código mais resistente a ataques de falsificação de mensagem e de força bruta.

PBKDF (Password-Based Key Derivation Functions)

Constroi uma chave apropriada para uma dada técnica **a partir de chaves fracas**.

PBKDF1: Considera um valor aleatório S(salt) e um número de iterações C (iteration count). Itera uma função de hash C vezes aplicada sobre P||S. **Limita o segredo obtido ao tamanho do resultado da função de hash**.

PBKDF2: Não limita o segredo ao tamanho da função de Hash. Parametrizada por uma pseudo-random function PRF.

Acordo de Chaves

A criptografia (simétrica) obriga à existência de chaves partilhadas. O pré-acordo de chaves é um procedimento custoso e pouco flexível.

Ek1(Ek2(X))=Ek2(Ek1(X)) A e B comunicam de forma segura sem partilharem segredos. Este esquema também exibe uma vulnerabilidade, man-in-the-middle attack.

Pode-se contornar o problema da distribuição de chaves se **ambas as partes acordarem num segredo comum, trocando mensagens sobre um canal público mas sem que que seja possível derivar o segredo conhecendo apenas as mensagens trocadas**.

Algoritmos eficientes para operações modulares

- Adição; multiplicação; resíduo (módulo)
- Exponenciação
- GCD (algoritmo de Euclides)
- Inversa multiplicativa (divisão)
- Primalidade (testar se um número é primo)

Problemas intratáveis:

- Factorização de um inteiro
- Logaritmo discreto
- Raiz quadrada discreta

Protocolo (ephemeral) Diffie&Hellman

Seja p um primo e g um gerador do grupo Zp*.

1. A gera um inteiro 1<x<p e envia a B gx mod p.
2. B gera um inteiro 1<y<p e envia a A gy mod p.
3. Segredo partilhado: gxy mod p = (gy)x mod p = (gx)y mod p

Normalmente, referimo-nos aos valores envolvidos no algoritmo como pares de chaves:

- x, gx : chave privada e pública de A.
- y, gy : chave privada e pública de B.

Man-in-the-middle (I)

Na presença de um **adversário** activo, é possível este fazer-**se passar por outro agente**.

1. A gera x, calcula gx e envia este último valor a B;
2. I intercepta a mensagem de A;
3. I gera z e calcula gz que envia para A:
4. A adopta o segredo K=(gz)x=gxz que presume acordado com B;
5. I conhece o segredo K=(gx)z=gxz que A pensa partilhar com B.

I pode ainda executar uma sessão análoga com B e assim colocar-se “no meio” da comunicação entre A e B.

Criptografia de Chave-Pública

A utilização de chaves distintas para as operações de cifra e decifragem permite contornar o problema da pré-distribuição de chaves. O ponto de partida é a observação que **só a chave para decifrar necessita ser mantida secreta**.

1. Chave pública: Kc; Chave privada: Kd
2. Para A enviar mensagem M a B: envia E(KcB,M), sendo note que KcB é publicamente conhecida
3. B decifra a mensagem utilizando a sua chave privada: E(KdB,M)=M

Envelope digital: Utilizado para **garantir confidencialidade** na transmissão de uma mensagem.

1. A gera uma chave de sessão KK.
2. A criptografa a mensagem MM com KK, resultando em EK(M)EK(M).
3. A criptografa a chave de sessão KK com a chave pública de B (KcBKcB), resultando em E(KcB,K)E(KcB,K).
4. A envia E(KcB,K)E(KcB,K) e EK(M)EK(M) para B.
5. B usa sua chave privada para descriptografar E(KcB,K)E(KcB,K) e obter KK.
6. B usa KK para descriptografar EK(M)EK(M) e obter MM.

Man-in-the-middle (II)

Na sua essência, este ataque traduz-se por **fazer uso da chave pública errada**.

1. **Ao pedido de A relativo à chave pública de B, I responde com a sua própria chave pública** KcI.
2. A envia E(KcI,M)…
3. I intercepta essa mensagem, decifra-a, e torna-a a cifrar utilizando a verdadeira chave pública de B
4. B decifra mensagem

Assinatura digital

Permite verificar:
- **Integridade:** A mensagem não é modificada após a assinatura;
- **Autenticidade:** A identidade do assinante pode ser confirmada;
- **Não repúdio:** É possível demonstrar a identidade do assinante.

Note que os MACs garantem os dois primeiros requisitos mas falham no último (não repúdio).

Processo:
1. A tem uma mensagem M. A usa sua chave privada KcA para criptografar M: S=E(M,KcA).
2. A envia MM e SS para B.
3. B recebe MM e SS e usa a chave pública de A (KdA) para descriptografar SS: M′=E(S,KdA)
4. B compara M′ com M:
   - Se M′==M, a assinatura S é válida e a mensagem M é autêntica.
   - Se M′≠M, a assinatura é inválida.

Assim, o **verificador confirma que a origem da mensagem M é S**.

Man-in-the-middle (III)

Na assinatura, esse ataque traduz-se na falha de garantias de autenticação após a verificação da assinatura (**a verificação é realizada com uma chave pública errada**).

1. O atacante E intercepta a comunicação entre A e B.
2. E substitui a chave pública de A (KdA) com a sua própria chave pública (KdE).
3. B acaba recebendo a mensagem M e a assinatura S, mas verifica a assinatura usando KdE em vez de KdA.

Certificação das chaves

Todos os agentes dispõe da chave pública de um agente fidedigno - a **Autoridade de certificação (CA)**. Essa chave pública deve ser obtida por via de um canal seguro.

A CA garante (assinando digitalmente) a associação entre chave-pública/identidade do agente - o que designamos por certificado de chave pública. É responsabilidade da CA a correcção da associação estabelecida.

Protocolo Station-to-Station

1. A escolhe um número aleatório x e calcula gx, onde g é uma base pública (geralmente um gerador de um grupo cíclico). A envia gx para B.
2. B escolhe um número aleatório y e calcula gy. B calcula a chave de sessão K=(gx)y. B assina a mensagem contendo gx e gy usando sua chave privada para garantir a autenticidade da mensagem: SigB(gx,gy). B cifra a assinatura SigB(gx,gy) usando a chave de sessão K: EK(SigB(gx,gy)). B envia gy e EK(SigB(gx,gy)) e o seu certificado digital CertB para A.
3. A calcula a mesma chave de sessão K=(gy)x (graças à propriedade de Diffie-Hellman, ambos compartilham a mesma chave de sessão KK). A assina a mensagem contendo gx e gy usando sua chave privada para garantir a autenticidade: SigA(gx,gy). A cifra a assinatura SigA(gx,gy) usando a chave de sessão K: EK(SigA(gx,gy)). A envia EK(SigA(gx,gy)) e o seu certificado digital CertA para B.

RSA

O RSA **gera pares de chaves**, uma chave pública e uma chave privada, que são usadas para criptografar e descriptografar dados, respectivamente.

Processo:
1. Elecione dois números primos grandes p e q.
2. Calcule nn, que é o produto de p e q.
3. $\phi(n)=(p-1)\times(q-1)$
4. Escolha um número ee tal que $1<e<\phi(n)$ e que seja coprimo a $\phi(n)$ (ou seja, o máximo divisor comum entre e e $\phi(n)$ deve ser 1): $\gcd(e,\phi(n))=1$
5. Calcule dd, que é o inverso multiplicativo de ee módulo $\phi(n)$. Em outras palavras, dd é o número que satisfaz: $d\times e\equiv 1 \bmod \phi(n)$
6. Chave Pública: (n,e); Chave Privada: (n,d);

As maiores críticas apontadas ao RSA resultam de ele ser determinístico, ou seja, uma dada mensagem cifrada repetidas vezes resulta sempre no mesmo criptograma. Exemplos são o RSA-OAEP, RSA-PSS RSA-KEM.

RSA-KEM (Paradigma KEM/DEM)

O **RSA-KEM** é um paradigma eficaz e amplamente utilizado para garantir a segurança e eficiência na troca de chaves e na comunicação segura em redes não confiáveis.

O **KEM** é responsável por **encapsular uma chave** de sessão gerada aleatoriamente e criptografá-la usando a chave pública do destinatário.

O **DEM** é responsável por **encapsular os dados reais** que serão transmitidos usando a chave de sessão gerada.

Processo:
1. O remetente e o destinatário possuem pares de chaves RSA, uma pública e outra privada.
2. O remetente gera uma chave de sessão aleatória para ser usada na comunicação. Esta chave de sessão é criptografada usando a chave pública RSA do destinatário, produzindo um criptograma KEM.
3. O criptograma KEM, juntamente com quaisquer parâmetros adicionais necessários, é transmitido ao destinatário.
4. O destinatário recebe o criptograma KEM e o descriptografa usando sua chave privada RSA correspondente, recuperando assim a chave de sessão.
5. Com a chave de sessão recuperada, o destinatário agora pode usar essa chave para descriptografar os dados encapsulados pelo DEM, que foram transmitidos juntamente com o criptograma KEM.

Resuminho de certificados

Temos de ter uma correta **associação entre as chaves públicas e identificadores**. Para garantir essas associações, faz-se uso de uma **entidade externa de confiança (EC)**:
- Os Certificados de Chave Pública, assinados pela EC, atestam essa associação;
- Os utilizadores aceitam como válida essa associação.

Os utilizadores, de cada vez que necessitarem de uma chave publica, solicitam o respectivo certificado:
- Confirmam a **validade do certificado** verificando a **assinatura nele contido**
- Para o fazer, usam a **chave pública da EC**(essa sim, terá de ser distribuída de forma segura, usando-se padrões de segurança muito elevados).

Certificado de Chave Pública

Dados contidos num certificado:
- Identificação do titular do certificado
- Chave pública do titular
- Identificação da EC

A **assinatura** dos dados é realizada pela EC. Uma instância de cerfificados de chave pública são os **Certificados X509**. Os dados são representados como estruturas de dados atributo/valor (dicionários). As codificações dos dados estão standardizada:
- **DER:** Formato binário definido pelo standard.
- **PEM:** Representa o da informação contido no formato DER em caracteres imprimiveis.

A **assinatura do certificado efectuada pela Entidade de Certifica codifica o DER dos dados nele contidos**.

Certificados X.509v3

Colmata as versões anteriores uma vez que permite **introduzir mais atributos**, a partir do campo **Extension**.

Atributos básicos:
- **version:** Versão do standard X509 (v3)
- **serialNumber:** Número único atribuído pela EC ao certificado.
- **subject:** Identificação do titular da chave pública contida no certificado.
- **subjectPublicKeyInfo:** Estrutura contendo a chave pública do titular do certificado e identificação do algoritmo correspondente.
- **issuer:** Identificação da EC que emite o certificado.
- **signature:** Estrutura que identifica o algoritmo utilizado para gerar a assinatura da EC que acompanha o certificado.
- **validity:** Estrutura com as duas datas que delimitam o per odo de validade do certificado.

Cadeias de Certificação e de Confiança

A **validação** do certificado implica **requer o conhecimento da chave pública da Entidade de Certificação que o emitiu**. Assim, existem duas alternativas:
- A chave pública já é do conhecimento do utilizador
- Tem de ser fornecida por via de um certificado emitido por uma outra EC

Isto resulta num procedimento recursivo.

As cadeias de certificação reflectem uma **hierarquia**. No topo dela reside uma EC denominada **Root**. O certificado desta EC é emitido e assinado por ela própria, ou seja, os campos subject e issuer do seu certificado são iguais.

Validação de Certificados

1. Validade da assinatura
2. Valida a aplicabilidade do certificado
3. Valida que não foi revogado, ou seja, consultando CRLs

Âncoras de Confiança

Um utilizador **conhece um número limitado de chaves públicas pertencentes a ECs** (em geral Root CAs) e que funcionam como raizes das relações de confiança.

Processos

Um processo é uma **instância de um programa em execução** num sistema operativo.

1. **Programa em Execução (Running Program)**: Um processo é criado quando um programa é carregado na memória e começa a ser executado.
2. **Espaço de Endereço Isolado (Isolated Address Space)**: Cada processo tem o seu próprio espaço de endereços na memória, isolado dos outros processos. Isto significa que um processo não pode acessar diretamente a memória de outro processo.
3. **Executado sob um ID de Utilizador e Grupo (Executed under a User and Group ID)**: Quando um processo é iniciado, ele é executado com as permissões associadas a um utilizador e a um grupo específicos. Isto define os privilégios que o processo tem, como acesso a ficheiros e outros recursos do sistema.
4. **Utilizadores Reais ou Virtuais (Actual or Virtual Users)**: Existem utilizadores virtuais, como serviços de sistema, frequentemente utilizados para tarefas automáticas e serviços do sistema operativo , que também podem iniciar processos.
5. **IDs de Utilizador e Grupo Reais ou Efetivos (Real or Effective User and Group IDs)**: Um processo tem IDs de utilizador e grupo reais, que correspondem ao utilizador e grupo que iniciaram o processo. Além disso, existem IDs de utilizador e grupo efetivos, que podem ser temporariamente modificados para conceder ao processo permissões adicionais durante a sua execução.
6. **Pode Gerar Outros Processos (May Spawn Other Processes)**: Um processo pode criar (ou "spawn") outros processos, chamados processos filho. Isto é feito através de chamadas de sistema como fork ou exec em sistemas Unix-like. Os processos filho são independentes, mas podem comunicar com o processo pai e herdam parte do contexto de execução, como variáveis de ambiente.

Espaço de Endereços de um Processo

1. **Modelo Linear Tradicional**: O espaço de endereços é organizado de forma contínua.
2. **Código, Dados e Stack**:
   - O **código** é a parte do espaço de endereços onde o código executável do programa é armazenado. Fica geralmente na parte inferior do espaço de endereços.
   - Os **dados** são a seção contém variáveis e outras informações necessárias para a execução do programa. É dividida em duas subpartes. Os **Dados Inicializados** contêm valores pré-definidos e específicos que são usados pelo programa. Os **dados Não-Inicializados (Heap)** são espaços reservados para alocação dinâmica de memória durante a execução do programa.
   - A **Stack** situa-se na parte superior do espaço de endereços, a stack é utilizada para armazenar informações temporárias, como variáveis locais e informações de controle de chamadas de funções.

Descritores do processo

1. **O que é um Descritor de Ficheiro?**

Um descritor de ficheiro é uma representação a nível do kernel de um objeto tipo ficheiro. Este objeto pode ser um ficheiro regular, um pipe, um socket de rede, ou até mesmo um dispositivo de hardware. O descritor de ficheiro é um número inteiro que identifica unicamente um recurso aberto e permite ao processo realizar operações de leitura, escrita e manipulação desse recurso.

2. **Herança de Descritores de Ficheiro**
   - **Descritores de Ficheiro Padrão (Standard File Descriptors)**: Cada processo herda três descritores de ficheiro padrão: entrada padrão (standard input, stdin), saída padrão (standard output, stdout) e erro padrão (standard error, stderr).
     - **Entrada Padrão (stdin)**: Geralmente associado ao teclado, permite ao processo receber dados.
     - **Saída Padrão (stdout)**: Normalmente associado ao ecrã, permite ao processo enviar dados.
     - **Erro Padrão (stderr)**: Usado para saída de mensagens de erro, também geralmente associado ao ecrã.
   - **Herança de Todos os Descritores de Ficheiro Abertos**: Quando um processo cria um novo processo (processo filho), este novo processo herda todos os descritores de ficheiro abertos pelo processo pai no momento da criação. Isto significa que o processo filho pode continuar a usar esses descritores de ficheiro para interagir com os mesmos recursos que o processo pai.
3. **Benefícios da Herança de Descritores de Ficheiro**
   - **Eficiência**: Permite a comunicação entre processos e a continuidade do trabalho sem a necessidade de reabrir ficheiros ou outros recursos.
   - **Facilidade de Implementação**: Facilita a implementação de pipelines e outras formas de comunicação interprocessos (IPC).
   - **Flexibilidade**: Permite que processos filhos executem tarefas específicas com os mesmos recursos do processo pai, como redirecionar a saída de um comando para um ficheiro ou outro programa.

Criação de Processos

1. **Processos São Criados por Clonagem**: Em sistemas Unix-like, os processos são frequentemente criados através da clonagem de um processo existente usando a chamada de sistema **fork**. Este processo de clonagem cria uma cópia quase idêntica do processo original (processo pai).
2. **Cópia do Espaço de Endereços (Lazy Copy)**: Quando um processo é clonado, o espaço de endereços é copiado de maneira "preguiçosa" (lazy copy). Isto significa que o sistema operativo inicialmente não duplica toda a memória física; em vez disso, ele usa a técnica de copy-on-write. Apenas quando o processo filho ou pai tenta modificar uma página de memória é que uma cópia real dessa página é feita.
3. **Descritores de Ficheiro Abertos São Herdados**: Todos os descritores de ficheiro abertos pelo processo pai são herdados pelo processo filho. Isto permite que o processo filho tenha acesso aos mesmos recursos que o processo pai, como ficheiros, sockets e pipes.

Execução de Processos

1. **Programas São Executados**: Após a criação do processo, um programa pode ser carregado para execução no espaço de endereços do processo. Isto é frequentemente feito com a chamada de sistema **exec**, que carrega um novo programa no espaço de endereços do processo existente.
2. **Substituição do Espaço de Endereços pelo Conteúdo de um Ficheiro Executável (ABI)**: Quando **exec** é chamado, o espaço de endereços do processo é substituído pelo conteúdo de um ficheiro executável. Este ficheiro é carregado de acordo com a Interface Binária da Aplicação (Application Binary Interface, ABI) do sistema, que define como os binários são estruturados e executados.

3. **Descritores de Ficheiro Abertos São Preservados**: **Apesar da substituição do espaço de endereços, os descritores de ficheiro abertos são preservados**. Isto significa que, após a execução do novo programa, o processo mantém os mesmos descritores de ficheiro abertos que herdou ou abriu antes da chamada exec.

Comunicação Interprocessos (IPC)

A Comunicação Interprocessos (IPC) é essencial para a coordenação e troca de dados entre processos.

1. **Ficheiros (Files)**
   - **Descrição**: Os processos podem comunicar-se através de ficheiros, onde um processo escreve dados num ficheiro e outro processo lê esses dados.
   - **Usos**: Este método é simples e útil para armazenamento e comunicação de dados persistentes.
   - **Vantagens**: Fácil de implementar e utilizar.
   - **Desvantagens**: Menor eficiência e maior latência devido ao acesso ao sistema de ficheiros.
2. **IPC Tradicional/Memória Partilhada (Shared Memory)**
   - **Descrição**: Dois ou mais processos podem acessar uma área de memória comum. Isto permite uma comunicação extremamente rápida, pois os dados não precisam ser copiados entre processos.
   - **Usos**: Aplicações que requerem troca rápida de grandes quantidades de dados.
   - **Vantagens**: Alta eficiência e baixa latência.
   - **Desvantagens**: Complexidade na sincronização e gestão da concorrência.
3. **Filas de Mensagens (Message Queues)**
   - **Descrição**: Permitem a troca de mensagens entre processos através de uma fila gerida pelo kernel.
   - **Usos**: Comunicação assíncrona onde a ordem das mensagens é importante.
   - **Vantagens**: Simplicidade na comunicação e capacidade de priorização de mensagens.
   - **Desvantagens**: Pode ter limitações de tamanho e latência comparada à memória partilhada.
4. **Semáforos (Semaphores)**
   - **Descrição**: Utilizados para controlar o acesso a recursos partilhados, sincronizando processos.
   - **Usos**: Sincronização e prevenção de condições de corrida (race conditions).
   - **Vantagens**: Eficaz na gestão de acesso concorrente.
   - **Desvantagens**: Pode ser complexo de implementar corretamente e propenso a deadlocks.
5. **Pipes (Nominais ou Não Nominais)**
   - **Descrição**: Pipes são canais de comunicação unidirecionais usados para conectar processos. Pipes **não nominais (anónimos) existem apenas enquanto os processos pai e filho estão em execução**, enquanto pipes nominais (nomeados) são identificados por um nome no sistema de ficheiros e podem ser usados por processos não relacionados.
   - **Usos**: Comunicação entre processos que têm uma relação hierárquica (pai-filho) ou qualquer processo (nominais).
   - **Vantagens**: Simples de usar e implementar.
   - **Desvantagens**: Limitados a comunicação unidirecional e ao mesmo sistema.
6. **Sockets (Domínios Locais ou de Rede)**
   - **Descrição**: Sockets permitem comunicação bidirecional entre processos, tanto no mesmo sistema (sockets locais) quanto em sistemas diferentes através de uma rede (sockets de rede).
   - **Usos**: Aplicações de rede e comunicação entre processos em sistemas diferentes.
   - **Vantagens**: Flexibilidade e capacidade de comunicação através de redes.
   - **Desvantagens**: Complexidade maior em comparação com pipes e outros métodos IPC.

Permissões e Modos de Ficheiros e Diretórias Unix

Estas permissões determinam quem pode **ler**, **escrever** ou **executar** um ficheiro ou diretoria.

1. **Conformidade com POSIX**: OSIX (Portable Operating System Interface) é uma família de normas especificadas pelo IEEE para manter a compatibilidade entre sistemas operativos, inclui diretrizes abrangentes para permissões de ficheiros e diretorias.
2. 2. **Tipos de Permissões** Cada ficheiro e diretoria tem um conjunto de permissões para três tipos de utilizadores:
   - **Proprietário (Utilizador)**: O criador do ficheiro ou diretoria.
   - **Grupo**: Um conjunto de utilizadores que partilham determinadas permissões.
   - **Outros (Mundo)**: Todos os outros utilizadores.

   As permissões são representadas em três conjuntos de três caracteres, por exemplo, rwxr-xr−.
   - **r**: Permissão de leitura
   - **w**: Permissão de escrita
   - **x**: Permissão de execução

   Permissões de **Ficheiros**:
   - **r**: Permite visualizar o conteúdo de um ficheiro.
   - **w**: Permite modificar o conteúdo de um ficheiro.
   - **x**: Permite executar o ficheiro como um programa.

   Permissões de **Diretorias**:
   - **r**: Permite listar o conteúdo do diretório.
   - **w**: Permite criar, apagar ou renomear ficheiros dentro do diretório.
   - **x**: Permite aceder (entrar) no diretório.

   As permissões também podem ser representadas numericamente usando notação octal, sendo **r = 4, w = 2, x = 1**. Por exemplo:
   - rwxr-xr− traduz-se para 755.
   - rw-r−r− traduz-se para 644.
3. **Modos Especiais** POSIX também define bits de permissão especiais.
   - **Set User ID (setuid, s)**: Executa o ficheiro com as permissões do proprietário do ficheiro.
   - **Set Group ID (setgid, s)**: Executa o ficheiro com as permissões do grupo do ficheiro.
   - **Sticky Bit (t)**: Usado em diretorias para restringir a eliminação de ficheiros apenas ao proprietário do ficheiro.
4. **Gestão na Linha de Comandos** As permissões podem ser modificadas usando o comando chmod (change mode):
   - Método simbólico: **chmod u+x ficheiro** (adiciona permissão de execução para o proprietário).
   - Método numérico: **chmod 755 ficheiro** (define as permissões para rwxr-xr-x).

   Para mudar o proprietário ou grupo de um ficheiro, são usados os comandos chown e chgrp:
   - **chown utilizador ficheiro**
   - **chgrp grupo ficheiro**

Utilizadores, Grupos e Senhas

1. **Identificação de Utilizadores e Grupos**
   - **User ID (UID)**: Identificação única atribuída a cada utilizador no sistema.
   - **Group ID (GID)**: Identificação única atribuída a cada grupo de utilizadores no sistema.

- **Grupos Primários e Secundários**: Cada utilizador tem um grupo primário associado ao seu UID e pode pertencer a vários grupos secundários.

2. **Bases de Dados de Utilizadores e Grupos**
   - **Base de Dados de Utilizadores**: Armazenada em **/etc/passwd**, contém informações básicas sobre os utilizadores, incluindo o UID, GID, diretório inicial e o shell padrão. A senha criptografada é armazenada no arquivo **/etc/shadow**.
   - **Base de Dados de Grupos**: Armazenada em **/etc/group**, contém informações sobre os grupos de utilizadores, incluindo os GIDs e os utilizadores que pertencem a cada grupo.

Propriedade de Objetos do Sistema de Ficheiros

1. **Proprietário de Utilizador Atribuído**
   - Cada objeto no sistema de ficheiros é atribuído a um utilizador como proprietário.
   - O proprietário de utilizador tem controlo total sobre o objeto e pode modificar as suas permissões e conteúdo, dependendo das permissões estabelecidas.

2. **Proprietário de Grupo Atribuído**
   - Além do proprietário de utilizador, cada objeto também é atribuído a um grupo como proprietário.
   - Os membros desse grupo têm permissões específicas sobre o objeto, definidas em conjunto com as permissões do proprietário de utilizador.

Permissoes

1. **Umask: Restrição Padrão do Utilizador nas Permissões**

O umask é uma configuração que define as permissões padrão impostas a novos arquivos e diretórios criados por um utilizador. Ele atua como uma máscara de bits, **onde os bits definidos no umask são removidos das permissões padrão**.

2. **Permissões de 12 Bits Mantidas no I-node do Recurso**

Cada recurso no sistema de ficheiros Unix é representado por um inode, que armazena informações sobre o recurso, incluindo suas permissões. As **permissões são representadas por 12 bits no inode**, controlando o acesso ao recurso para diferentes classes de utilizadores.

3. **Permissões Influenciam Chamadas de Sistema (Não Comandos)** As permissões determinam quais ações os utilizadores podem realizar em um recurso através das chamadas de sistema do kernel, como open, read, write, chmod, entre outras. As permissões afetam diretamente como o sistema operativo permite ou nega o acesso a um recurso.

4. **Apenas o Superutilizador Pode Modificar a Propriedade**

Somente o superutilizador (root) tem permissão para modificar a propriedade de um arquivo ou diretório, ou seja, o proprietário e o grupo a que pertence. Isso é crucial para garantir a integridade e segurança do sistema, evitando alterações não autorizadas na propriedade dos recursos.

5. **Apenas o Proprietário (ou Superutilizador) Pode Modificar Permissões**

Apenas o proprietário de um arquivo ou diretório, ou o superutilizador, tem permissão para modificar as suas permissões.

Atributos gerais

1. **Atributos gerias**
   - **Permissões**: Tipo de permissões e conjunto de permissões para proprietário, grupo e outros.
   - **Proprietário de Utilizador (u-owner)**: O utilizador proprietário do recurso.
   - **Proprietário de Grupo (g-owner)**: O grupo proprietário do recurso.
   - **Data de Criação/Modificação (c/m-date)**: Data em que o recurso foi criado ou modificado.
   - **Nome**: O nome do recurso.
2. **Atributos de Permissões**
   - **Tipo**: Indica se o recurso é um **arquivo (-) ou um diretório (d)**.
   - **Permissões**: Representa as permissões para o proprietário, grupo e outros usuários.
   - **Set UID/GID**: Indica se o bit setuid ou setgid está definido no recurso.
   - **Sticky Bit**: Indica se o bit sticky está definido no diretório.
   - **Outros**: Pode haver outros atributos específicos do sistema.

   Exemplos: -rwxr−r−, drwxr-xr-x (arquivo e diretório, respectivamente)

   drwsrwsrwt, drwSrwSrwT (diretórios com setuid, setgid e sticky bits definidos)

   -rwxrw-r−+ (atributos adicionais especificados, como ACLs)

Permissões: Experimentação

1. **Comandos para Manipulação de Permissões**
   - **chmod**: Utilizado para alterar as permissões de um arquivo ou diretoria.
   - **chown**: Utilizado para alterar o proprietário de um arquivo ou diretoria.
   - **chgrp**: Utilizado para alterar o grupo proprietário de um arquivo ou diretoria.
2. **Chamadas de Sistema para Operações de Arquivo**
   - **read()**: Utilizado para ler o conteúdo de um arquivo.
   - **write()**: Utilizado para escrever no conteúdo de um arquivo.
   - **exec()**: Utilizado para executar um arquivo como um programa.
   - **unlink()**: Utilizado para remover um arquivo.
   - **stat()**: Utilizado para obter informações sobre um arquivo, incluindo permissões, proprietário e grupo.
3. **Ferramenta para Rastreamento de Chamadas de Sistema**
   - **strace**: Uma ferramenta de linha de comando que permite **rastrear e registrar as chamadas de sistema feitas por um processo**. Útil para entender quais chamadas de sistema estão sendo usadas por um programa específico e diagnosticar problemas de execução.
4. **Ferramentas de criação**
   - **mkdir**: Cria pasta
   - **touch**: Cria ficheiro
   - **ls -la pasta**: Lista as propriedades dos ficheiros da pasta.

Utilizador e Grupo Real vs. Efetivo

1. **Identificação Padrão**: Por padrão, o **Identificador de Utilizador Efetivo (EUID) e o Identificador de Grupo Efetivo (EGID) são iguais ao Identificador de Utilizador Real (RUID) e ao Identificador de Grupo Real (RGID)**.

2. **Se SUID, SGID estiverem definidos nos recursos:** Se o bit Set UID (SUID) ou o bit Set GID (SGID) estiverem definidos em um recurso, como um arquivo executável, o **EUID e o EGID serão alterados para os do proprietário do recurso quando esse recurso for executado**.

Essa distinção entre o utilizador e o grupo real e efetivo é importante para entender como as permissões são aplicadas e como a execução de determinados recursos pode alterar temporariamente a identidade do utilizador e do grupo.

Atributos SUID e SGID

Os atributos **Set UID (SUID)** e **Set GID (SGID)** são úteis para permitir que um usuário **execute um programa com os privilégios do proprietário do programa**. No entanto, eles também podem ser perigosos, pois podem levar a violações do princípio de privilégios mínimos.

Os atributos SUID e SGID permitem que um usuário execute um programa com privilégios além do necessário. Isso **viola o princípio de conceder apenas os privilégios mínimos** necessários para realizar uma tarefa.

Quando o atributo SUID está definido em um arquivo executável, é representado por **'s'** no lugar do bit de permissão de execução. Se o arquivo for não-executável, é representado por **'S'**.

SUID & SGID em Ficheiros

1. **'S' no ID do Proprietário do Utilizador: Sem Significado**: Quando o bit SUID ('s') é definido no ID do proprietário do utilizador, não tem nenhum significado específico. Esse cenário não é comumente encontrado e não resulta em um comportamento específico.
2. **'S' no ID do Proprietário do Grupo: Bloqueio Obrigatório**: Quando o bit SGID ('s') é definido no ID do proprietário do grupo, ele indica a aplicação de bloqueio obrigatório no arquivo. Isso significa que, ao ler ou escrever no arquivo, o sistema operativo deve aplicar o bloqueio obrigatório.
3. **'s' não tem efeito em scripts... porquê?** Quando o bit SUID ou SGID é definido em um script, como um script shell (bash, por exemplo), ele é ignorado pelo sistema operativo. Isso ocorre por motivos de segurança. Permitir que scripts sejam executados com SUID ou SGID pode ser uma séria vulnerabilidade, pois poderia permitir que um usuário execute comandos com privilégios elevados de forma não intencional ou maliciosa.

SUID & SGID em Bibliotecas Compartilhadas (DLLs)

1. **Objetos Compartilhados: .so, .so.numero**: As bibliotecas compartilhadas, também conhecidas como objetos compartilhados, são arquivos que contêm funções e dados compartilhados por diferentes programas em um sistema Unix. Elas geralmente têm a extensão ".so" ou ".so.numero".
2. **Padrões do Sistema a Nível de Sistema: /etc/...** Os sistemas Unix geralmente têm padrões definidos globalmente para bibliotecas compartilhadas, armazenados em diretórios como "/etc/...".
3. **Sobrescrita: LD_LIBRARY_PATH**: O LD_LIBRARY_PATH **é uma variável de ambiente que especifica uma lista de diretórios nos quais o linker dinâmico procura bibliotecas compartilhadas ao executar um programa**. Ela pode ser usada para substituir os diretórios padrão de busca de bibliotecas compartilhadas.

4. **Perigoso para SUID, SGID... porquê?** Permitir que bibliotecas compartilhadas sejam executadas com SUID ou SGID pode ser perigoso, pois elas são frequentemente utilizadas por vários programas e podem conter código potencialmente perigoso. Se um usuário mal-intencionado pudesse substituir uma biblioteca compartilhada com SUID ou SGID por uma versão maliciosa, poderia executar comandos com privilégios elevados de forma não intencional ou maliciosa.
5. **Se SUID ou SGID, então LD_LIBRARY_PATH é ignorado**: Por razões de segurança, quando um executável é marcado com SUID ou SGID, o LD_LIBRARY_PATH é ignorado. Isso significa que, mesmo que um usuário tenha modificado o LD_LIBRARY_PATH para apontar para um diretoria específica, essa diretoria não será usada para buscar bibliotecas compartilhadas quando o executável for executado com privilégios elevados.

Bit Sticky em Ficheiros

O bit sticky, também conhecido como bit de texto, é um atributo de permissão que pode ser aplicado a arquivos em sistemas Unix. O bit sticky foi **originalmente concebido para indicar ao sistema operacional que o código de um arquivo deve ser mantido na troca (swap) ou na memória, de modo que ele permaneça disponível para acesso rápido**. Com o advento da memória virtual, que permite o mapeamento de páginas de memória para o espaço de endereçamento do processo de forma dinâmica, o bit sticky tornou-se obsoleto e é geralmente desconsiderado.

Quando o bit sticky está definido em um arquivo executável, ele é representado por **'t'** no lugar do bit de permissão de execução. Se o arquivo for não-executável, é representado por **'T'**.

O bit sticky **não tem efeito em arquivos não-executáveis**. Ele era relevante apenas para arquivos cujo código precisava ser mantido na troca ou na memória para acesso rápido.

SUID & SGID em Diretórios

O bit SUID (Set User ID) **não tem efeito em diretorias**. Ele só tem efeito em arquivos executáveis.

Em sistemas Linux, quando o bit SGID (Set Group ID) é definido numa diretoria, o proprietário de grupo da diretoria é copiado para novos arquivos ou diretórios criados dentro dele. No entanto, isso não significa que eles herdarão as permissões do diretório pai.

A semântica do SGID em diretorias não está incluída na especificação Single UNIX Specification (SUS) V4. Portanto, seu comportamento pode variar entre diferentes sistemas operacionais Unix-like.

Bit Sticky em Diretorias

Quando o bit sticky está definido em diretorias, apenas o proprietário de um recurso, o proprietário do diretório (ou o superutilizador), pode mover, renomear ou excluir o recurso. **Isso significa que mesmo que outros usuários tenham permissão de escrita no diretório, eles não podem mover, renomear ou excluir os arquivos de outros usuários**.

O bit sticky funciona em conjunto com as permissões de escrita da diretoria.

É uma medida de segurança para evitar que usuários não autorizados excluam ou modifiquem arquivos de outros usuários em diretorias compartilhadas, como /tmp.

Experimentação com ACLs

Para ACLs, podem ser utilizados os comandos **setfacl** e **getfacl**. O **setfacl** é usado para definir as ACLs em arquivos e diretorias, enquanto o **getfacl** é usado para visualizar as ACLs existentes.

É importante entender a diferença entre a máscara de permissão real e efetiva ao trabalhar com ACLs. A máscara de permissão real refere-se às permissões tradicionais de usuário, grupo e outros, enquanto a **máscara de permissão efetiva refere-se às permissões definidas pelas ACLs**.

Quando as ACLs estão em uso, é comum ver um símbolo **'+'** após as permissões tradicionais listadas para indicar que as ACLs estão definidas para esse arquivo ou diretoria.

Atributos Adicionais/Estendidos

1. **Atributos Adicionais**: Diferentes sistemas de arquivos, como NTFS e famílias ext (como ext2, ext3, ext4), suportam atributos adicionais que podem ser associados a arquivos e diretorias. No sistema de arquivos ext2, os **atributos adicionais podem ser visualizados e modificados** usando os comandos **lsattr** e **chattr**.
2. **Atributos Estendidos**: Os atributos estendidos permitem **associar pares de nome e valor a arquivos e diretorias em sistemas de arquivos** que os suportam. Eles oferecem uma maneira de armazenar metadados adicionais além das permissões tradicionais e dos atributos do sistema de arquivos. Os comandos **getfattr** e **setfattr** são usados para **visualizar e definir atributos estendidos** em sistemas Unix.
3. **Indicação por ' ' ou '.' após as permissões, a menos que a ACL esteja definida** Quando atributos adicionais ou estendidos são definidos para um arquivo ou diretório, pode ser exibido um espaço (' ') ou um ponto ('.') após as permissões tradicionais para indicar sua presença. Isso ocorre quando uma ACL não está definida.

Poderes de Root

Certas operações no sistema operacional, como desligamento do sistema, ligação de portas de rede e outras, podem ignorar as permissões normais de arquivo e diretoria.

Para determinar se uma operação é realizada com poderes de root, pode-se verificar se o **Identificador de Utilizador Efetivo (EUID) é igual a 0, o que é o UID do root no sistema Unix**.

O **mecanismo de capacidades** é uma solução para lidar com o problema de programas que precisam de privilégios de root apenas para realizar operações específicas. **Ele permite que um programa inicie com privilégios de root, mas renuncie às capacidades que não são necessárias para realizar suas tarefas**.

É uma prática de segurança recomendada minimizar os privilégios concedidos a programas e processos, a fim de reduzir a superfície de ataque e mitigar o impacto de possíveis exploits.

Chroot Jail

Chroot, abreviação de “change root”, é uma técnica usada em sistemas Unix e Linux para **alterar o diretório raiz aparente para um processo e seus processos filhos**. Isso faz com que a **diretoria especificado seja tratado como a diretoria raiz do sistema de arquivos pelo processo e por quaisquer processos filhos**.

Quando um processo é colocado em um chroot jail, ele e todos os processos filhos gerados por ele operam dentro do novo diretório raiz especificado.

A capacidade de **usar chroot é restrita ao superutilizador (root) do sistema**.

O chroot jail é frequentemente utilizado em situações como **testes e desenvolvimento de software**, controle e compatibilidade de dependências, recuperação do sistema e separação de privilégios.

Essa técnica é útil para isolar e limitar o acesso de um processo a apenas um subconjunto específico do sistema de arquivos, proporcionando um ambiente controlado e seguro para várias finalidades, desde o desenvolvimento de software até a execução de serviços em ambientes de produção

Melhores Práticas do Chroot

1. **Mudar o diretório de trabalho para dentro do jail antes do chroot**: Garante que o processo não tenha acesso a partes indesejadas do sistema de arquivos após o chroot.
2. **Alterar o ID de usuário real/efetivo para um não-root**: Reduz o potencial de danos caso ocorra uma violação de segurança dentro do jail.
3. **Manter o mínimo possível dentro do jail**: Reduz a superfície de ataque e simplifica a gestão do jail.
4. **Ter o root como proprietário de tantos arquivos do jail quanto possível, com permissões apenas de leitura:** Limita as capacidades do processo dentro do jail e reduz os riscos de modificação ou comprometimento de arquivos críticos.
5. **Limitar todas as permissões de arquivos e diretórios**: Garante a segurança do jail, impedindo a modificação não autorizada de arquivos.
6. **Criar um script para configurar as permissões**: Automatiza e padroniza o processo de configuração do jail, reduzindo erros humanos.
7. **Executar o chroot de dentro do próprio daemon (evitar wrapping)**: Evita a necessidade de executar um wrapper em torno do daemon para iniciar o jail, o que pode introduzir complexidade e potencialmente vulnerabilidades de segurança adicionais.
8. **Carregar objetos carregados dinamicamente previamente**: Certifique-se de que todos os objetos dinamicamente carregados necessários estão disponíveis dentro do jail antes de executar o chroot. Isso evita problemas de execução devido à falta de dependências.
9. **Evitar o uso do arquivo /etc/passwd do jail:** Ao usar um chroot, é preferível evitar o uso do arquivo /etc/passwd do jail, pois isso pode causar conflitos com o arquivo /etc/passwd do sistema host. Em vez disso, é recomendável configurar uma fonte de autenticação alternativa dentro do jail.
10. **Fechar descritores de arquivo agressivamente antes do chroot:** Certifique-se de fechar todos os descritores de arquivo que não são necessários para o processo antes de executar o chroot. Isso ajuda a garantir que o processo não tenha acesso acidental a recursos externos após o chroot.

11. **Linkar arquivos de configuração de fora**: Para evitar a duplicação desnecessária de arquivos de configuração, é possível linkar arquivos de configuração de fora do jail para dentro do jail, desde que isso seja feito de forma segura e controlada.
12. **Atualizar variáveis de ambiente para refletir o novo root**: Após o chroot, é importante atualizar as variáveis de ambiente relevantes, como PATH, LD_LIBRARY_PATH, etc., para refletir o novo root. Isso garante que o processo encontre corretamente os recursos dentro do jail.

Chroot Mínimo

Este é um procedimento mínimo para configurar um chroot.

1. **Fechar descritores de arquivo não utilizados**: Antes de executar o chroot, feche todos os descritores de arquivo que não são necessários para o processo dentro do jail. Isso ajuda a garantir que o processo não tenha acesso a recursos externos após o chroot.
2. **Alterar o diretório de trabalho para o diretório do jail**: Mude o diretório de trabalho para o diretório do jail para garantir que o processo esteja operando dentro do ambiente isolado.

Definições de acesso de controlo

O controlo de acesso é um aspeto fundamental das estruturas de segurança, que regula as permissões e restrições impostas às entidades que procuram interagir com informações, serviços ou instalações físicas.

1. **NISTIR 7298**: O controlo de acesso é o processo de permitir ou negar pedidos específicos para:
   - Obter e utilizar informações juntamente com serviços de processamento de informações associados.
   - Entrar em locais físicos designados ou instalações.
2. **RFC 4949**: O controlo de acesso envolve:
   - Regular a utilização de recursos do sistema de acordo com uma política de segurança predefinida.
   - Autorizar o acesso apenas a entidades (como utilizadores, programas, processos ou outros sistemas) explicitamente permitidas pela política especificada.

Aqui, o objetivo é garantir que pessoas ou entidades sem autorização não consigam aceder a informações confidenciais, sistemas ou outros recursos críticos. Isso implica garantir que mesmo utilizadores legítimos só possam aceder aos recursos de acordo com as políticas de segurança estabelecidas. O objetivo final do controlo de acesso é facilitar o acesso autorizado de utilizadores legítimos a recursos conforme necessário para desempenhar as suas funções.

Controlo de acesso (contexto)

1. **Autenticação**: É o processo de **verificar se as credenciais fornecidas por um utilizador ou outra entidade do sistema são válidas**. Isso geralmente envolve o uso de um nome de **utilizador e uma senha**, mas também pode incluir métodos mais avançados, como autenticação biométrica (impressões digitais, reconhecimento facial, etc.) ou tokens de segurança.
2. **Autorização**: Refere-se à **concessão de um direito ou permissão a uma entidade do sistema para acessar um recurso do sistema**. Esta função determina quem é confiável para um determinado propósito. Por exemplo, após um utilizador ser autenticado com sucesso, a autorização determina quais recursos específicos ele ou ela têm permissão para acessar e quais ações podem ser realizadas nesses recursos.
3. **Auditoria**: Auditoria envolve uma **revisão independente e exame dos registos e atividades do sistema para testar a adequação dos controlos do sistema, garantir conformidade com políticas e procedimentos operacionais estabelecidos, detetar violações** de segurança e recomendar quaisquer alterações indicadas nos controlos, políticas e procedimentos.

Enquanto a autenticação verifica a identidade de um utilizador ou entidade, a autorização determina o que essa entidade pode fazer após ter sido autenticada. A auditoria, por sua vez, garante que todas as atividades sejam registradas e examinadas para garantir a conformidade com políticas e procedimentos e detectar possíveis violações de segurança.

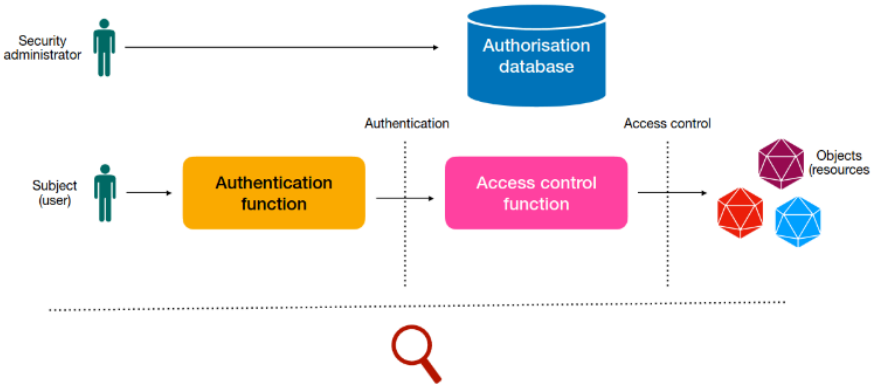


Figure 16: Fluxo do primeiro caso de uso

Politicas de acesso de controlo

1. **Controlo de Acesso Discricionário (DAC)**: Esta política controla o acesso com base na identidade do solicitante e em **regras de acesso (autorizações)** que indicam o que os solicitantes estão (ou não estão) autorizados a fazer. É chamada de discricionária porque uma entidade pode ter direitos de acesso que lhe permitem, por sua própria vontade, permitir que outra entidade acesse algum recurso. **Em resumo, o proprietário do recurso tem o poder de decidir quem pode acessá-lo e em que medida**.
2. **Controlo de Acesso Mandatório (MAC)**: Nesta política, o acesso é controlado comparando etiquetas de segurança (que indicam o quão sensíveis ou críticos são os recursos do sistema) com autorizações de segurança (que indicam que entidades do sistema têm permissão para acessar certos recursos). É chamada de mandatória porque uma entidade que tem autorização para acessar um recurso pode não, apenas por sua própria vontade, permitir que outra entidade acesse esse recurso. **Em outras palavras, o acesso é determinado por regras de segurança definidas pelo sistema, e não pelas preferências dos utilizadores ou administradores**.
3. **Controlo de Acesso Baseado em Funções (RBAC)**: Esta política controla o acesso com base nos papéis que os utilizadores desempenham no sistema e nas regras que indicam quais acessos são permitidos aos utilizadores em determinados papéis. **Em vez de atribuir diretamente permissões aos utilizadores, o RBAC atribui permissões aos papéis e, em seguida, os utilizadores são atribuídos a esses papéis**. Por exemplo, em vez de conceder acesso a um recurso específico a cada utilizador individualmente, o acesso é concedido a um papel (como administrador, utilizador padrão,...), e os utilizadores são atribuídos a esses papéis conforme necessário.
4. **Controlo de Acesso Baseado em Atributos (ABAC)**: Nesta política, o acesso é controlado com base nos atributos do utilizador, do recurso a ser acedido e nas condições ambientais atuais. **Em vez de depender apenas da identidade do utilizador ou de regras estáticas, o ABAC leva em consideração uma variedade de atributos, como a função do utilizador, a hora do dia, a localização do utilizador, etc., para determinar o acesso**. Por exemplo, um utilizador pode ter permissão para aceder a um recurso apenas se estiver a aceder a partir de um determinado local e durante um determinado período de tempo.

Subject

Um sujeito é uma **entidade capaz de aceder a objetos dentro de um sistema de computador**. Aqui estão alguns pontos importantes sobre os sujeitos:

1. **Capacidade de Acesso**: Um sujeito tem a capacidade de aceder a objetos dentro de um sistema. Isso pode incluir ler, modificar, executar ou de outra forma interagir com os objetos.
2. **Geralmente é um Processo**: Na maioria dos casos, um **sujeito é representado por um processo em execução dentro do sistema**. Este processo pode ser uma aplicação, um serviço, ou qualquer outro tipo de entidade que interage com os recursos do sistema.
3. **Representação de Utilizador**: Os sujeitos muitas vezes **agem em nome de um utilizador**, que pode ser humano ou virtual. Por exemplo, um processo de servidor web pode agir em nome dos utilizadores que acessam um site através desse servidor.
4. **Responsabilidade**: Um sujeito pode ser responsabilizado pelas ações que inicia. Isso significa que as **ações realizadas pelo sujeito podem ser atribuídas a ele, e ele pode ser responsabilizado por quaisquer consequências dessas ações**.
5. **Trilha de Auditoria**: Uma trilha de auditoria pode ser usada para **registar a associação de um sujeito com ações relevantes para a segurança realizadas num objeto**. Isso ajuda a garantir a responsabilização e a rastrear atividades suspeitas ou maliciosas.

Um sujeito é uma entidade ativa dentro de um sistema de computador que pode aceder a objetos e executar ações, muitas vezes em nome de um utilizador, e pode ser responsabilizado pelas suas ações. A manutenção de uma trilha de auditoria ajuda a monitorizar e garantir a segurança do sistema.

Classes de sujeitos

1. **Proprietário**: O proprietário é geralmente o **criador de um recurso**, como um arquivo. Para recursos do sistema, a propriedade pode pertencer a um administrador do sistema. Para recursos de projeto, um líder de projeto pode ser designado como proprietário. O proprietário geralmente tem os **privilégios mais elevados** em relação ao recurso, incluindo o direito de modificar permissões de acesso e até mesmo de remover o recurso.
2. **Grupo**: Além dos privilégios atribuídos ao proprietário, um **grupo nomeado de utilizadores também pode ser concedido direitos de acesso**. A adesão a esse grupo é suficiente para exercer esses direitos de acesso. Na maioria dos esquemas, **um utilizador pode pertencer a múltiplos grupos**, o que permite uma gestão mais flexível dos direitos de acesso. Por exemplo, em um ambiente de trabalho, pode haver grupos separados para diferentes departamentos ou equipes de projeto.
3. **Mundo (ou Todos)**: A classe Mundo representa todos os **utilizadores que podem aceder ao sistema, mas que não estão incluídos nas categorias de proprietário e grupo** para o recurso em questão. Estes utilizadores têm o menor nível de acesso e geralmente têm permissões limitadas, apenas para visualizar ou executar operações básicas no recurso.

Essas classes permitem uma gestão eficiente e granular dos direitos de acesso aos recursos do sistema, garantindo que apenas utilizadores autorizados tenham acesso aos recursos necessários e que os direitos de acesso sejam atribuídos de forma adequada com base nas necessidades e nas responsabilidades dos utilizadores.

Objetos e tipos de objetos

Um objeto é um **recurso ao qual o acesso é controlado**.

1. **Definição de Objeto**: Um objeto é uma entidade utilizada para conter e/ou receber informações. Em geral, um objeto pode ser qualquer coisa, desde dados estruturados, como registros, blocos, páginas e segmentos, até unidades mais abstratas, como arquivos, diretorias, caixas de correio, mensagens e programas. Além disso, objetos também podem ser componentes de hardware, como bits, bytes, processadores, portas de comunicação, relógios e nós de rede.
2. **Tipos de Objetos**: Os tipos de objetos que podem ser protegidos por um sistema de controlo de acesso são diversos e variam de acordo com o ambiente em que o sistema opera e os requisitos específicos de segurança. Eles podem incluir:
   - **Dados Estruturados**: Registros, blocos, páginas, segmentos.
   - **Unidades de Informação**: Arquivos, diretorias, árvores de diretórios, mensagens.
   - **Componentes de Hardware**: Bits, bytes, processadores, portas de comunicação, relógios, nós de rede.
3. **Variedade Dependente do Ambiente**: O número e os tipos de objetos a serem protegidos por um sistema de controlo de acesso dependem do ambiente em que o sistema opera e dos requisitos específicos de segurança. Por exemplo, em um ambiente de rede, objetos podem incluir nós de rede e portas de comunicação, enquanto em um sistema de arquivos, objetos podem ser arquivos individuais e diretorias.
4. **Tradeoff de Segurança, Complexidade e Facilidade de Uso**: A seleção dos objetos a serem protegidos por um sistema de controlo de acesso é influenciada pelo tradeoff entre segurança, complexidade e facilidade de uso. **Quanto mais objetos forem protegidos e quanto mais complexas forem as políticas de controlo de acesso, maior será a segurança, mas também maior será a complexidade do sistema e, potencialmente, a dificuldade de uso para os utilizadores finais**.

Os objetos em um sistema de controlo de acesso podem variar desde dados estruturados até componentes de hardware, e a seleção dos objetos a serem protegidos depende do ambiente operacional e dos requisitos específicos de segurança, bem como do equilíbrio entre segurança, complexidade e usabilidade.

Direitos de acesso

Os direitos de acesso descrevem a **forma como um sujeito pode aceder a um objeto**, ou seja, a ação que o **sujeito pode executar sobre um objeto**. Aqui estão alguns exemplos comuns de direitos de acesso:

1. **Ler**: O utilizador pode visualizar as informações em um recurso do sistema, como um arquivo, registros selecionados em um arquivo, campos selecionados dentro de um registro, ou uma combinação deles. O acesso de leitura inclui a capacidade de copiar ou imprimir informações.
2. **Escrever**: O utilizador pode adicionar, modificar ou apagar dados em um recurso do sistema, como arquivos, registros ou programas. O acesso de escrita inclui o acesso de leitura, permitindo também a visualização das informações contidas no recurso.
3. **Executar**: O utilizador pode executar programas específicos. Este acesso é geralmente concedido para programas executáveis no sistema.

4. **Apagar**: O utilizador pode apagar certos recursos do sistema, como arquivos ou registros. Este acesso permite a remoção permanente do recurso.
5. **Criar**: O utilizador pode criar novos arquivos, registros ou campos em um recurso do sistema. Este acesso permite a adição de novos dados ao sistema.
6. **Pesquisar**: O utilizador pode listar os arquivos em um diretório ou, de outra forma, pesquisar o diretório em busca de informações específicas. Este acesso facilita a localização de recursos dentro do sistema

Discretionary Access Control (DAC)

Discretionary Access Control (DAC) é um esquema no qual uma **entidade pode ser concedida direitos de acesso que permitem que ela, por sua própria vontade, habilite outra entidade a acessar algum recurso. Isso significa que o proprietário do recurso tem controle total sobre quem pode acessá-lo e em que capacidade**. Duas maneiras comuns de representar esse controle são através da **Matriz de Controle de Acesso** e de **Listas de Controle de Acesso (ACLs)** e **Tickets de Capacidade**.

1. **Matriz de Controle de Acesso**: A Matriz de Controle de Acesso é uma estrutura que lista todas as combinações de sujeitos (como utilizadores, dispositivos, aplicações, etc.) e objetos (recursos, como arquivos, diretórios, etc.) no sistema. **Cada célula da matriz representa os direitos de acesso que um sujeito específico possui sobre um objeto específico**. No entanto, como essa matriz pode se tornar muito grande e geralmente é esparsa (ou seja, muitas células estão vazias), é comum decompor essa matriz em estruturas mais gerenciáveis, como Listas de Controle de Acesso (ACLs) e Tickets de Capacidade.

2. **Listas de Controle de Acesso (ACLs)**: As ACLs são listas associadas a cada objeto que especificam quais sujeitos têm permissão para acessar esse objeto e quais operações eles podem realizar. **Cada entrada na ACL geralmente contém o identificador do sujeito e os direitos de acesso associados a esse sujeito**. **Decomponhem a Matriz de Controlo de Acesso por coluna**. Cada entrada na ACL de um objeto lista os utilizadores ou grupos de utilizadores autorizados a aceder ao objeto e os direitos de acesso concedidos a eles. Por exemplo, pode especificar se um utilizador tem permissão para ler, escrever, executar ou apagar o objeto. Uma ACL pode conter uma **entrada padrão que define os direitos de acesso para sujeitos que não estão explicitamente listados na ACL**. Isso é útil para garantir que todos os utilizadores tenham algum nível de acesso padrão aos objetos. As ACLs devem seguir o princípio do **privilégio mínimo, garantindo que os utilizadores tenham apenas os direitos de acesso necessários para realizar suas tarefas**. Isso ajuda a limitar o potencial de danos causados por acesso indevido ou malicioso. A integridade das ACLs deve ser protegida e garantida pelo sistema operativo. Normalmente, o **sistema operativo mantém as ACLs em uma área reservada, que só pode ser modificada pelo proprietário do objeto**. Isso impede que utilizadores não autorizados modifiquem as ACLs e comprometam a segurança do sistema.

3. **Tickets de Capacidade**: Os Tickets de Capacidade são uma forma alternativa de DAC, na qual o controle é baseado em capacidades (ou tickets) específicos concedidos pelo proprietário do recurso a outros utilizadores. **Cada ticket confere ao titular do ticket direitos específicos de acesso ao recurso associado a esse ticket**. Em vez de verificar uma lista de controle de acesso para determinar se um sujeito tem permissão para acessar um objeto, **o sujeito apresenta seu ticket correspondente ao objeto, e o sistema verifica a validade do ticket para conceder ou negar o acesso**. **As Capability Tickets decomponem a matriz por linha**, ou seja, cada linha representa um utilizador específico e os seus direitos de acesso aos objetos. Um Capability Ticket especifica objetos autorizados e operações permitidas para um utilizador específico. Cada utilizador possui um número de tickets e pode ser autorizado a emprestá-los ou cedê-los a outros utilizadores. **Uma característica importante dos Capability Tickets é que eles podem ser dispersos pelo sistema, o que pode apresentar um problema de segurança maior do que as ACLs. Isso ocorre porque, se um ticket for comprometido, o utilizador que o possui pode acessar os recursos associados ao ticket**. O sistema operativo pode manter todos os tickets em nome dos sujeitos em uma área privada, e pode incluir um token inalterável (como uma grande senha ou um código de autenticação de mensagem criptográfica) que pode ser verificado pelo recurso relevante sempre que o acesso for solicitado. **Os Capability Tickets são particularmente úteis em ambi-**

entes distribuídos, nos quais os recursos podem estar localizados em diferentes sistemas ou servidores. Eles permitem que os utilizadores possuam tickets específicos para acessar recursos em diferentes partes do sistema, sem depender de uma lista central de controlo de acesso.

ACLs vs. Capability Tickets

As **ACLs** são convenientes para **determinar quais utilizadores têm direitos de acesso a um objeto específico**. Isso significa que elas são eficazes quando você precisa verificar quem pode aceder a um recurso em particular. As ACLs são geralmente representadas como uma **lista associada a cada objeto, descrevendo quais utilizadores ou grupos de utilizadores têm quais direitos de acesso**.

Os **Capability Tickets** são convenientes para **determinar os direitos de acesso disponíveis para um utilizador específico**. Isso significa que eles são úteis quando você precisa saber quais recursos um determinado utilizador pode acessar. Os Capability Tickets são normalmente representados como uma **lista de tickets atribuídos a cada utilizador, descrevendo quais recursos e operações esse utilizador pode realizar**.

A **Tabela de Autorização** é uma alternativa que não é esparsa e pode ser mais conveniente do que ACLs e Capability Tickets em certos casos. Em uma tabela de autorização, **cada linha representa um sujeito, um direito de acesso e um objeto**. Isso significa que você pode determinar facilmente os direitos de acesso de um sujeito específico a um objeto específico, ou vice-versa. Além disso, a tabela de autorização pode ser facilmente implementada como um banco de dados relacional, proporcionando flexibilidade e facilidade de gerenciamento.

Role-Based Access Control (RBAC)

Enquanto os sistemas DAC tradicionais definem os direitos de acesso de utilizadores individuais e grupos de utilizadores, o RBAC **baseia-se nos papéis que os utilizadores assumem no sistema, em vez da identidade do utilizador. Um papel é uma função de trabalho dentro de uma organização que descreve as responsabilidades e as tarefas atribuídas a um utilizador**.

Nos sistemas RBAC, **os direitos de acesso são atribuídos aos papéis em vez de utilizadores individuais**. Isso significa que as permissões são definidas uma vez para cada função e depois atribuídas a todos os utilizadores que ocupam esse papel.

Os utilizadores podem ser atribuídos a diferentes **papéis de forma estática (ou seja, permanentemente) ou dinâmica (ou seja, com base em necessidades temporárias ou circunstanciais)**. Isso permite uma gestão flexível e adaptável dos direitos de acesso, à medida que as responsabilidades dos utilizadores mudam ao longo do tempo.

Em resumo, o RBAC proporciona uma abordagem mais estruturada e organizada para gerir os direitos de acesso, baseada nos papéis que os utilizadores desempenham nas organizações. Isso simplifica a gestão de permissões e promove a segurança do sistema, garantindo que os utilizadores tenham apenas os direitos de acesso necessários para desempenhar as suas funções no sistema.

No RBAC, existe uma **relação muitos para muitos**. Isso significa que **um utilizador pode ter múltiplos papéis e um papel pode ser atribuído a múltiplos utilizadores, assim como um papel pode conceder direitos de acesso a múltiplos recursos e um recurso pode ter permissões concedidas por múltiplos papéis**.

No RBAC, **os papéis podem ser tratados como objetos, o que permite a definição de hierarquias de papéis. Isso significa que os papéis podem herdar permissões de outros papéis hierarquicamente superiores**, simplificando assim a atribuição de permissões e permitindo uma gestão mais eficiente dos direitos de acesso.

A abordagem muitos para muitos do RBAC proporciona **flexibilidade e escalabilidade, permitindo que os sistemas se adaptem facilmente às mudanças nas estruturas organizacionais e nos requisitos de segurança**.

Modelos de referência RBAC

1. **RBAC0**: Este é o modelo básico que estabelece os requisitos mínimos para um sistema RBAC. Ele define a **estrutura fundamental do RBAC, incluindo a definição de papéis, permissões de acesso e atribuições de papéis a utilizadores**.
2. **RBAC1**: O modelo RBAC1 **inclui tudo no RBAC0 e adiciona a capacidade de criar hierarquias de papéis**. Isso permite que os papéis sejam organizados em uma estrutura hierárquica, simplificando a gestão e atribuição de permissões.
3. **RBAC2**: O modelo RBAC2 expande ainda mais o RBAC1, **adicionando a capacidade de impor restrições ou políticas adicionais ao sistema**. Isso pode incluir restrições baseadas em tempo, restrições de acesso condicionais ou outras políticas específicas do sistema.
4. **RBAC3**: O modelo RBAC3 combina todas as funcionalidades dos modelos anteriores, incluindo o RBAC0, RBAC1 e RBAC2. Ele incorpora hierarquias de papéis, restrições e políticas adicionais, fornecendo uma estrutura robusta e completa para sistemas de controlo de acesso baseados em papéis.

Elementos fundaentais RBCA

1. **Utilizador (User)**: Um indivíduo que tem acesso ao sistema de computador. **Cada indivíduo tem um identificador de utilizador associado que o identifica no sistema**.
2. **Papel (Role)**: Uma função de trabalho nomeada dentro da organização que controla o sistema de computador. Tipicamente, **cada papel está associado a uma descrição da autoridade e responsabilidade concedida a esse papel e a qualquer utilizador que assuma esse papel**.
3. **Permissão (Permission)**: Uma **aprovação de um determinado modo de acesso a um ou mais objetos**. Termos equivalentes incluem direito de acesso, privilégio e autorização.
4. **Sessão (Session)**: Uma **associação entre um utilizador e um subconjunto ativado do conjunto de papéis aos quais o utilizador está atribuído**. Durante uma sessão, o utilizador tem acesso aos recursos associados aos papéis ativados durante essa sessão.

Hierarquias de papéis no RBAC

As **hierarquias de papéis refletem a estrutura organizacional ao representar as relações entre diferentes funções de trabalho ou papéis**. Em muitas organizações, existe uma hierarquia clara onde certos papéis têm mais autoridade e responsabilidade do que outros.

Tipicamente, funções de trabalho ou papéis de nível mais alto têm maior autoridade para aceder a recursos dentro da organização. Por exemplo, um gestor pode ter acesso mais amplo em comparação com um funcionário regular. As hierarquias de papéis permitem essa diferenciação de acesso com base na posição hierárquica do papel dentro da organização.

As hierarquias de papéis utilizam o conceito de herança, onde um papel subordinado herda os direitos de acesso do seu papel superior. Isso significa que um papel subordinado ganha automaticamente acesso aos mesmos recursos e permissões do seu papel superior, juntamente com quaisquer permissões adicionais específicas do papel subordinado.

Um papel subordinado geralmente tem um subconjunto dos direitos de acesso do seu papel superior. Isso garante que os direitos de acesso sejam concedidos de forma controlada e hierárquica, com papéis de nível superior tendo acesso mais amplo e papéis de nível inferior tendo acesso mais limitado.

Restrições

As restrições no RBAC fornecem um meio de adaptar o modelo de Controlo de Acesso Baseado em Papéis (RBAC) às especificidades das políticas administrativas e de segurança em uma organização.

As restrições permitem que o RBAC seja ajustado para se adequar às políticas específicas de uma organização. Isso significa que as organizações podem personalizar as regras de acesso de acordo com suas necessidades e requisitos exclusivos.

Uma **restrição é uma relação definida entre papéis ou uma condição relacionada com os papéis. Por exemplo, podem existir restrições de papéis mutuamente exclusivos, onde um utilizador não pode possuir simultaneamente dois papéis que sejam incompatíveis entre si**.

Existem diferentes tipos de restrições que podem ser aplicadas, incluindo:

- **Papéis mutuamente exclusivos**: Garante que um **utilizador não possa ter acesso a papéis que são incompatíveis entre si**. Essa restrição pode ser **aplicada estaticamente**, ou seja, definida permanentemente, **ou dinamicamente**, durante uma sessão de utilização do sistema. Um utilizador **só pode ser atribuído a um papel no conjunto** (seja de forma estática ou dinâmica).

- **Cardinalidade**: Define limites para o **número de utilizadores que podem possuir um determinado papel**. Também pode referir-se ao **número máximo de papéis que um utilizador pode ser atribuído**, ou pode ativar numa sessão. Pode ainda incluir o **número máximo de papéis aos quais uma permissão específica pode ser concedida**.
- **Papéis pré-requisito**: Estabelece que **determinados papéis só podem ser atribuídos a utilizadores que já possuem outros papéis específicos**. Um utilizador pode ser atribuído a um **papel superior apenas se já estiver atribuído a um papel imediatamente inferior**.

Controlo de Acesso Baseado em Atributos (ABAC)

Define **autorizações que expressam condições sobre propriedades do recurso, do sujeito e do ambiente**. Permite que as autorizações sejam especificadas de forma mais detalhada e adaptativa. No entanto, pode **haver uma penalidade de desempenho** que pode ser um obstáculo para a utilização geral, especialmente em sistemas de grande escala.

É especialmente adequado para **serviços da web e sistemas de computação em nuvem, onde a flexibilidade é crucial devido à natureza dinâmica desses ambientes**.

Os **elementos de um modelo ABAC** incluem:

- **Atributos**, que são **definidos para entidades em uma configuração**, como utilizadores, recursos ou condições ambientais.
  - **Atributos de sujeito**: Um sujeito é uma entidade ativa (por exemplo, um utilizador, uma aplicação, um processo ou um dispositivo) que faz a informação fluir entre objetos ou muda o estado do sistema. Um papel de sujeito também pode ser um atributo.
  - **Atributos de objeto**: Um objeto, também referido como recurso, é uma entidade passiva (no contexto do pedido dado) relacionada com o sistema de informação (por exemplo, dispositivo, ficheiro, registo, tabela, processo, programa, rede, domínio) que contém ou recebe informação.
  - **Atributos de ambiente**: Eles descrevem o ambiente ou contexto operacional, técnico e até mesmo situacional em que o acesso à informação ocorre (geralmente ignorado na maioria das políticas de controlo de acesso). Por exemplo, data e hora atuais, nível de segurança atual.
- **Um modelo de política**, que **define as políticas ABAC**, especificando como os atributos são combinados para determinar as autorizações.
- **O modelo de arquitetura**, que se aplica às políticas que implementam o controlo de acesso, incluindo como as **autorizações são aplicadas e verificadas**.

Attribute-Based Access Control (ABAC) - Logical Architecture

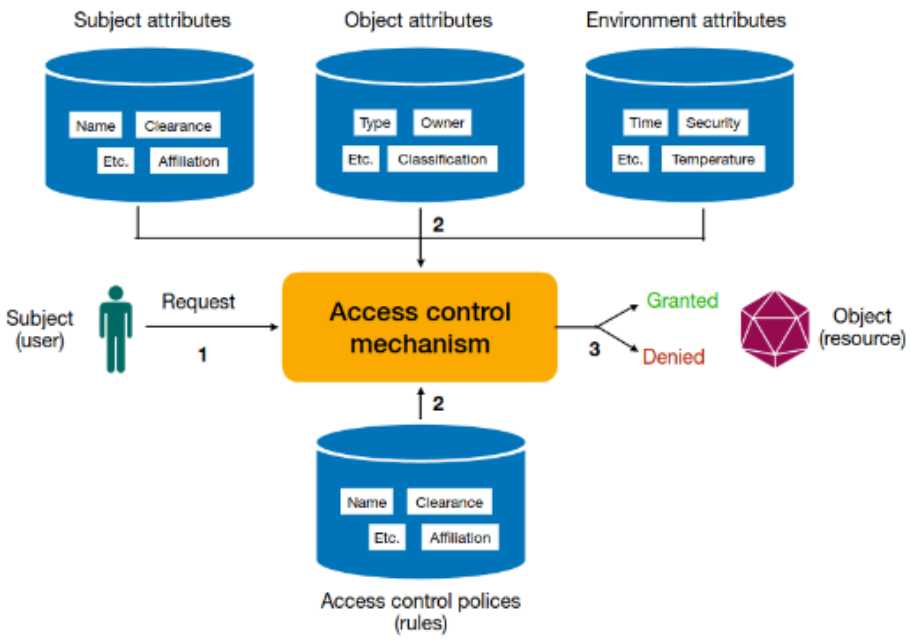


Figure 17: Fluxo do primeiro caso de uso

1. **Um sujeito solicita acesso a um objeto. Essa solicitação é encaminhada para um mecanismo de controlo de acesso.**
2. O mecanismo de controlo de acesso é governado por um conjunto de regras definidas por uma política de controlo de acesso pré-configurada. **Com base nessas regras, o mecanismo de controlo de acesso avalia os atributos do sujeito, do objeto e das condições ambientais atuais para determinar a autorização**.
3. **O mecanismo de controlo de acesso concede ao sujeito acesso ao objeto se o acesso for autorizado e nega o acesso se não for autorizado**.

- Attribute-Based Access Control (ABAC) - Politicas
- **Política**: é um **conjunto de regras e relações que governam o comportamento permitido dentro de uma organização, com base nos privilégios dos sujeitos e em como os recursos ou objetos devem ser protegidos sob quais condições de ambiente**.
- **Privilégios**: (também conhecidos como direitos, autorizações, ou atribuições) representam o **comportamento autorizado de um sujeito**; eles são definidos por uma autoridade e incorporados em uma política. A política geralmente é escrita do ponto de vista do objeto que precisa ser protegido e dos privilégios disponíveis para os sujeitos. A política é geralmente escrita a partir da perspetiva do objeto que precisa de proteção e dos privilégios disponíveis para os sujeitos.

Um futuro com menos falhas de segurança

1. Eliminar bugs / escrever **código sem bugs** e praticar codificação eficiente.
2. Eliminar código / seguir uma arquitetura **modular simples e limpa**
3. **Reduzir** a base de **código** confiável.

Eliminar bugs

- Usar modularidade e encapsulamento
- Evitar variáveis globais
- Verificar sempre os limites de arrays antes de aceder a eles
- Ter sempre em mente as limitações de representação de quantidade em linguagens de programação
- Verificar sempre a disponibilidade de recursos
- Liberar recursos assim que não forem mais necessários
- Escrever sempre um teste para cada funcionalidade do código
- Sempre que possível, usar melhores ferramentas e processos de desenvolvimento de software, ou seja, usar linguagens de programação com extensão automática de arrays
- Seguir sempre práticas de programação defensiva
- O código deve ser fácil de ler
- O código deve aderir a um padrão de codificação uniforme
- Preferir o explícito ao implícito

Eliminar código

- O código deve ser o menor possível
- Implementar um conjunto mínimo de funcionalidades
- Refatorar o código tanto quanto possível
- Aproveitar os mecanismos e abstrações do sistema operacional, ou seija, controle de acesso, processos, comunicação entre processos, sistema de arquivos e infraestrutura de serviços

Reduzir a base de código confiável

- Fazer o mínimo possível em programas setuid
- Fazer o mínimo possível como root
- Mover funções separadas para programas mutuamente desconfiados
- Programas executam com diferentes IDs de usuário
- Programas não confiam nas suas entradas (validam primeiro, atuam depois)
- Programas executam com privilégios mínimos e com o máximo de restrições de recursos possíveis (ver próximo slide)
- DJB sugere estruturar o software como funções de transformação (também conhecidas como filtros UNIX), ou seja, função de análise (parsing é sempre propenso a erros)

Receita simples para restringir a execução de processos

- Proibir novos ficheiros, novos sockets, etc., definindo os limites atuais e máximos de RLIMIT_NOFILE para 0.
- Proibir acesso ao sistema de ficheiros: usar chdir e chroot para um diretório vazio.
- Escolher um uid dedicado a este ID de processo. Isto pode ser tão simples como adicionar o ID do processo a um uid base, desde que outras ferramentas de administração do sistema evitem a mesma faixa de uid.
- Garantir que nada está a correr sob o uid: forkar um filho para executar setuid(targetuid), kill(-1,SIGKILL) e _exit(0), e depois verificar se o filho saiu normalmente.
- Proibir kill(), ptrace(), etc., definindo gid e uid para o uid alvo.
- Proibir fork(), definindo os limites atuais e máximos de RLIMIT_NPROC para 0.
- Definir os limites desejados para alocação de memória e outros recursos.
- Executar o resto do programa.

Perguntas do género das do teste

1. **Descreva sumariamente os principais mecanismos de protecção do Unix clássico. Tenha em atenção a necessidade de suportar múltiplos utilizadores, processos, dispositivos e recursos (reais e virtuais).**

   Existem diferentes políticas de controle de acesso em sistemas Unix.

   Uma delas é o Controlo de Acesso Discricionário (DAC). Este controla o acesso baseado na identificação do solicitante e vendo as autorizações que estes têm. È discriminatória pois uma entidade pode ter direitos de acesso que lhe permitam dar permissões a outras entidades.

   Outra política de controlo é o Controlo de Acesso Mandatório (MAC). Neste caso, as regras de segurança são ditadas pelo sistema, não sendo definidas nem por utilizadores nem administradores.

   Temos ainda Controlo de Acesso Baseado em Funções (RBAC). Neste caso, ao invés de atribuir autorizações aos utilizadores, atribui-se autorizações a papéis. Os utilizadores são atribuídos a esses papéis.

   Por último, temos o Controlo de Acesso Baseado em Atributos (ABAC). Neste caso, não dependemos apenas da identificação do utilizador e das suas permissões, como também é levado em consideração uma série de atributos, como a hora, o local onde se encontra o utilizador…

2. **Descreva os conceitos e relações entre attack, exploit, threat e vulnerability.** Uma vulnerabilidade é uma falha no sistema, software ou rede que pode ser usado para causar danos e obter acesso a algo não autorizado. Por exemplo, um bug em um software que permite injeção de SQL.

   Uma Ameaça/Threat é uma circunstância ou evento que pode despoletar uma vulnerabilidade. Um exemplo é um hacker que deseja roubar informações sensíveis de um banco de dados.

   Exploit é um pedaço de código ou técnica que aproveita uma vulnerabilidade para realizar um ataque. Um exemplo é um script que executa uma injeção de SQL em uma aplicação vulnerável para extrair dados.

   Ataque (Attack) utiliza um exploit para tirar proveito da vulnerabilidade e então comprometer um sistema, roubar dados, causar interrupções… Um exemplo é um ataque onde um hacker usa um exploit de injeção de SQL para acessar e roubar informações de um banco de dados.

3. **Concorda com a afirmação "O Linux, tal como o Unix clássico, oferece um controlo de acesso exclusivamente discricionário no que diz respeito ao sistema de ficheiros"? Justifique.**

   A afirmação é falsa. Existem outros mecanismos de controlo de acesso como o MAC, RBAC e ABAC. Estes não são discriminatórios.

4. **Um envelope digital (ou cifra híbrida) combina:**
   1. **uma cifra simétrica com uma cifra assimétrica**
   2. **uma cifra assimétrica com uma assinatura digital**
   3. **uma cifra simétrica com um MAC**
   4. **nenhuma das opções anteriores**

   O envelope usa uma chave simétrica e uma chave assimétrica. Inicialmente usa-se uma chave simétrica para comunicar. Posteriormente cria-se uma chave de sessão. Nesse caso, a mesma chave é usada para criptografar e descriptografar.

   R: 1

5. **Num Linux clássico (sem ACLs estendidas) um ficheiro com permissões r–r-xrw-:**
    1. **pode ser lido e mas não pode ser escrito pelo seu dono**
    2. **pode ser escrito por qualquer utilizador que não seja dono**
    3. **pode ser executado por todos os utilizadores do grupo dono**
    4. **pode ser apagado por qualquer utilizador que não seja dono ou faça parte do grupo dono**

    O proprietário (dono) do arquivo tem permissão de leitura, mas não tem permissão de escrita nem execução.

    Os usuários no mesmo grupo que o dono podem executar o arquivo e podem lê-lo, mas não têm permissão de escrita.

    Outros usuários podem ler e escrever, mas não têm permissão de execução.

    R:1