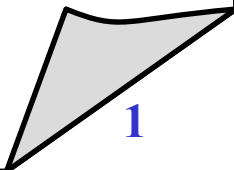




Sicurezza



Sistemi Operativi

Modulo 8: Sicurezza

Renzo Davoli
Alberto Montresor

Copyright © 2002-2021 Renzo Davoli, Alberto Montresor

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found at:

<http://www.gnu.org/licenses/fdl.html#TOC1>

Materiale ripreso e modificato da S. Ferretti

Sommario

■ Introduzione

- Definizioni: protezione, sicurezza, fiducia, politiche e meccanismi
- Crittografia
- Attacchi e vulnerabilità
- Codice “maligno”

■ Autenticazione

- Gestione password
- Metodi alternativi

■ Controllo dell'accesso

- Protezione del sistema operativo
- Autorizzazione

■ Sicurezza

- E' il problema generale, che coinvolge non solo il sistema informatico, ma anche aspetti amministrativi, legali, politici e finanziari

■ Trust

- E' la misura della *fiducia* sulla sicurezza di un sistema informatico
- Concetto “relativo”: diversi gradi di fiducia

■ Protezione

- L'insieme dei *meccanismi* utilizzati per proteggere il sistema informativo

- **Data confidentiality**

- Come mantenere la segretezza/riservatezza dei dati

- **Data integrity**

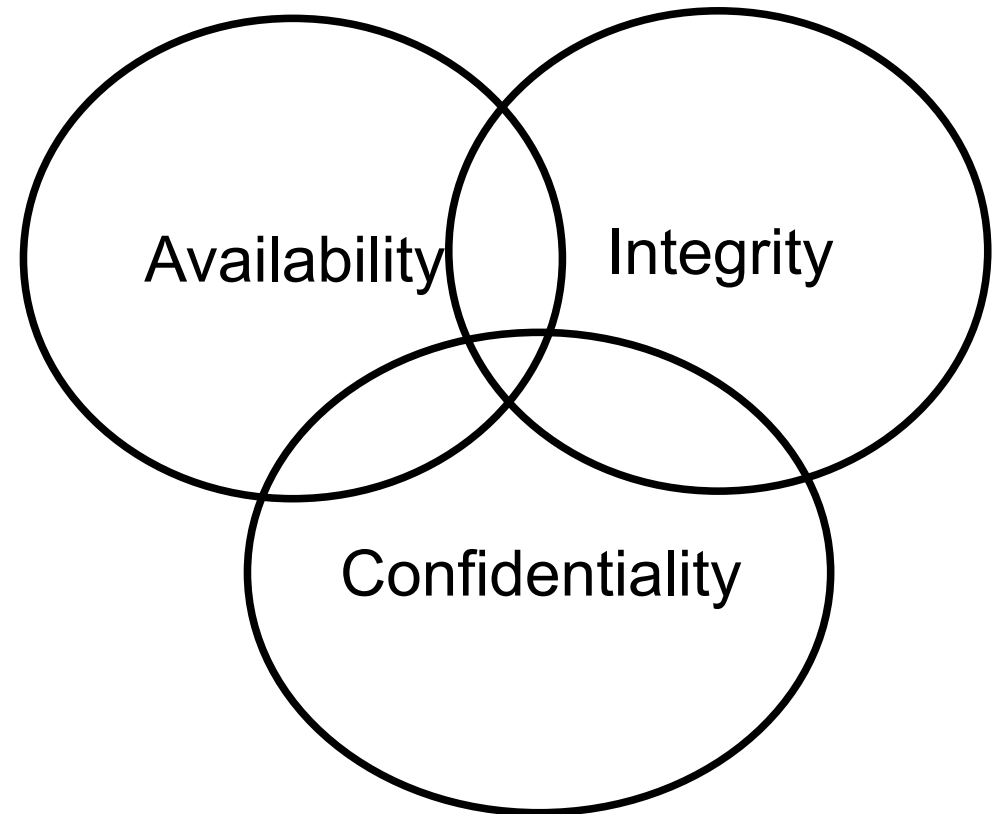
- Come evitare che i dati vengano alterati

- **System availability**

- Come garantire che il sistema continuerà ad operare

- **Alcune note:**

- Alcuni di questi obiettivi sono in contrasto fra di loro



- **Un sistema è sicuro se tutte le sue risorse sono accedute nei modi previsti e autorizzati.**
- **Violazioni alla sicurezza**
 - *Disclosure* (furto di informazione, attacco alla confidentiality)
 - *Alteration* (modifica dei dati, attacco all'integrity)
 - *Denial of service* (system availability)
- **Le informazioni contenute nei sistemi informatici sono beni e hanno un valore:**
 - Le effrazioni alla sicurezza sono pertanto reati (riconosciuti anche dal nostro codice penale)

- **Separazione della politica dai meccanismi**
 - Una *security policy* (politica di sicurezza) descrive (formalmente o informalmente) i requisiti di sicurezza del sistema
 - I *meccanismi* implementano la security policy
- **E' un concetto fondamentale di software engineering**
 - La componente che prende le decisioni "politiche" può essere completamente diversa da quella che implementa i meccanismi
 - Rende possibile
 - Cambiare la politica senza cambiare i meccanismi
 - Cambiare i meccanismi senza cambiare la politica

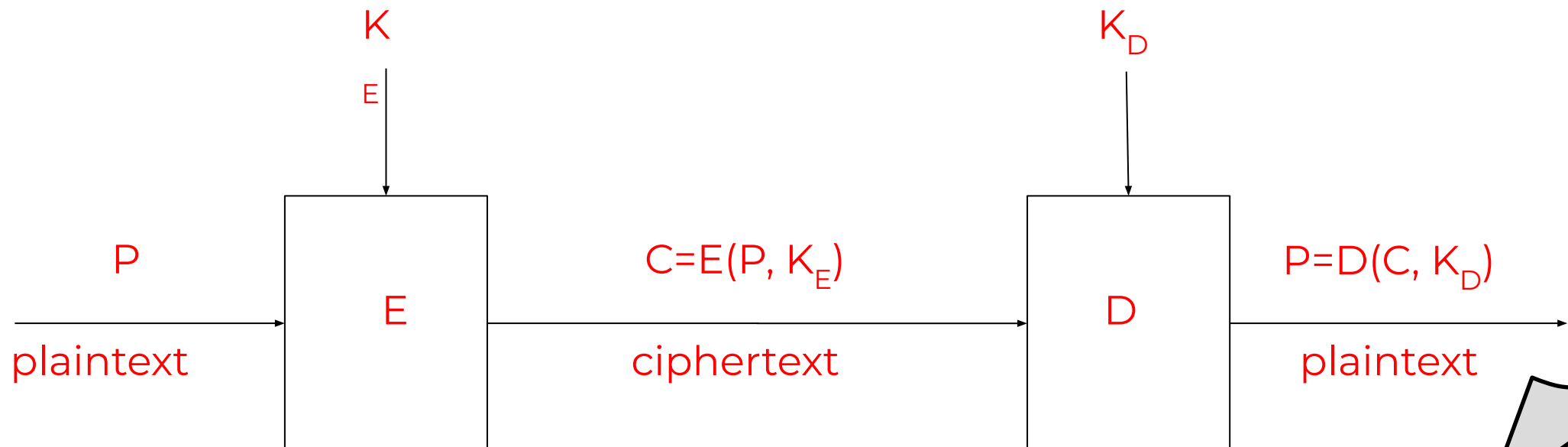
- **La scelta di una politica di sicurezza dipende da:**
 - Il tipo di attacchi e attaccanti attesi
 - Il valore delle informazioni contenute nel sistema
 - I costi dovuti all'utilizzazione di una politica di sicurezza



Due parole sulla Crittografia

■ Definizioni

- L'insieme dei meccanismi utilizzati per trasformare messaggi in chiaro (*plaintext*) in un messaggi cifrati (*ciphertext*)
- I messaggi cifrati devono essere leggibili solo a chi possiede le opportune autorizzazioni
- L'autorizzazione assume la forma di un insieme di informazioni (*chiave*) necessarie per convertire un messaggio cifrato in un messaggi in chiaro



■ Funzioni a senso unico (one-way function, OWF)

- nel seguito, utilizzeremo funzioni $y=f(x)$ con la seguenti proprietà:
 - dato x , calcolare $f(x)$ è relativamente semplice
 - dato $f(x)$, calcolare x è “computazionalmente difficile” o “impossibile”

■ Come sono fatte queste funzioni?

- generalmente, tendono a "mescolare" i bit in modo molto complesso, spesso e volentieri tramite diverse iterazioni sullo stesso insieme di bit, con operazioni di bit swapping, inversioni, etc.
- calcolare $f(x)$ consiste nel seguire l'algoritmo, invertire l'algoritmo è difficile

■ Due possibilità

- "Security by obscurity"
 - la sicurezza viene ottenuta (?) mantenendo segreti gli algoritmi di crittografia (ed ovviamente anche la chiave)
 - esempio: cifrario di cesare
- Sicurezza basata su chiavi
 - la sicurezza viene ottenuta
 - mantenendo le chiavi segrete
 - utilizzando spazi di chiavi di grandi dimensioni, in modo da rendere difficili attacchi di forza bruta
 - l'algoritmo può essere pubblico
 - maggior numero di persone può analizzare il suo comportamento (e individuare eventuali problemi)

- **A chiave segreta (crittografia simmetrica)**
 - Si usa la stessa chiave per cifrare e decifrare
 - In altre parole, $K_{\text{enc}} = K_{\text{dec}}$
- **A chiave pubblica (crittografia asimmetrica)**
 - Le chiavi di cifratura e decifratura sono diverse
 - In altre parole, $K_{\text{enc}} \neq K_{\text{dec}}$

■ Crittografia a chiave privata (o segreta)

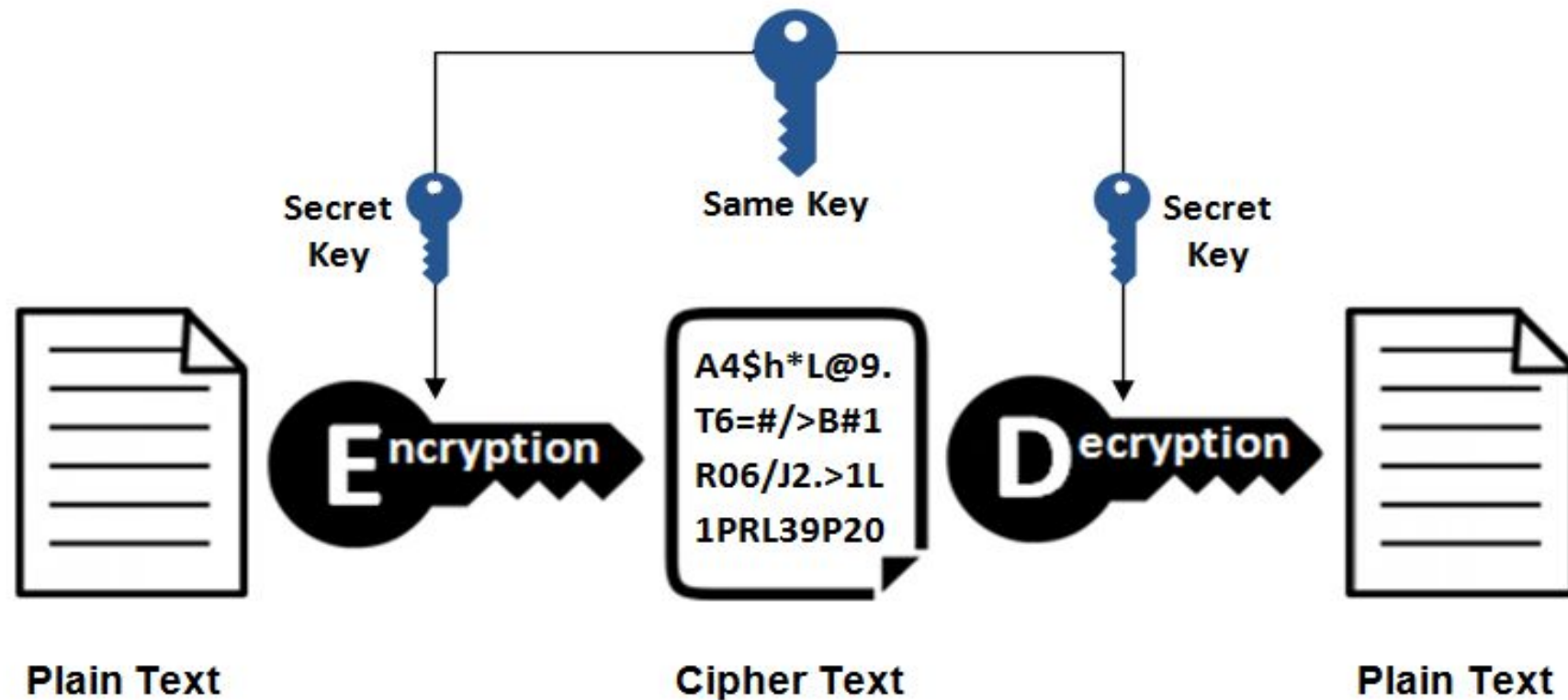
- la chiave per criptare i messaggi è la stessa usata per decriptarli
- la chiave privata deve essere mantenuta segreta, e deve essere conosciuta ad entrambi gli estremi della comunicazione
- storicamente i primi (e.g. la crittografia di Cesare)

■ Vantaggi

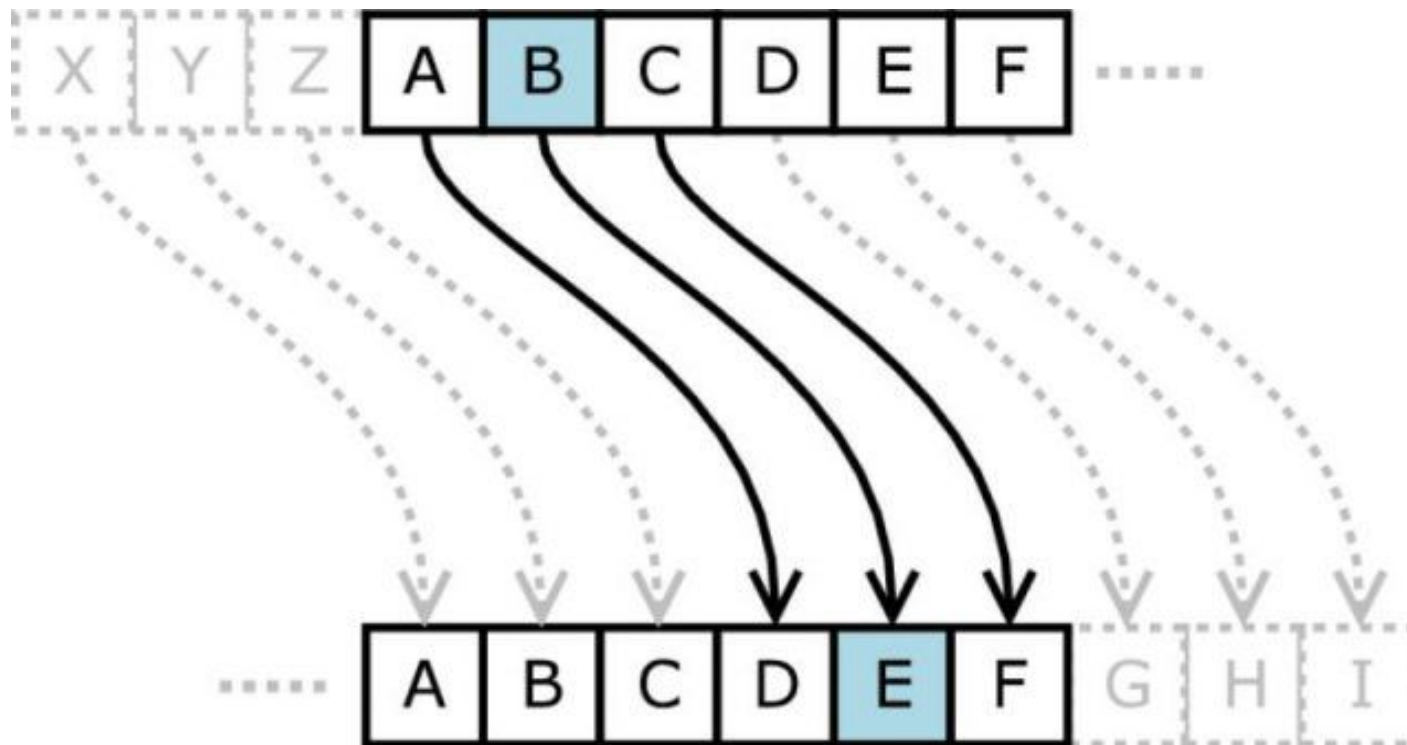
- gli algoritmi utilizzati sono molto veloci e possono essere implementati in hardware

■ Svantaggi

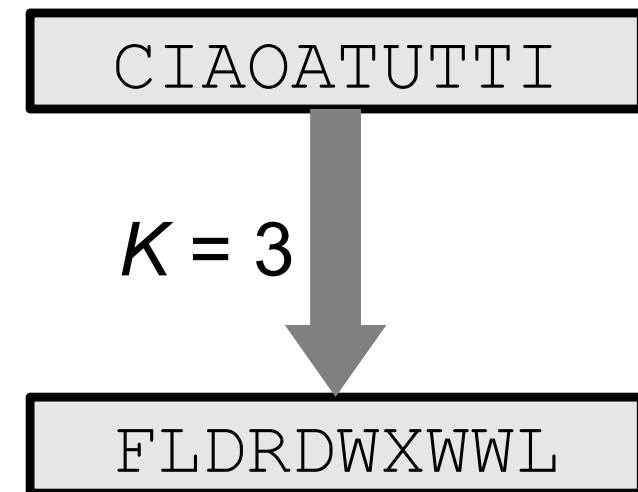
- la distribuzione delle chiavi private è un problema di sicurezza; può essere necessario uno scambio "fisico"



- Uno dei primi esempi è il “cifrario di Cesare”
- La chiave K è un numero intero
- Ogni lettera dell'alfabeto viene sostituita da quella che la segue di K posizioni



- Esempio: se $K = 3$ allora
 - $A \rightarrow D$
 - $B \rightarrow E$
 - $C \rightarrow F$
 - ...
 - $Z \rightarrow C$
- Il cifrario di Cesare non è sicuro!



- Progettato da IBM e adottato come standard dal governo USA nel 1977
 - Chiave lunga 56 bit
 - Messaggio diviso in blocchi da 64 bit che vengono cifrati individualmente
- Esistono $2^{56} \approx 7.2 \times 10^{16}$ chiavi
 - Sembra un numero grande, ma un moderno calcolatore può esaminarle tutte in poche ore!
- Una variante (*Triplo DES*) usa chiavi più lunghe e fornisce un livello accettabile di sicurezza

Key size (bits)	Cipher	Number of Alternative Keys	Time Required at 10^9 decryptions/ μs	Time Required at 10^{13} decryptions/ μs
56	DES	$2^{56} \approx 7.2 \times 10^{16}$	$2^{55} \mu s = 1.125$ years	1 hour
128	AES	$2^{128} \approx 3.4 \times 10^{38}$	$2^{127} \mu s = 5.3 \times 10^{21}$ years	5.3×10^{17} years
168	Triple DES	$2^{168} \approx 3.7 \times 10^{50}$	$2^{167} \mu s = 5.8 \times 10^{33}$ years	5.8×10^{29} years
192	AES	$2^{192} \approx 6.3 \times 10^{57}$	$2^{191} \mu s = 9.8 \times 10^{40}$ years	9.8×10^{36} years
256	AES	$2^{256} \approx 1.2 \times 10^{77}$	$2^{255} \mu s = 1.8 \times 10^{60}$ years	1.8×10^{56} years

Fonte: W. Stallings and L. Brown, *Computer Security: Principles and Practice*, Pearson; 2 edition, 2011, ISBN 978-0132775069

Quanto tempo serve per esplorare tutto lo spazio delle chiavi?

Tipo password	Lunghezza	Numero di password possibili	Tempo brute force a 10^9 pwd/s	Tempo brute force a 10^{13} pwd/s
alfa	8	$26^8 \approx 2.09 \times 10^{11}$	3.48 min	Istantaneo
alfa	10	$26^{10} \approx 1.41 \times 10^{14}$	1.64 giorni	14 sec
alfa	12	$26^{12} \approx 9.54 \times 10^{16}$	3.03 anni	2 ore
alfa+ALFA+num	12	$52^{12} \approx 3.23 \times 10^{21}$	1.0×10^5 anni	10 anni
alfa+ALFA+num	15	$62^{15} \approx 7.69 \times 10^{26}$	2.4×10^{10} anni	2.4×10^6 anni

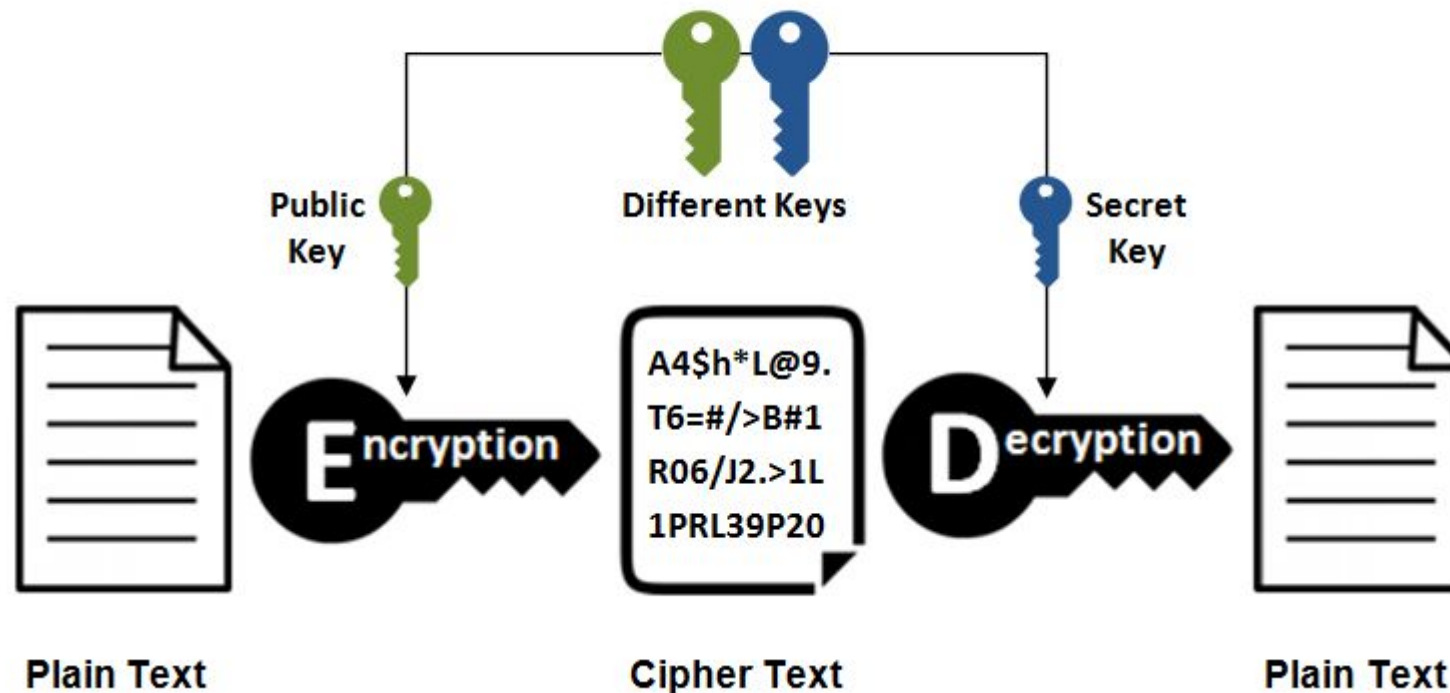
■ Punti deboli

- **Distribuzione delle chiavi** – Necessario un meccanismo sicuro per decidere la chiave da usare tra due parti
- **Scalabilità** – Per ogni coppia di interlocutori, serve una chiave diversa → il numero di chiavi cresce in maniera esponenziale
- **Sicurezza** limitata – Fornisce confidenzialità, ma limitata autenticità e non garantisce non-repudiabilità

■ Crittografia a chiave pubblica

- esistono due chiavi distinte
 - la *chiave pubblica* è utilizzata per criptare i messaggi in chiaro
 - la *chiave privata* è utilizzata per decriptare i messaggi cifrati
- la chiave privata deve essere mantenuta segreta, la chiave pubblica può (deve) essere pubblicata
- relazione fra chiave pubblica e chiave privata
 - ovviamente, devono essere in qualche modo collegate
 - dobbiamo garantire però che data la chiave pubblica, sia praticamente impossibile risalire alla chiave privata

Asymmetric Encryption



▪ Funzionamento di RSA (schema)

- vengono scelti due numeri primi molto grandi **p** e **q** (almeno 100 cifre)
- si chiami **n=pq**.
- si sceglie un valore **d** in modo tale che sia primo con **(p-1)(q-1)**
 - cioè: **MCD(d,(p-1)(q-1))=1**.
- sia **e** l'inverso moltiplicativo di **d mod (p-1)(q-1)**
 - cioè: **de mod (p-1)(q-1)=1**
- allora
 - **E(m) = C = m^e mod n**
 - **D(C) = m = C^d mod n**

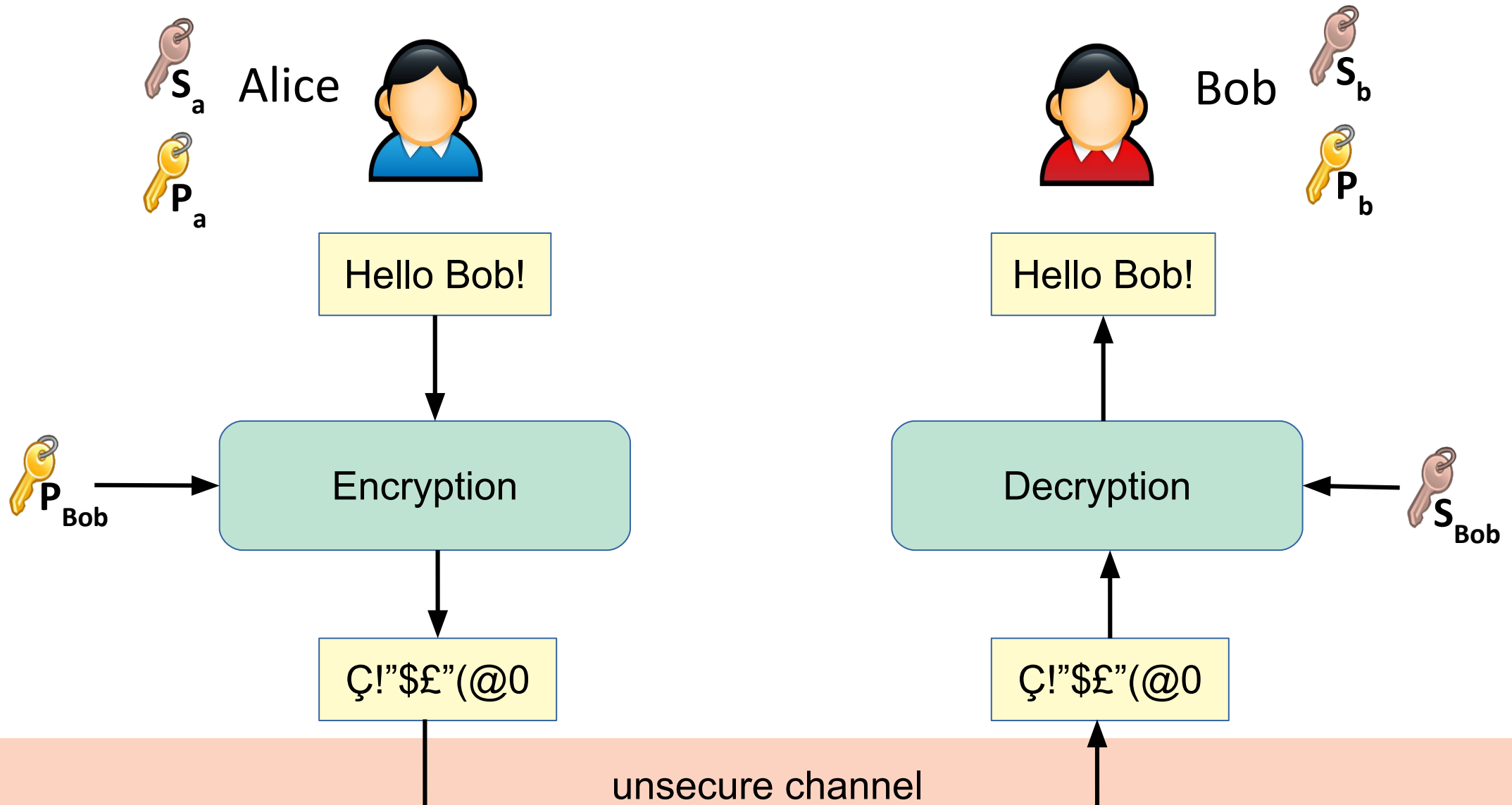
- Passo 1: Scelta dei numeri primi
 - $p = 61$ (primo)
 - $q = 53$ (primo)
- Passo 2: Calcolo di n
 - $n = p \times q = 61 \times 53 = 3233$
- Passo 3: Calcolo $(p-1)(q-1)$
 - $(p-1)(q-1) = (61-1)(53-1) = 60 \times 52 = 3120$
- Passo 4: Scelta di d
 - Dobbiamo trovare un d tale che $\text{MCD}(d, 3120) = 1$
 - Proviamo $d = 2753$:
 - Verifichiamo: $\text{MCD}(2753, 3120) = 1$ ✓

- Passo 5: Calcolo e (inverso moltiplicativo)
 - Dobbiamo trovare e tale che: $d \times e \equiv 1 \pmod{3120}$
 - Usando l'[algoritmo di Euclide esteso](#):
 - $e = 17$
 - Verifica: $2753 \times 17 = 46801 = 15 \times 3120 + 1$
 - Quindi: $2753 \times 17 \equiv 1 \pmod{3120}$ ✓
- **Chiavi risultanti**
 - **Chiave pubblica:** $(e, n) = (17, 3233)$
 - **Chiave privata:** $(d, n) = (2753, 3233)$

- **Chiavi risultanti**
 - **Chiave pubblica:** $(e, n) = (17, 3233)$
 - **Chiave privata:** $(d, n) = (2753, 3233)$
- **Esempio** - Messaggio da cifrare: $m = 123$
 - Cifratura: $C = m^e \bmod n$
 - $C = 123^{17} \bmod 3233$
 - $C = 855$
 - Decifratura: $m = C^d \bmod n$
 - $m = 855^{2753} \bmod 3233$
 - $m = 123$ ✓

Crittografia a chiave pubblica: Esempio di uso

Crittografia



- vedi script: [crypto_demo.sh](#)

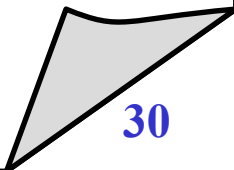
- **Comandi usati**

- `openssl genpkey` - Generazione chiavi private RSA
- `openssl pkey` - Estrazione chiavi pubbliche
- `openssl pkeyutl` - Operazioni di crittografia/firma
- `openssl dgst` - Calcolo hash SHA-256
- `hexdump` - Visualizzazione dati binari
- Standard Unix: `mkdir`, `cp`, `cat`

- **Simuleremo le comunicazioni copiando file in 2 directory ...**



Attacchi



■ Per motivazione:

- Crimine di opportunità vs un attacco con obiettivo specifico
- Motivazione finanziaria vs motivazione politica
- Divertimento – ovvero, nessun movente
- Attacchi *attivi* vs attacchi *passivi*

■ Per metodologia:

- Attacchi “interni”
 - Software o utenti “all'interno” del sistema protetto
 - Possono seguire un attacco esterno precedente
- Attacchi “esterni”
 - Tramite interfacce di comunicazione con l'esterno
 - Tramite rete, ma anche tramite CD, floppy, etc.

- **Quali sono gli obiettivi degli attaccanti?
(dal punto di vista tecnico)**
 - Acquisire una qualche forma di controllo della macchina
 - Se possibile, acquisire il controllo totale della macchina
- **Nota sui sistemi operativi multi-utente**
 - Permettono a più persone di accedere allo stesso sistema informativo
 - Vi è una distinzione fra:
 - *Utenti normali*
 - hanno accesso solo ad un sottoinsieme di risorse personali
 - *Superutenti, root, amministratori di sistema, administrator*
 - hanno accesso all'intero insieme di risorse della macchina
 - la controllano totalmente

■ Hardware

- L'hardware fornisce i meccanismi base per implementare i meccanismi di protezione più complessi
- Ma innanzitutto: cosa si intende con hardware?
 - Il processore, la memoria, i dispositivi
 - Ma anche l'ambiente fisico in cui si trova la macchina

■ Sistema operativo (nucleo)

- Il nucleo del sistema operativo fornisce due meccanismi fondamentali per garantire la protezione del sistema:
 - Autenticazione
 - Autorizzazione

▪ Librerie, tool di sistema

- Spesso e volentieri
 - Contengono grandi quantità di codice
 - Vengono eseguiti in modalità superutente
- Quali sono gli attacchi possibili a questi elementi?
 - Attacchi interni e esterni

▪ Applicazioni

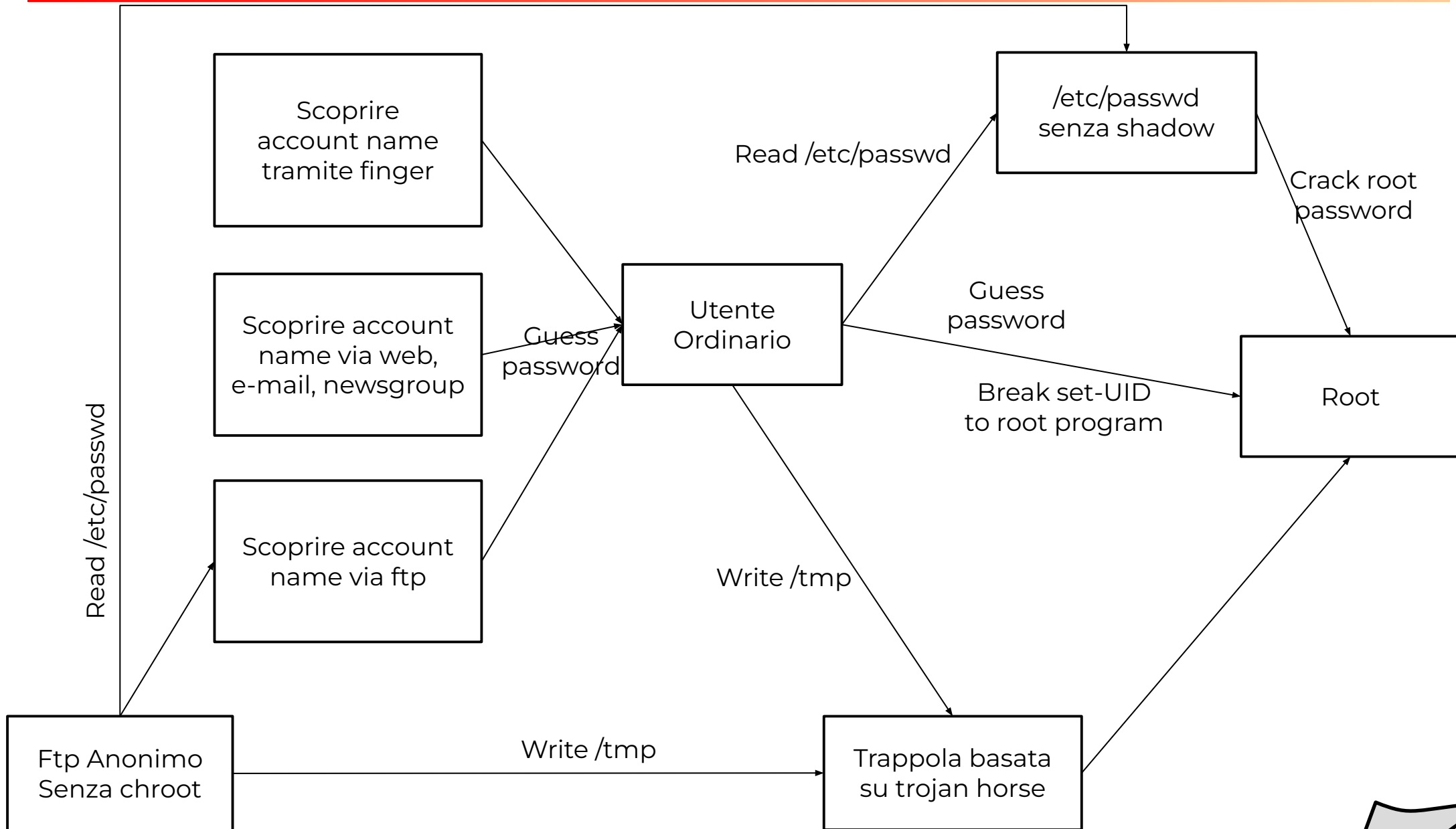
- Possono fornire una prima "testa di ponte" per attaccare un sistema

▪ Utenti

- Sono l'anello più debole!
- Tutta la sicurezza ruota intorno a loro

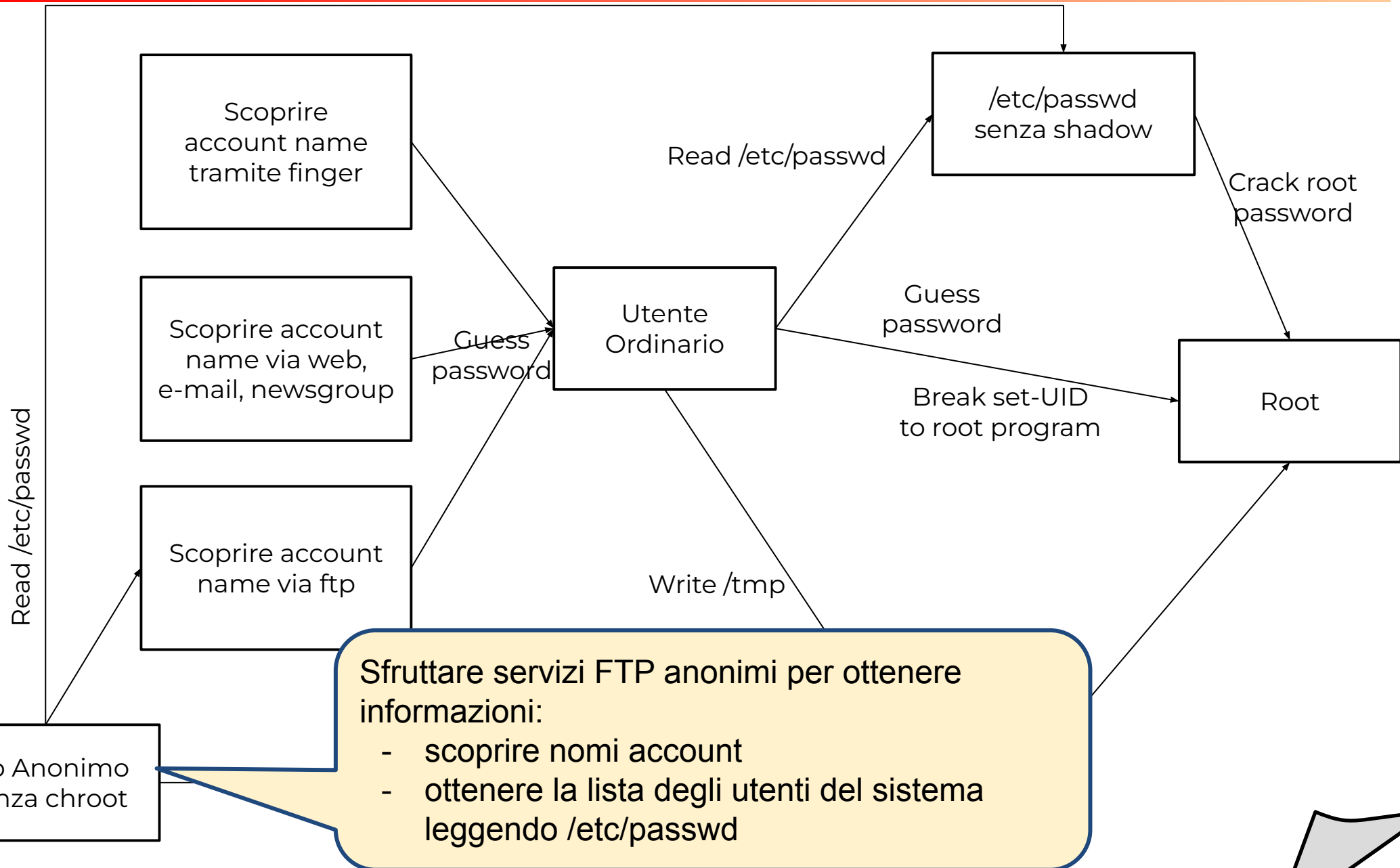
Percorsi di attacco

Attacchi



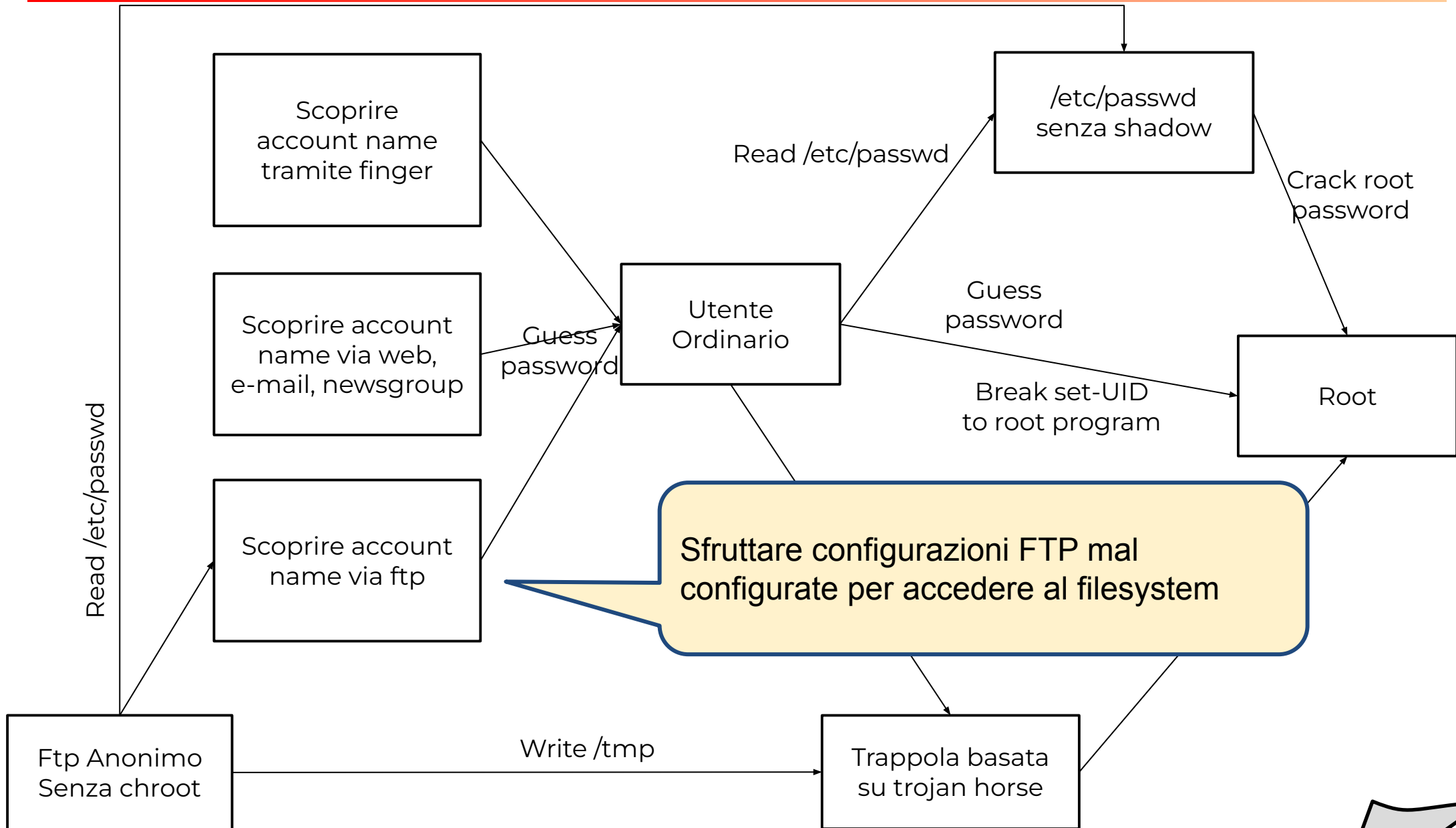
Percorsi di attacco

Attacchi



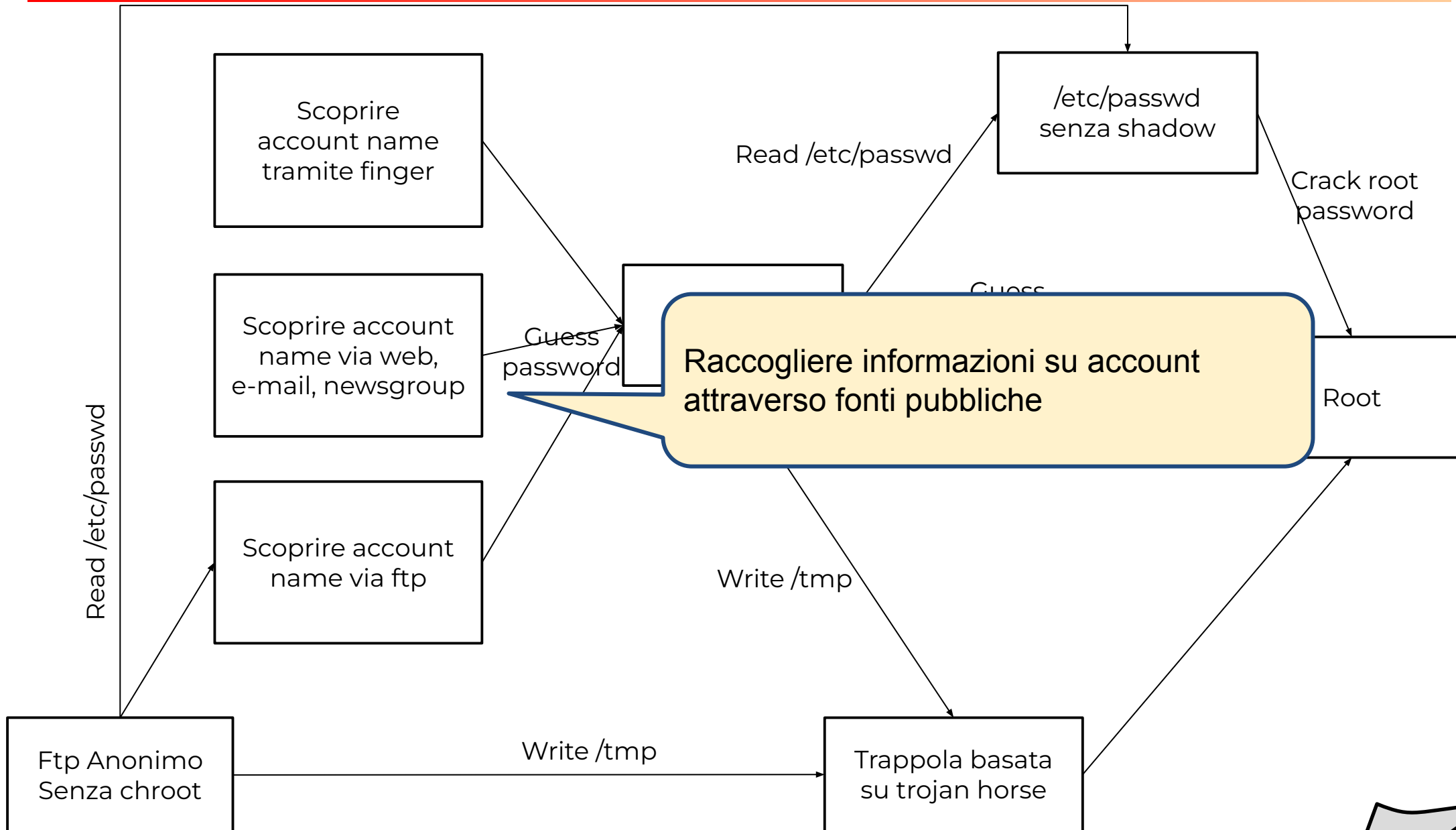
Percorsi di attacco

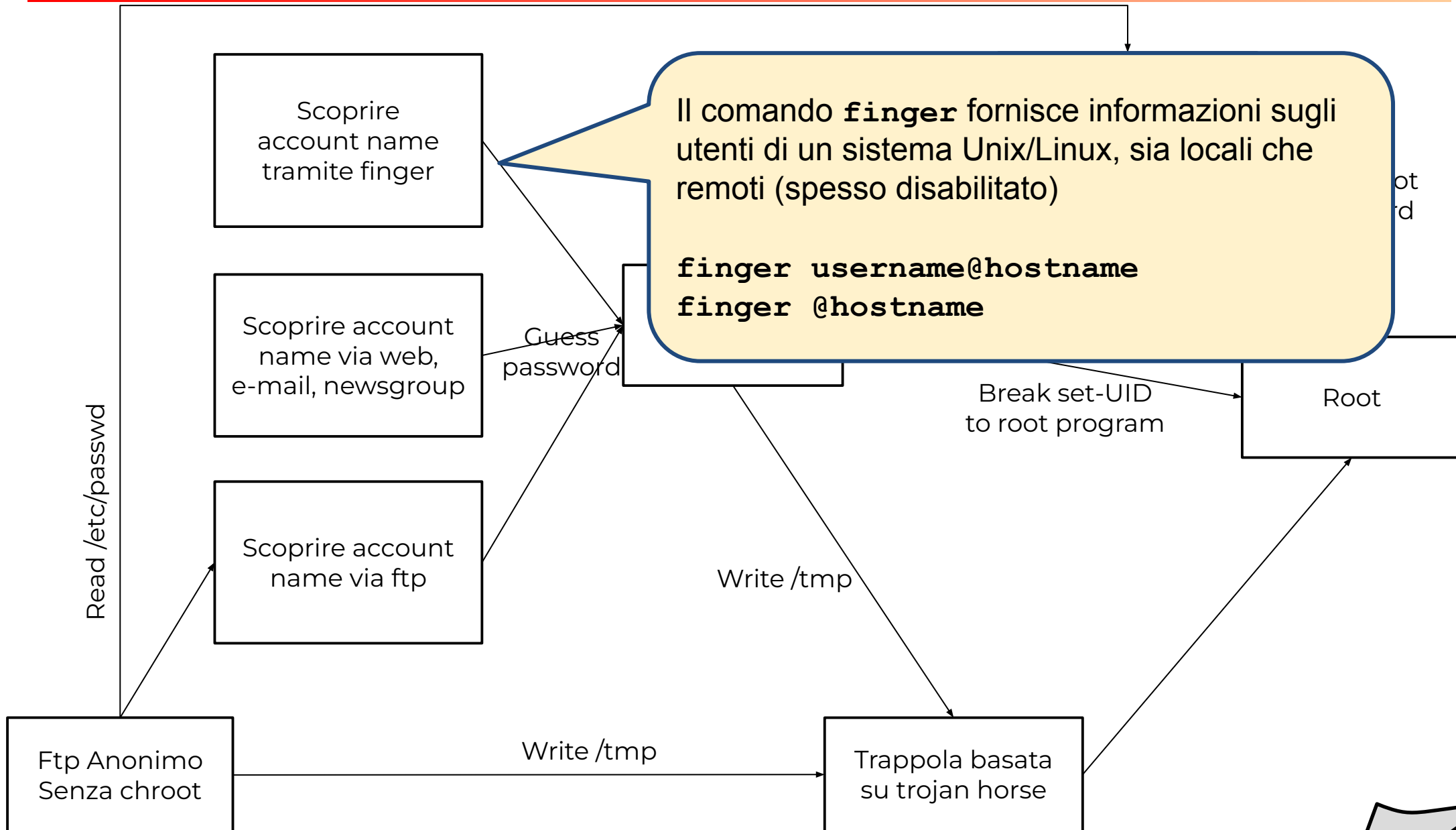
Attacchi



Percorsi di attacco

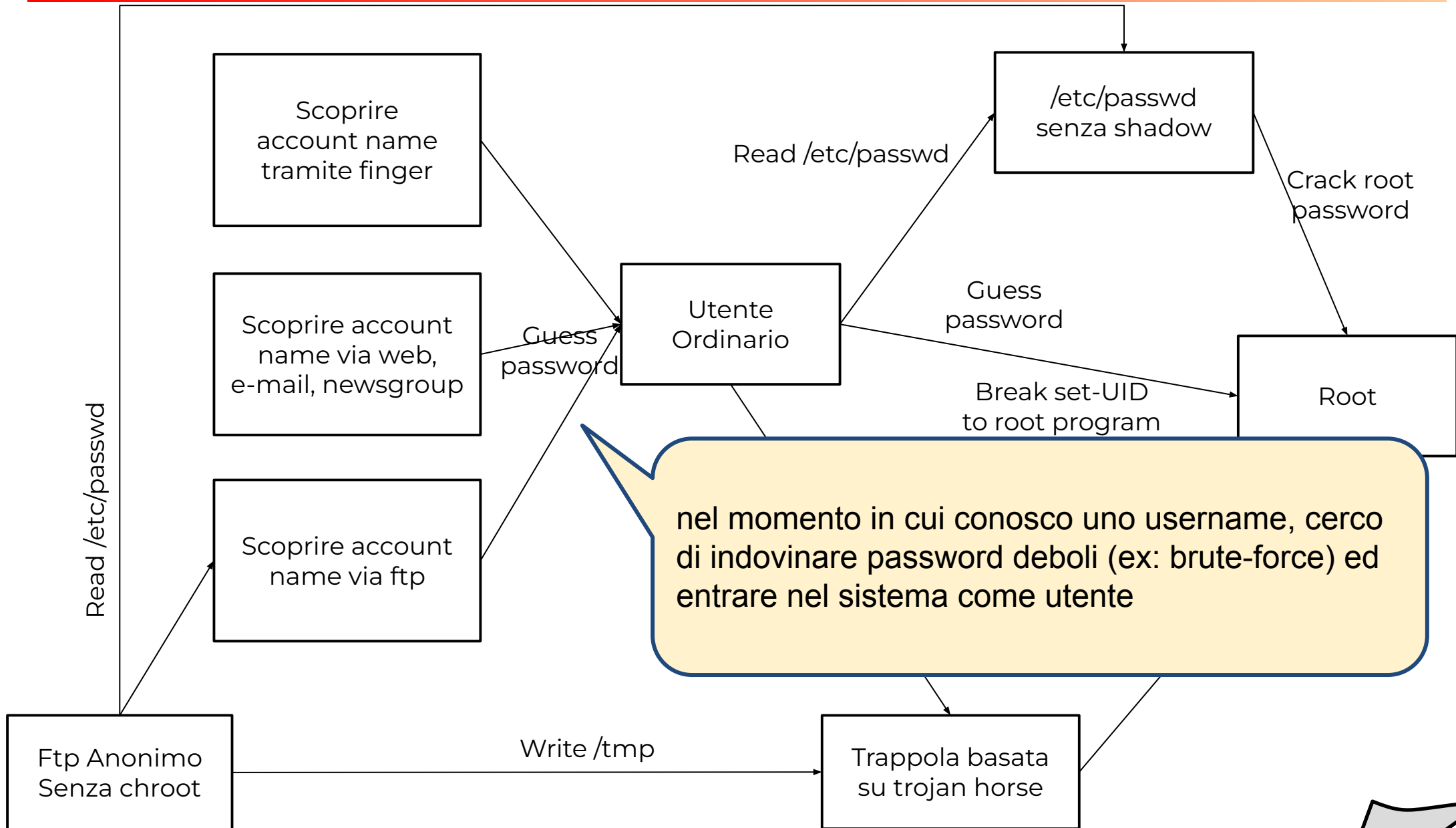
Attacchi





Percorsi di attacco

Attacchi



Scoprire
account name
tramite finger

Read /etc/passwd

/etc/passwd
senza shadow

Crack root
password

root

Il file `/etc/passwd` contiene informazioni sugli account utente del sistema.

Struttura del file

Ogni riga rappresenta un utente e contiene 7 campi separati da due punti (:):

`username:password:UID:GID:GECOS:home_directory:shell`

password:

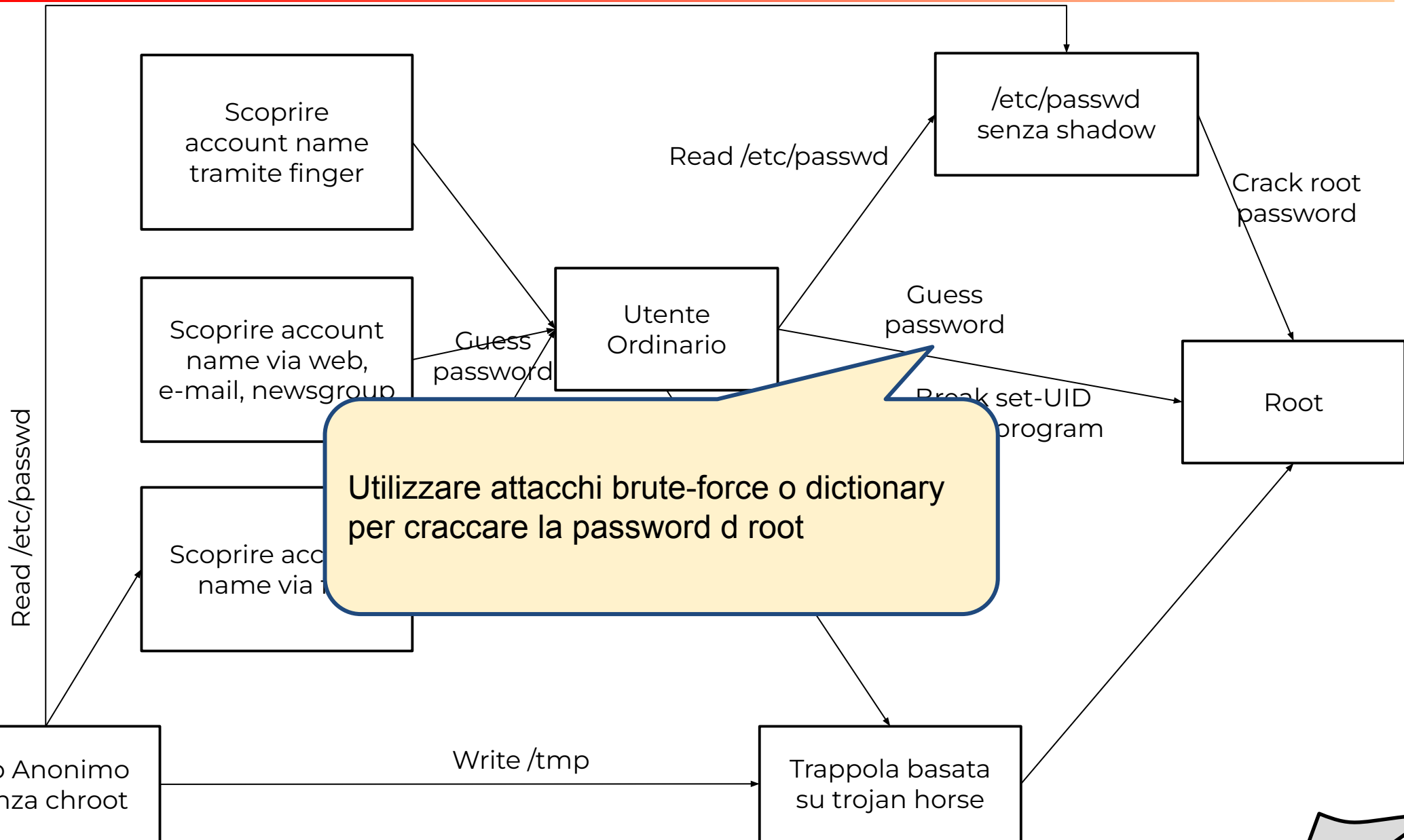
- Nei sistemi moderni: solitamente x o * (password memorizzata in `/etc/shadow`)
- Nei sistemi più vecchi: hash della password (vulnerabilità di sicurezza!)

UID, GID: user e group id

GECOS: info generiche su utente (nome, ufficio, num tel, ...)

Percorsi di attacco

Attacchi



Il bit **setUID** (Set User ID) è un permesso speciale che permette a un programma di essere eseguito con i privilegi del proprietario del file anziché dell'utente che lo esegue.

Esempio legittimo - comando passwd (cambio passwd):

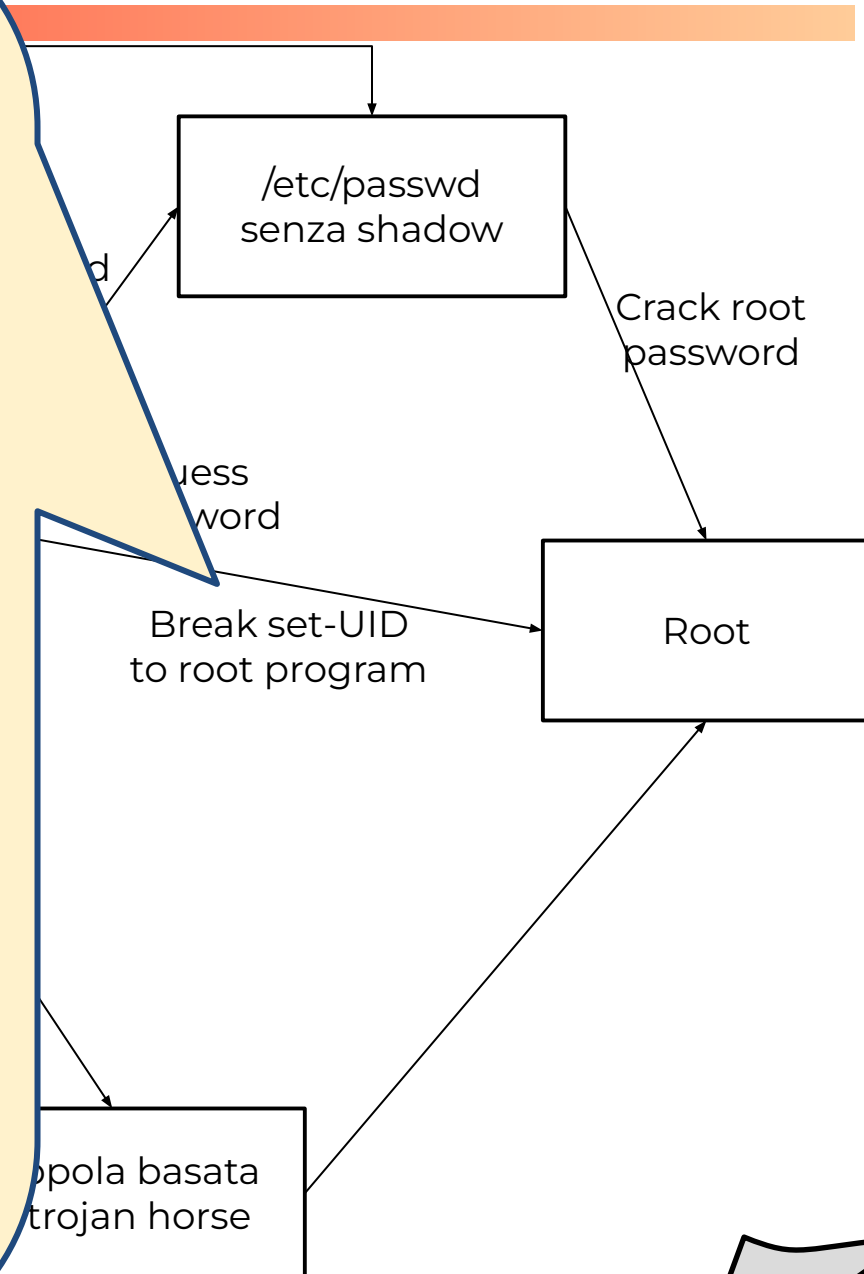
```
ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 68208 passwd
```

Nota la **s** al posto della **x** nei permessi del proprietario. Questo significa:

- Il file appartiene a root
- Quando un utente normale esegue passwd, il programma viene eseguito con privilegi di root
- È necessario perché passwd deve modificare **/etc/shadow** (accessibile solo a root)

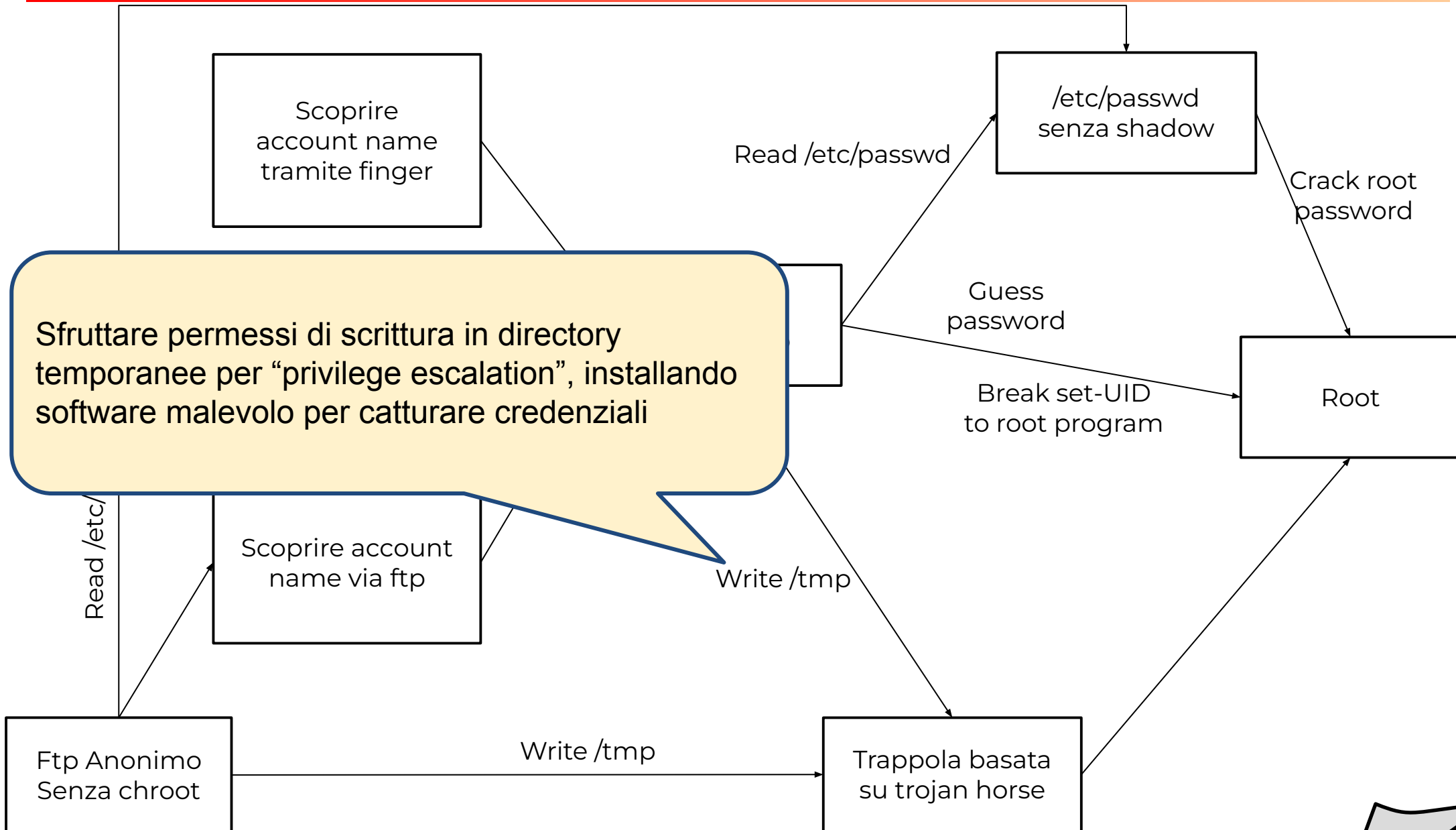
Il problema di sicurezza

Se un programma **setUID** contiene vulnerabilità, un attaccante può sfruttarle per ottenere i privilegi del proprietario del file.



Percorsi di attacco

Attacchi



- **Individuare il sistema da attaccare**
 - Tramite web, perché si è ricevuto un mail, etc.
 - Ad esempio, la ditta yahuu
- **Cercare di carpire qualche prima informazione**
 - Tramite ping verifichiamo se esiste

```
$ ping www.yahuu.com
```

```
PING www.yahuu.com (217.12.3.11) 56(84) bytes of data.
```

```
64 bytes from www.yahuu.com (217.12.3.11): icmp_seq=1 ttl=242 time=88.7 ms
```

```
64 bytes from www.yahuu.com (217.12.3.11): icmp_seq=2 ttl=242 time=89.5 ms
```

- In effetti esiste...

- Cercare di carpire qualche prima informazione

- Proviamo con finger

```
$ finger root@www.yahuu.com
```

```
[www.yahuu.com]
```

```
Login: root      Name: root
```

```
Directory: /root Shell: /bin/tsch
```

```
Last login Thu Jan  8 00:12 (CET) on tty2
```

- Ottimo! Ora sappiamo
 - Finger è abilitato (oggi giorno è molto raro)
 - L'utente root non è al momento connesso

- **Cerchiamo ora di scoprire qualche nome utente**

- Abbiamo bisogno di un dizionario di nomi
- Possiamo utilizzare finger per scoprire se il nome esiste

```
$ finger dennis
```

```
Finger: dennis: no such user
```

```
$ finger paul
```

```
Login: paul
```

```
Name: Paul Hughes
```

```
Directory: /home/paul
```

```
Shell: /bin/bash
```

```
Office: 789-123456
```

```
Last login Wed Jan 7 19:05 (CET) on pts/4 from ...
```

- **Una volta scoperto il nome utente**
 - Possiamo utilizzare un meccanismo automatico per tentare di indovinare la password
 - Funziona quando la password è banale (vedi lucidi successivi)
- **Il passo successivo è cercare di acquisire privilegi da superutente**
 - Probabilmente: indovinare la password di root non avrà successo
 - Proviamo a vedere cosa contiene la variabile **\$PATH** di root:

```
cat /root/.bash_profile
```

```
export PATH="/usr/local/bin:/etc:.: $PATH"
```

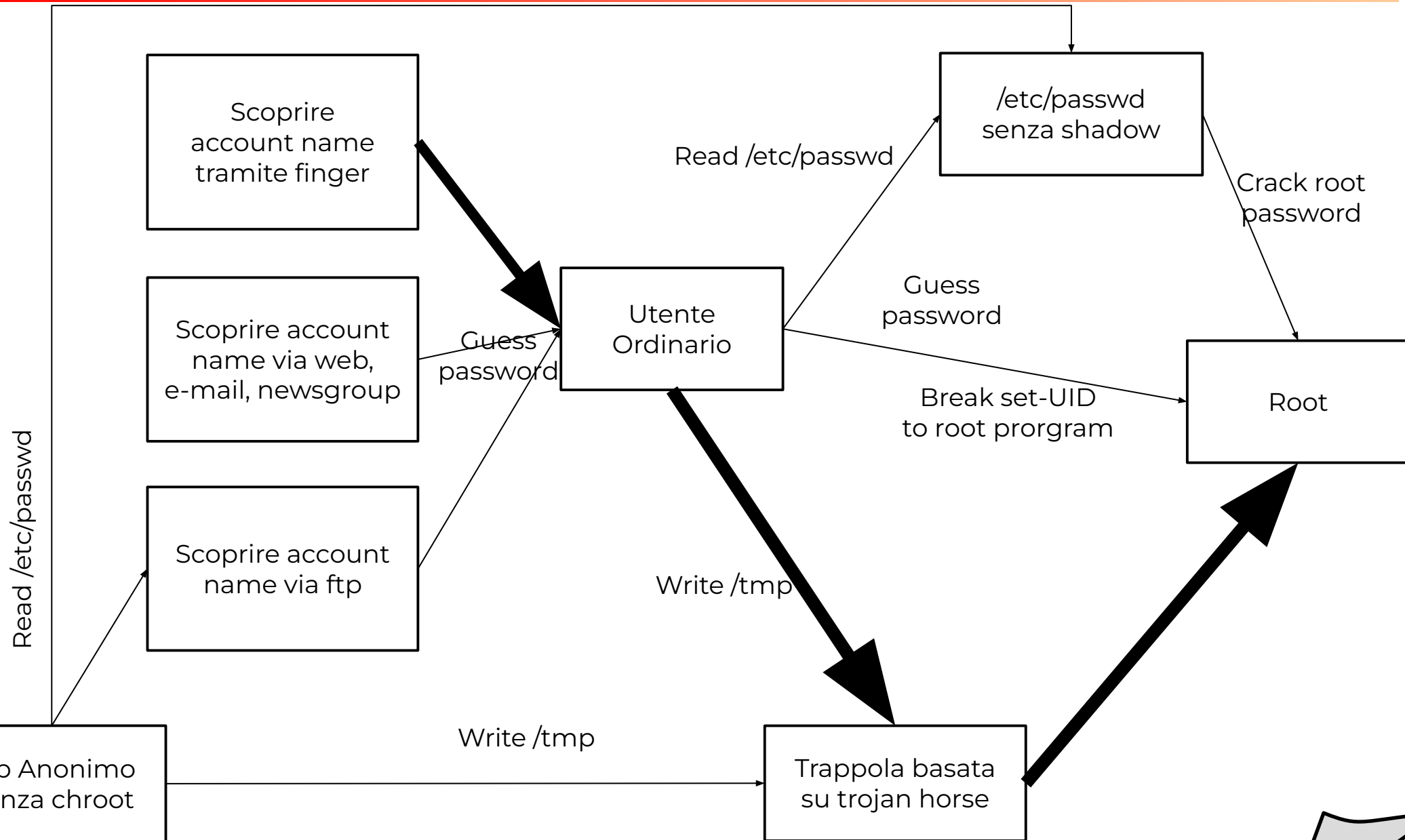
- Ottimo! Il superutente sprovveduto permette di eseguire qualunque programma nella directory corrente

■ Prepariamo una trappola

- Ad esempio in `/tmp`
 - Si crea un file di dimensioni enormi, che in qualche modo catturi l'attenzione del superutente
 - Si installa un insieme di *trojan horse* quali ad esempio ls, vi, etc.
 - Quando vengono eseguiti, questi programmi possono eseguire qualunque cosa l'attaccante voglia

Percorsi di attacco

Attacchi





Vulnerabilità del software

Quali sono i possibili “difetti” del software?

- **Errori di programmazione:**
 - Violazione dei limiti (buffer overflow)
 - Errori tempo controllo / tempo utilizzo
- **Codice maligno**
 - Cavalli di troia (trojan horse)
 - Bombe logiche
 - Backdoor
 - Virus, worm



Vulnerabilità del software - Errori di programmazione

Buffer Overflow

■ Qual è la tecnica:

- L'idea generale è quella di fornire ad un programma (server) un insieme di dati di dimensioni superiori a quella prevista (buffer overflow)

■ Cosa succede?

- Nel "migliore" dei casi, il programma va in crash (segmentation fault)
- Nei casi peggiori, è possibile che l'attaccante prenda il controllo della macchina

■ Un po' di statistica...

- **Più del 50% degli incidenti riportati al CERT sono dovuti a buffer overflow**

Buffer Overflow

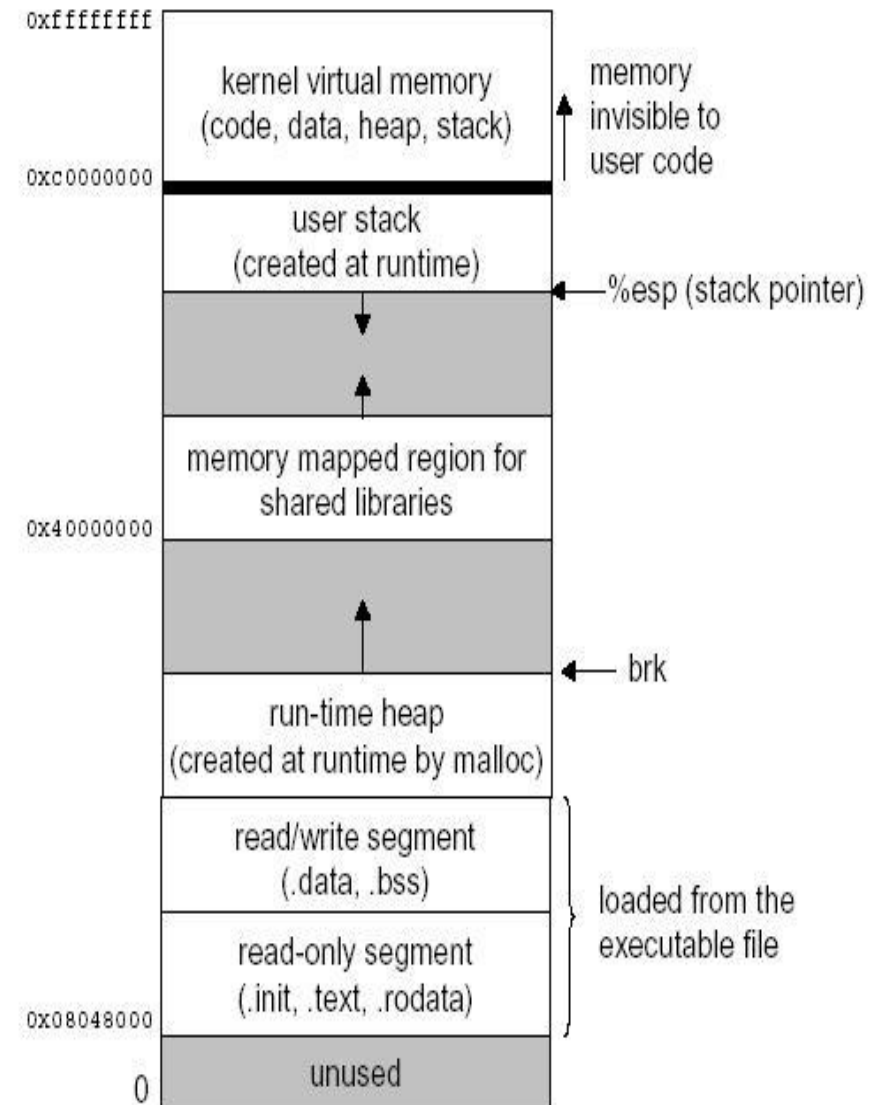
Vulnerabilità

■ Realizzare un exploit tramite buffer overflow:

- I principi base uguali per ogni architettura
- Ma si richiede conoscenza del sistema operativo e della CPU target

■ Background necessario

- Organizzazione della memoria
 - Segmento codice
 - Segmento dati
 - Segmento stack
- Le funzioni C e lo stack
- Un po' di linguaggio macchina



■ Funzioni C

```
int main()  
{  
    int buffer[10];  
    buffer[20] = 5;  
}
```

■ Il buffer

- Viene creato nello stack (è "temporaneo" per la funzione C)
- Non vengono effettuati alcun tipo di controllo di dimensione
- La seconda istruzione sovrascrive altri dati nello stack

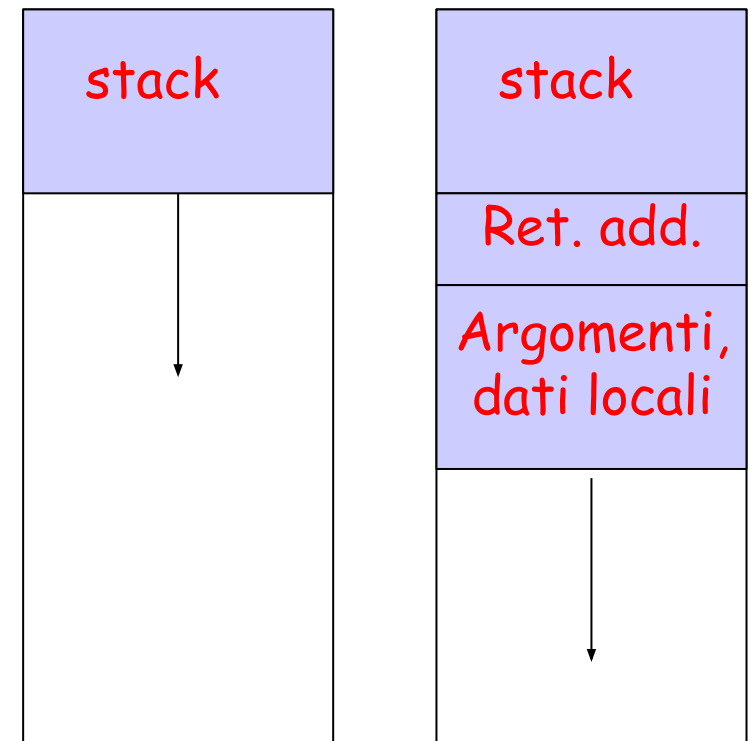
Time to code!



- `buffer_overflow.c`

■ Un po' di linguaggio macchina

- Ad ogni chiamata di funzione, nello stack vengono collocati
 - Indirizzo di ritorno
 - Eventuali argomenti della funzione
 - Eventuali dati locali



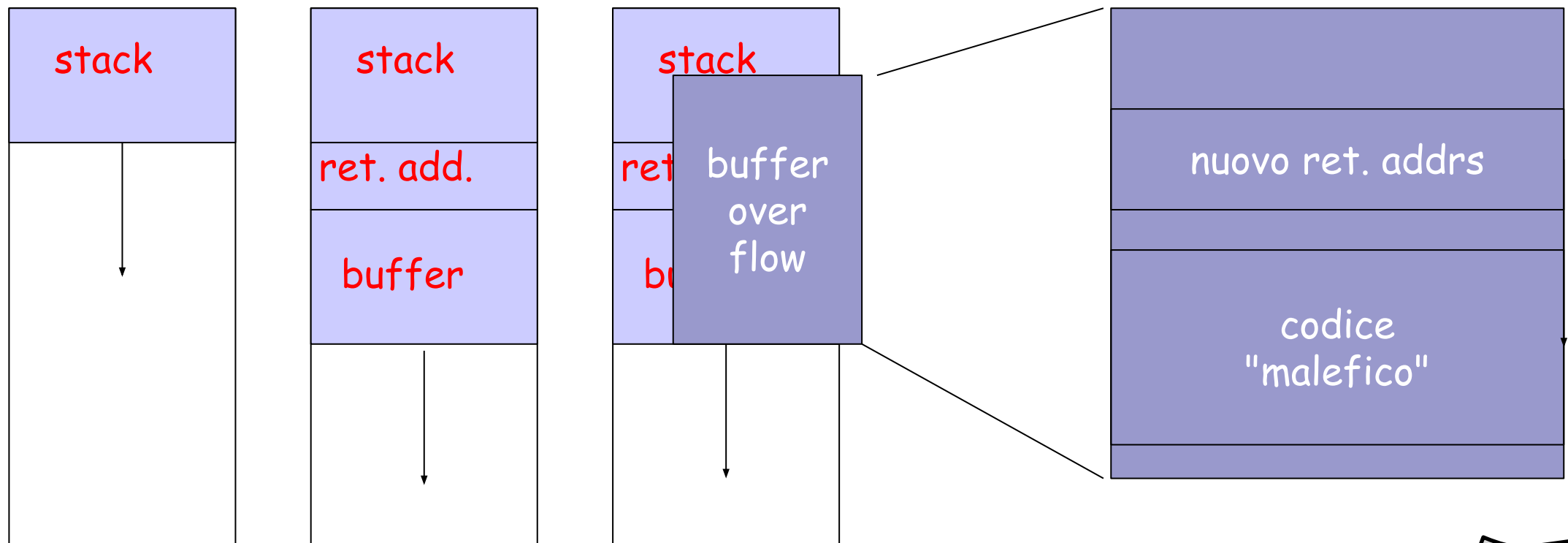
■ Quale tipo di codice porta ad un buffer overflow?

- Utilizzazione di funzioni di libreria deboli che non controllano la dimensione dei loro argomenti
 - **gets()** - legge lo standard input
 - **strcpy()** - copia una stringa in un'altra
 - **sprintf()** - formatta una stringa in un buffer

■ Esempio:

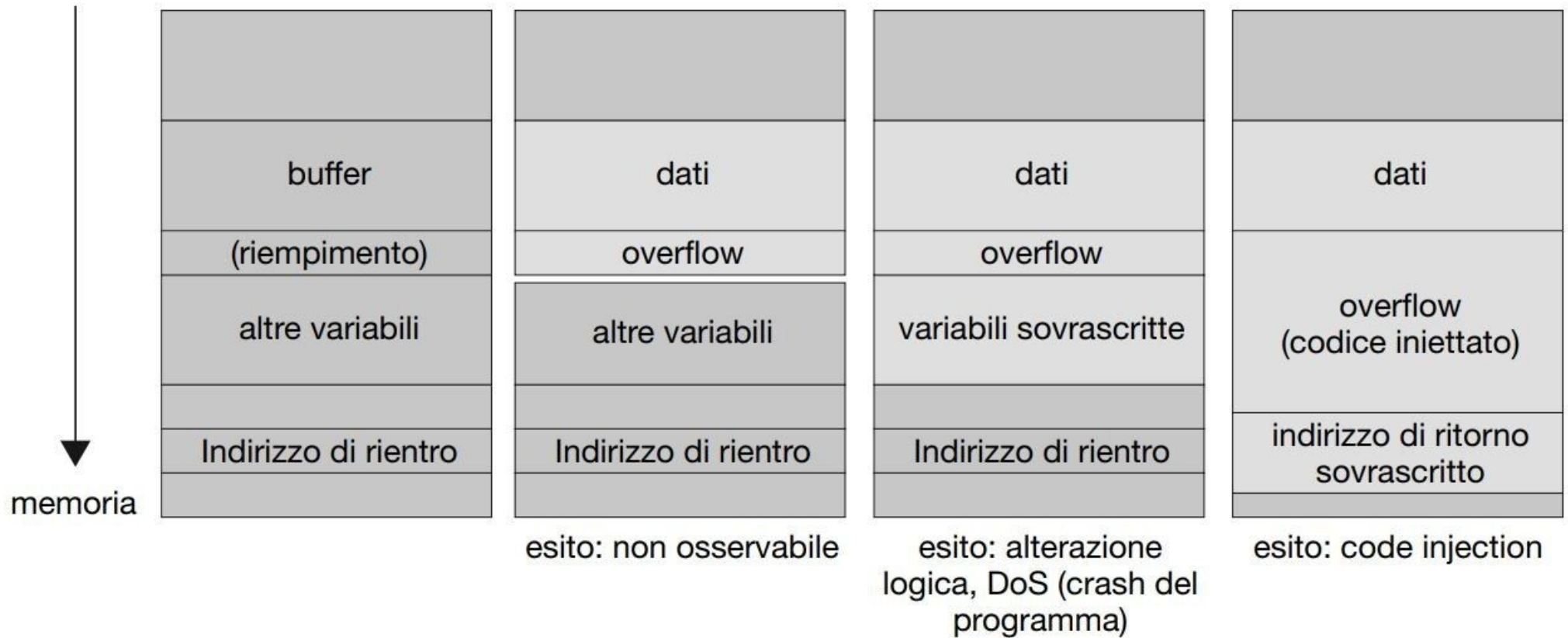
```
int main()  
{  
    char buffer[10];  
    printf("Inserisci il CAP");  
    gets(buffer);  
}
```

- **L'idea generale è quella di riempire il buffer**
 - Con codice "maligno"
 - Un nuovo indirizzo di ritorno che punti a tale codice maligno



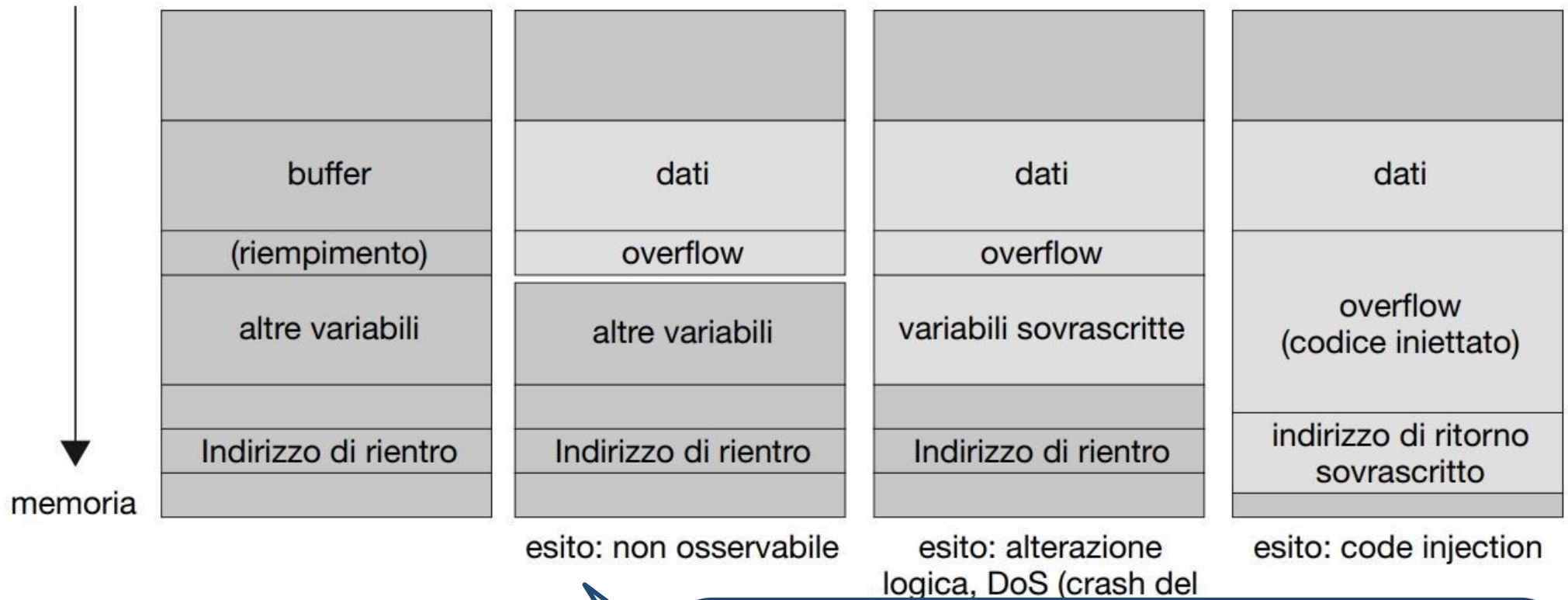
Buffer Overflow

Vulnerabilità



Buffer Overflow

Vulnerabilità



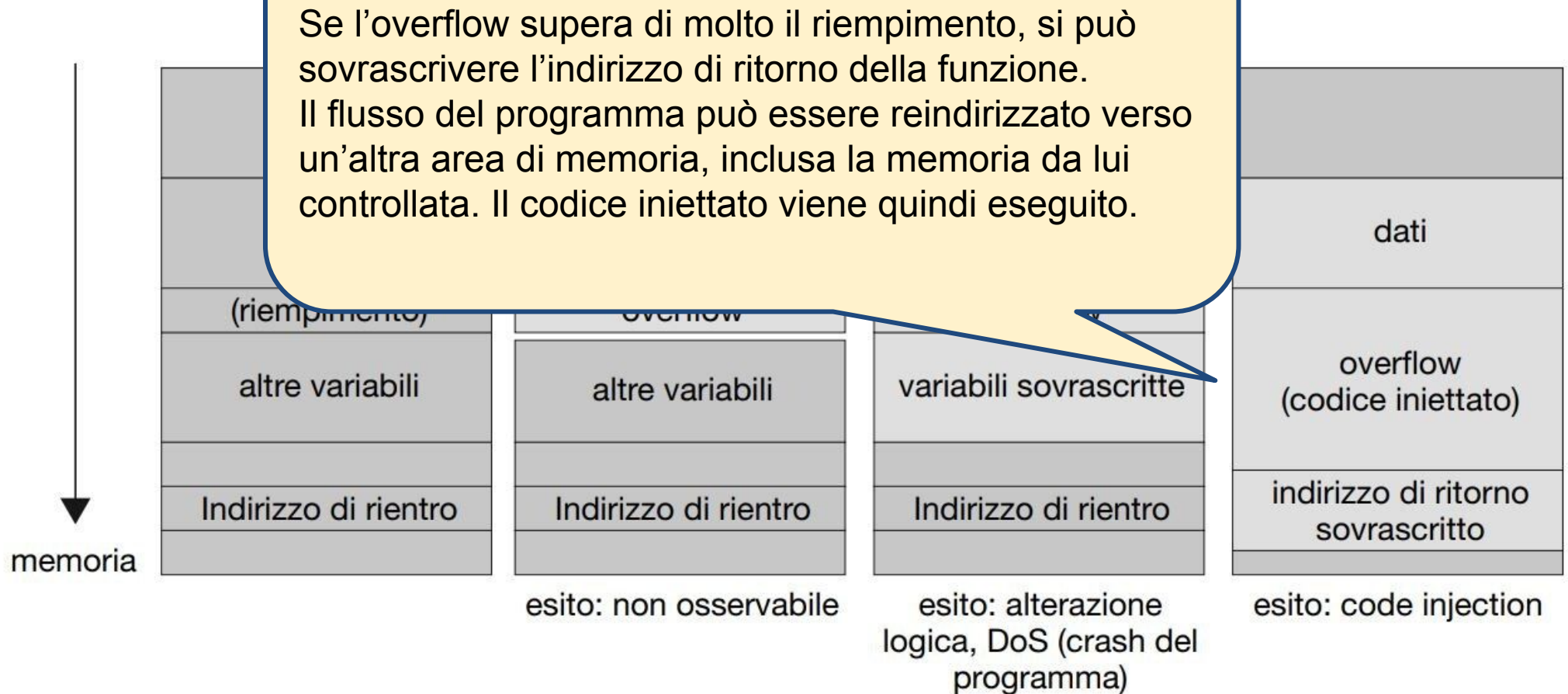
Se l'overflow è molto piccolo (se va di poco oltre la dimensione `BUFFER_SIZE`), ci sono buone probabilità che passi del tutto inosservato

Se l'overflow supera il riempimento, la successiva variabile automatica presente nello stack viene sovrascritta dal contenuto in eccesso.
In assenza di controlli questo overflow può causare un arresto anomalo del programma, poiché un valore imprevisto in una variabile può portare a un errore non correggibile



Buffer Overflow

Vulnerabilità



■ Contromisure

• Scrivere codice sicuro

- I buffer overflow sono il risultato di "sloppy programming"
- Evitare le funzioni citate, consultare i manuali delle librerie, etc.

■ Contromisure

• Impedire l'esecuzione di codice che si trova nello stack

- Lo stack dovrebbe contenere solo dati (variabili locali, parametri, indirizzi di ritorno)
 - NON dovrebbe mai contenere codice eseguibile
 - Se si marca lo stack come "non-eseguibile", anche se l'attaccante ci inserisce shellcode, non può essere eseguito
- PROBLEMA: in Linux, si utilizzano "funzioni trampolino"
 - piccoli pezzi di codice che vengono messi temporaneamente nello stack per gestire i signal handler
- SOL: Esistono delle versioni sicure del kernel linux
 - Eliminano la necessità delle funzioni trampolino tradizionali

■ Contromisure

• Controlli a tempo di compilazione

- Patch per compilatori che aggiungono controlli sulla dimensione dei buffer
 - Tendono a far crescere il codice e diminuire le prestazioni
- Patch per compilatori che controllano unicamente l'indirizzo di ritorno
 - Copiato in una posizione sicura
 - StackGuard, StackShield
- Nota: richiede ri-compilazione delle applicazioni

• Controlli a run-time

- Verificano a run-time le dimensioni degli stack frame
- Se gli stack frame presentano qualche errore, terminano il programma
- Librerie pre-caricate tipo libsafe



Vulnerabilità del software - Errori di programmazione

Errori tempo controllo / tempo utilizzo

Time-of-Check to Time-of-Use (TOCTOU)

Definizione:

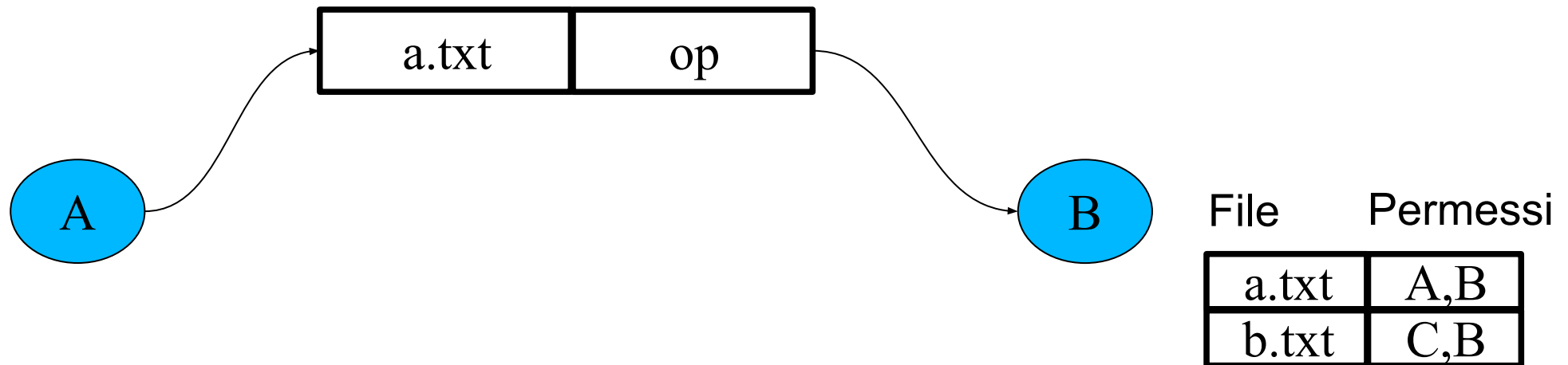
- Vulnerabilità che si verifica quando esiste un intervallo temporale tra il **controllo** di una condizione e il suo **utilizzo**
- Durante questo intervallo, lo stato del sistema può cambiare
- L'attaccante può sfruttare questa finestra temporale per modificare le condizioni

Caratteristiche principali:

- Race condition tra processi
- Violazione dell'atomicità delle operazioni
- Problema di consistenza temporale

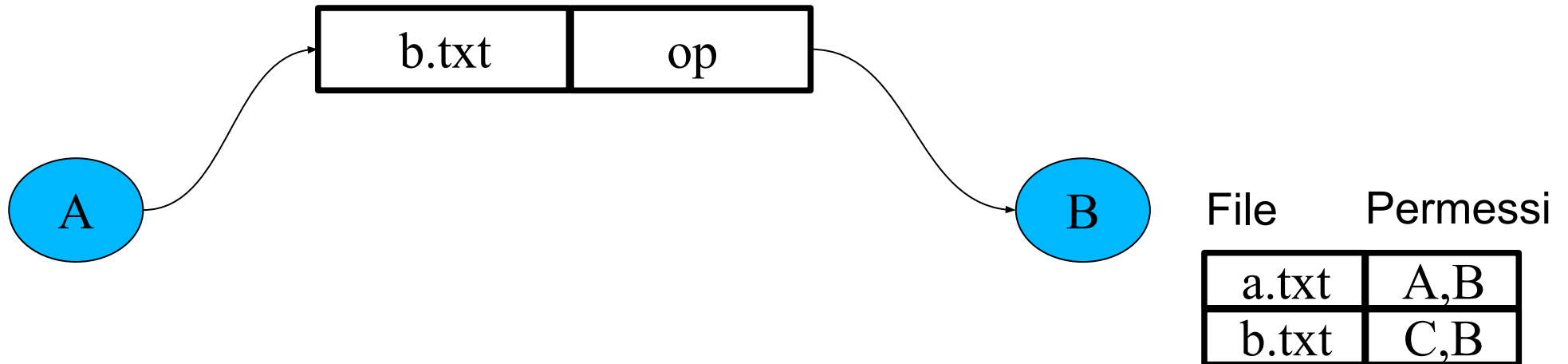
■ Fase 1:

- L'entità A chiede all'entità B di eseguire l'operazione **op** sul file **a.txt**
- B autorizza l'operazione in base alle informazioni di accesso



■ Fase 2:

- L'entità A modifica il file da accedere nel periodo di tempo che intercorre fra il controllo e l'utilizzo
 - ex: crea link nominato a.txt che punta a b.txt
- B effettua l'operazione op su b.txt



■ Fase 1: Il Controllo

L'utente Alice chiede di leggere "documento.txt"

Il sistema controlla:

`ls -l documento.txt -rw-r--r-- alice users documento.txt`

Alice può leggere ✓

Sistema: "OK, Alice può leggere documento.txt"

■ Fase 2: L'Attacco

MENTRE il sistema sta per aprire il file... Alice velocemente fa:

`rm documento.txt` *# Cancella il file originale*

`ln -s /etc/passwd documento.txt` *# Crea un link simbolico al file delle password*

Ora "documento.txt" punta a /etc/passwd!

■ Fase 3: L'Utilizzo

Il sistema esegue l'operazione autorizzata:

`cat documento.txt`

Ma ora legge /etc/passwd invece del documento originale!

// Codice vulnerabile in un programma con setuid

```
int main() {  
    // TIME OF CHECK  
    if (access("/tmp/userfile", R_OK) == 0) {  
        printf("File accessibile, procedo...\n");  
        sleep(1); // GAP TEMPORALE CRITICO!  
  
        // TIME OF USE  
        FILE *f = fopen("/tmp/userfile", "r");  
        // Legge il file...  
    }  
}
```

Errori tempo controllo / tempo utilizzo: Esempio

Vulnerabilità

// Codice vulnerabile in un prog

```
int main() {
```

```
    // TIME OF CHECK
```

```
    if (access("/tmp/userfile", R_OK) == 0) {
```

```
        printf("File accessibile, procedo...\n");
```

```
        sleep(1); // GAP TEMPORALE CRITICO!
```

```
    // TIME OF USE
```

```
    FILE *f = fopen("/tmp/userfile", "r");
```

```
    // Legge il file...
```

```
}
```

```
}
```

1. Attaccante crea
`/tmp/userfile` con
permessi normali

2. Attaccato esegue
questo codice

3. Attaccante sostituisce
il file con link simbolico a
`/etc/passwd`

Escalation dei Privilegi:

- Accesso a file riservati
- Modifica di configurazioni di sistema
- Bypass dei controlli di accesso

Violazione dell'Integrità:

- Modifica non autorizzata di dati
- Corruzione di file di sistema
- Alterazione di log di sicurezza

Denial of Service:

- Eliminazione di file critici
- Riempimento dello spazio disco
- Crash di applicazioni

1. Controllo Non Atomico:

- Separazione tra check e use
- Gap temporale sfruttabile

2. Risorsa Modificabile:

- Oggetto condiviso tra processi
- Permessi di modifica per l'attaccante

3. Race Condition Vincibile:

- Timing predicibile o ripetibile
- Finestra temporale sufficiente

4. Privilegio della Vittima:

- Programma con privilegi elevati
- Accesso a risorse protette

1. Eliminare il Gap Temporale

// CORRETTO: Uso diretto senza check preliminare

```
FILE *f = fopen(filename, "r");
```

```
if (f == NULL) {  
    perror("Impossibile aprire file");  
    return -1;  
}
```

// Procede con l'uso del file descriptor

2. Operazioni Atomiche

// CORRETTO: Usa open() con flags appropriati
// O_NOFOLLOW: se il path è un link simbolico, fallisce

```
int fd = open(filename, O_RDONLY | O_NOFOLLOW);
```

```
if (fd == -1) {  
    perror("Errore apertura");  
    return -1;  
}
```

// Procede con l'uso del file descriptor

3. Locking Appropriato

// Lock esclusivo sulla risorsa

```
if (flock(fd, LOCK_EX) == -1) {  
    perror("Cannot lock file");  
    return -1;  
}
```

Usa operazioni atomiche

- `open()` invece di `access()` + `fopen()`
- Flags appropriati (`O_EXCL`, `O_NOFOLLOW`)

Valida sui file descriptor

- `fstat()` invece di `stat()`
`int stat(const char *path, struct stat *buf);`
stat() ritorna le informazioni sul file specificato da path e le mette in buf
`int fstat(int fd, struct stat *buf);`
fstat() è identico a stat(), ma il file è identificato dal descriptor fd
- Mantieni aperti i descriptor necessari



Vulnerabilità del software - Codice Maligno

■ Definizione

- Sono programmi che replicano le funzionalità di programmi di uso comune o programmi dall'apparenza innocua ma che contengono codice malevolo

■ Tipicamente

- Catturano informazioni e le inviano al creatore del programma
 - Informazioni critiche per la sicurezza del sistema
 - Informazioni “private” dell'utente
- Compromettono o distruggono informazioni importanti per il funzionamento del sistemi

■ Esempi:

- Un applicativo può leggere informazioni relative al vostro sistema o al software installato e inviarle alla ditta creatrice del programma.....
 - Spyware in programmi di file sharing
- Inserimento di programmi nelle directory normalmente accessibili tramite **\$PATH** con il nome "simile" a quello di programmi noti sperando che un utente commetta un errore di battitura
 - Esempio: **dri** (al posto di `dir`), **la** (al posto di `ls`)
- Login spoofing

■ Descrizione

- Il creatore di un software utilizzato internamente in una compagnia inserisce un software che può attivarsi sotto particolari condizioni
 - bomba logica: destinata ad attivarsi ad un certo timeout
 - finché il dipendente è presente, può periodicamente disinnescarla (o ritardarla)
- Esempio di condizione
 - Il creatore viene licenziato e il suo nome scompare dal database dei salari
 - Non potendo più accedere al sistema, non può evitare che la bomba logica si attivi

■ Contromisure

- Analisi del codice sorgente (code reviews)
- Ma si veda l'articolo "Reflections on Trusting trust", di D. Knuth.

■ Descrizione

- Il creatore di un software può deliberatamente lasciare “porte di servizio” per entrare aggirando i sistemi di protezione

■ Alcuni esempi

- Il film War Games!
- Back Orifice, Sobig, MyDoom
- Nov. 2003: un tentativo di inserire un backdoor nel kernel linux
 - Sventato!
- Cosa succede nei sistemi proprietari?

■ Contromisure

- Analisi del codice sorgente (code reviews)

■ Un virus

- E' un frammento di programma che può infettare altri programmi non maligni modificandoli

■ Un worm (batterio)

- E' un programma che diffonde copie di se stesso in una rete

■ Quali differenze...

- I worm operano sulle reti, i virus possono usare qualunque supporto
- I worm sono programmi autonomi, i virus infettano programmi esistenti
- Non vi è più una vera distinzione, oramai...

L'attività di un virus si divide in due fasi:

■ Riproduzione (infezione)

- Un buon virus deve bilanciare infezione e possibilità di essere scoperto
- Esistono tecniche per cercare di nascondersi

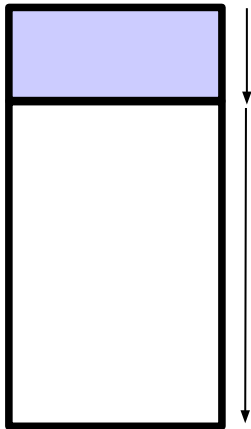
■ Attivazione

- Può essere scatenata da uno specifico evento
- In generale, un virus attivato compie azioni dannose (o semplicemente consuma risorse)

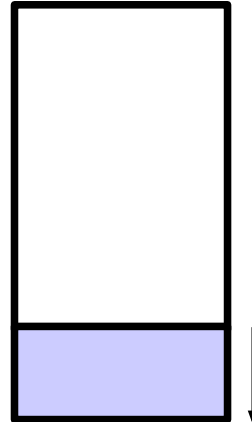
■ Come si diffondono i virus?

- Un virus deve essere eseguito per potersi diffondere/attivare
- Come può forzare la propria esecuzione?
 - Accodandosi ad un programma esistente
 - EXE/COM in MS-DOS
 - Portable Executable in Windows
 - ELF in Linux
 - Sfruttando un meccanismo di bootstrap
 - boot sector
 - master boot record
 - Accodandosi a file dati che permettono l'esecuzione di codice (anche in maniera automatica)
 - posta elettronica
 - macro virus in Office

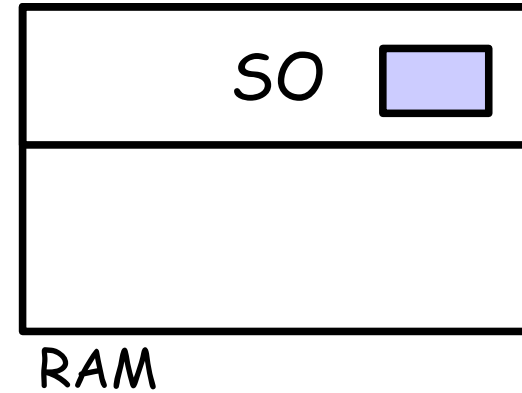
Virus "accodato"



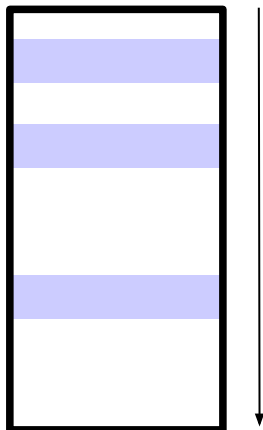
Macro virus



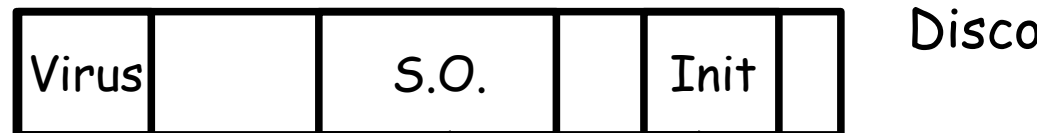
Virus residenti



Virus "integrato"



Virus in boot sector / MBR



■ Come individuare un virus?

- I virus non sono invisibili
 - Devono essere memorizzati
 - Eseguono azioni “tipiche”
- E' possibile estrapolare una *definizione* o *firma* che caratterizza il virus
- Questa firma viene cercata dai programmi anti-virus

■ Esempi:

- Il worm *code-red* inizia con:
`/default.ida?NNNNN`

■ Come si nascondono i virus?

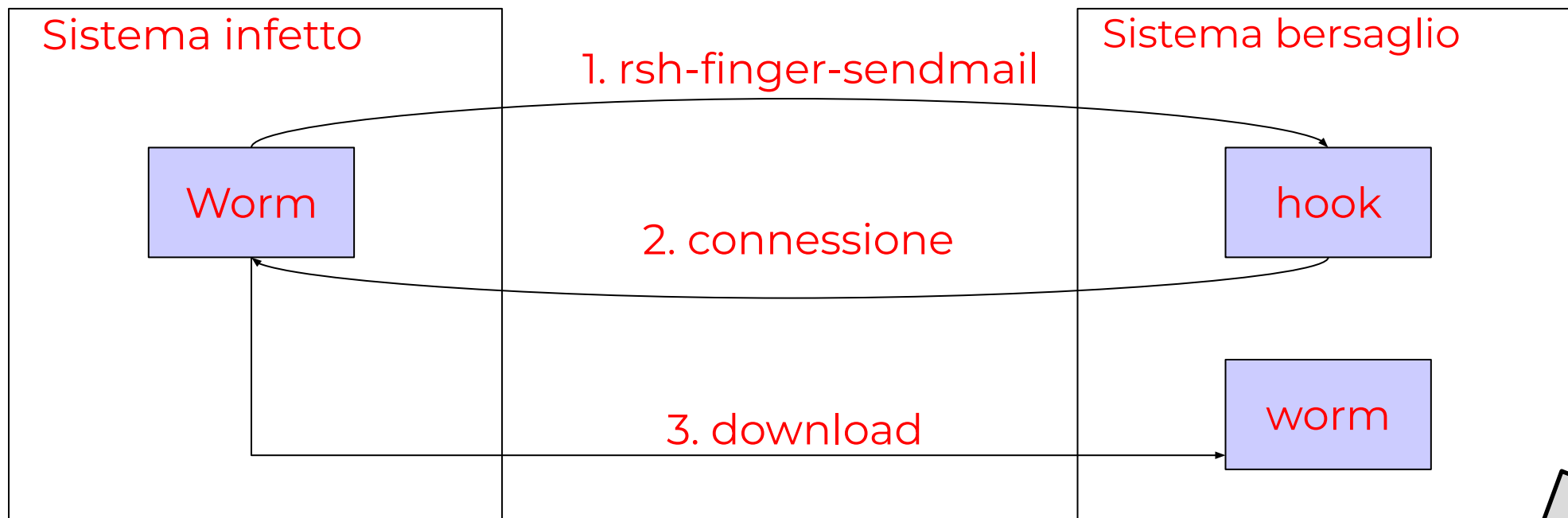
- Virus “stealth”
- Virus “polimorfici”

■ Un po' di numeri:

- Più di 100.000 virus
- Relativamente pochi (~1000) sono “attivi” in un certo periodo

■ Internet Worm di Morris

- Il primo worm conosciuto (1988)
- Sfruttava alcuni bug di sistemi di comunicazione
- Sostanzialmente benigno, bloccò però la rete per un giorno
- Fu la “goccia” che fece nascere il CERT





Autenticazione

■ Definizione

- *L'operazione di **associare ad un utente un'identità***

■ Note

- E' il problema alla base di tutti i meccanismi di protezione
- Senza autenticazione, tutti i meccanismi che vedremo in seguito sarebbero completamente inutili

■ Meccanismi per ottenere autenticazione

- Basati su qualcosa che l'utente "conosce"
- Basati su qualcosa che l'utente "ha"
- Basati su qualcosa che l'utente "è"

- **Autenticazione basata su password**
 - è ovviamente la più utilizzata
- **Problemi**
 - Possibilità che la password venga "indovinata"
 - Scarsa cultura della sicurezza
 - Password banali
 - Post-it con la password attaccata allo schermo
 - Possibilità che la password venga "carpita"
 - un intruso può sbirciare chi sta digitando la password
 - login spoofing
 - sniffing di rete

Top 10 Globale

- 123456 (utilizzata 21.627.656 volte)
- admin (in uso 21.030.012 volte)
- 12345678 (usata 8.274.408 volte)
- 123456789 (utilizzata 5.673.712 volte)
- 12345 (in uso 3.950.777 volte)
- password (usata 3.545.119 volte)
- Aa123456 (utilizzata 2.520.728 volte)
- 1234567890 (in uso 1.418.939 volte)

■ Come memorizzare le password

- Nei sistemi più obsoleti, le password venivano memorizzate in chiaro in un file, protetta dai normali meccanismi di protezione
- Problema: troppe persone possono accedere a questo file

■ Password codificate

- Si utilizza una funzione one-way
- Il file delle password contiene le password codificate dalla funzione one-way scelta
- Quando viene effettuato il login, si codifica la password e si confronta il risultato con il valore contenuto nel file delle password
- Password file in Unix/Linux: **/etc/passwd**

- Il file `/etc/passwd` contiene informazioni sugli account utente del sistema.

Struttura del file

Ogni riga rappresenta un utente e contiene 7 campi separati da due punti (:):

`username:password:UID:GID:GECOS:home_directory:shell`

`password:`

- Nei sistemi moderni: solitamente `x` o `*` (password memorizzata in `/etc/shadow`)
- Nei sistemi più vecchi: hash della password (vulnerabilità di sicurezza!)

UID, GID: user e group id

GECOS: info generiche su utente (nome, ufficio, num tel, ...)

password

- Password file in Unix/Linux: `/etc/passwd`

▪ Attaccare lo schema precedente (*cracking*)

- Si utilizza un dizionario di password banali
- Le password nel dizionario vengono codificate facendo uso della stessa funzione one-way
- Le password codificate vengono confrontate con quelle contenute in file di password

Dizionario

in chiaro

codificate

280771	al47r33km
password	ldcfjoiu3
rossi	rlekek1139
franchi	8kfdjkj kf
gelato	39u4jrk0

Password file

```
rossi:8kfdjkj kf:500:500:...  
franchi:rlekek1139:501:501:...
```

■ Il test di Morris-Thompson

- La password è più corta di sette caratteri?
- Contiene solo 7-8 caratteri alfabetici (tutti minuscoli o tutti maiuscoli)
- La password è una qualsiasi parola da un dizionario
 - Anche con l al posto della i, 0 al posto della o, 3 al posto di e
 - Anche scritta al contrario
- La password è una data di nascita

■ Regole da seguire:

- Almeno 8 caratteri, un carattere per ognuna di queste categorie:
 - Lettere minuscole
 - Lettere maiuscole
 - Cifre numeriche
 - Caratteri speciali: , . ; : - _ # [] ? ^ + * ' ` ~ { } ()

■ Alcune statistiche

- Morris-Thompson, 1979 - **86%** delle password identificate
Password raccolte in vari sistemi Unix dell'epoca
- Klein, 1990 - **25%** delle password identificate
Password raccolte da vari sistemi Unix in UK/USA
- Kabay, 1997 - **82%** delle password identificate
Password utilizzate nel financial district di Londra

■ Utilizzo del meccanismo di *salt*

- Prima di essere criptate e memorizzate nel file di password, le password vengono concatenate con un numero casuale (*salt*)
- Un salt diverso per utente, che viene memorizzato in chiaro nel file delle password
- In questo modo, il dizionario deve essere codificato con tutti i possibili salt, richiedendo un tempo maggiore (per ogni utente/salt devi ri-computare il dizionario → aumenta il costo)

Dizionario

in chiaro

codificate

gelato000	al47r33km
gelato0001	ldcfjoiu3
gelato0002	rlekekll39
...	...
gelato9999	39u4jrk0

Password file

```
pippo:8kfdjkjkf/2000:500:500:...
```

■ Shadow password file

- Il file `/etc/passwd` è leggibile a tutti perchè contiene informazioni che vanno al di là della password
- Ma questo rende(va) la vita facile agli attaccanti
- Il meccanismo del file shadow memorizza le password in un file separato, leggibile solo a root
 - `/etc/shadow`

■ Esempio di `/etc/passwd` con shadow password

```
tizio:x:501:502:Vittorio Tizio:/home/tizio:/bin/bash
```

```
caio:x:502:503:Aristide Caio:/home/caio:/bin/bash
```

```
sempronio:x:503:504:Franco Sempronio:/home/sempronio:/bin/bash
```

```
pinco:x:504:505:Pallino Pinco:/home/pinco:/bin/bash
```

■ Login spoofing

- L'attaccante scrive un programma (testuale o grafico) che presenta una finta schermata di login
- Attende che la vittima inserisca login/password
- Memorizza o spedisce la coppia login/password
- Visualizza un messaggio di “Login incorrect”
- Fa partire il vero programma di login – per esempio terminando la shell attuale
- La vittima crede di aver digitato male la password, questa volta entrando senza problemi nel sistema

■ Un piccolo esempio

```
#!/bin/csh -f
# Only for demonstration - (C) Bob Toxen
cat /etc/issue
echo -n "hostname login: "
set x="$<"
stty -echo
echo -n "Password: "
set y="$<"
echo $x ", " $y >> captured.dat
stty -echo
echo ""
echo Login incorrect
echo ""
exit
```


- **Un esempio di sequenza di login "normale":**

```
Welcome to XYZ Linux K.L (i586) - Kernel A.B.CD (tty1)
```

```
hostname login: franchi
```

```
Password:
```

```
Login incorrect
```

```
hostname login:
```

- **Un esempio di sequenza di login "faked":**

```
Welcome to XYZ Linux K.L (i586) - Kernel A.B.CD (tty1)
```

```
hostname login: franchi
```

```
Password:
```

```
Login incorrect
```

```
Welcome to XYZ Linux K.L (i586) - Kernel A.B.CD (tty1)
```

```
hostname login:
```

■ `/etc/issue`

- Deve essere più esplicito...
- Non deve contenere informazioni sul sistema (tipo di s.o., versione, etc.)!

■ **Esempio [(C) Bob Toxen]**

This is a private computer system. Unauthorized access is prohibited. Information here is proprietary and confidential. Your identifying information and actions are being logged. Violators will be prosecuted and defended against to the maximum extend of the law.

This message is displayed only at the start of the login sequence. If you see this message after attempting to login and receiving "Login incorrect" you may have been the victim of a "login simulator" which has stolen your password.

If this happens then notify your system administrator by the fastest possible means! Joe's pager number is 800-999-9999.

■ Windows

- Permette il login tramite la sequenza Ctrl-Alt-Canc
- La schermata iniziale di Windows XP con la lista degli utenti non dovrebbe essere utilizzata

■ Packet sniffing

- un packet sniffer è un software che analizza il traffico di rete
- cerca di individuare pacchetti contenenti coppie login/password spediti "in chiaro" da meccanismi di comunicazione come telnet e ftp
- memorizza le coppie login/password per uso futuro

■ Challenge and response

- Basato su funzioni one-way f
- La password (chiave) k è nota sia all'utente che al sistema a cui si vuole accedere (eventualmente codificata)
- Il sistema propone una challenge (sfida), ovvero un valore numerico c che viene spedito all'utente
- Sia il sistema che l'utente calcolano $f(c, k)$
- L'utente comunica al sistema il valore di $f(c, k)$
- Il sistema confronta questo valore con il valore calcolato localmente

■ Questo meccanismo rende inservibili i packet sniffer

■ Password – Note per l'amministratore di sicurezza

- Ogni utente è una possibile falla nella sicurezza del sistema
- Una volta entrato come utente, un attaccante ha varie possibilità di raggiungere i privilegi di superutente

■ Quali contromisure?

- Educare gli utenti sull'importanza di utilizzare password non deboli
- Eseguire periodicamente programmi di cracking per verificare la sicurezza delle password attuali
 - Nota: può richiedere autorizzazione "dall'alto"
- Implementare meccanismi per evitare password banali
- Utilizzare password one-shot: usa e getta

■ Password "one-shot"

- L'utente deve utilizzare una password diversa per ogni accesso.
- L'utente può avere un elenco di password (stampato) e scegliere elementi successivi della lista.

■ Problemi

- E' pericoloso: la lista può essere letta o copiata
- Viene utilizzato in ambiente militare, nel caso sia possibile proteggere fisicamente questa lista

■ Autenticazione tramite oggetti fisici

- Tessera bancomat
- Facili da copiare, basano la loro sicurezza su due meccanismi:
 - Codice PIN per evitare che si utilizzino carte copiate (e rubute)
 - Carta personale per evitare PIN troppo complessi

■ Smart card

- Difficili da copiare
- Dispongono di un'unità di calcolo, anche se semplice
- Permettono di utilizzare meccanismi di challenge e response

■ Autenticazione biometrica

- impronte digitali, retina, voce

■ **Tradizionalmente:**

- In UNIX, l'autenticazione avviene in diversi contesti (login, ftp, ...)
- Ogni servizio aveva la propria implementazione dell'autenticazione
- I meccanismi erano fissi e non configurabili

■ **Ai giorni nostri:**

- Librerie di autenticazioni, con moduli liberamente caricabili
- Meccanismo di autenticazione altamente configurabile

■ **Pluggable Authentication Module (PAM)**

- Un servizio generale di autenticazione basato su file di configurazione
- In quali UNIX?
 - Standard in Linux, FreeBSD, HP-UX, Solaris, MAC OS X
 - AIX utilizza LAM (L=loadable), simile ma proprietario di IBM

■ Componenti principali in PAM

- Servizi
 - Versioni PAM-aware dei tradizionali programmi di autenticazione UNIX
 - Eseguono chiamate ai moduli di autenticazione utilizzando l'API di PAM
 - Ad esempio: login, passwd, su, etc.
- Moduli che effettuano task di autenticazione
 - Autenticazione, gestione account, gestione password, logging
 - Ad esempio, un modulo per **/etc/shadow**
- File di configurazione generali
 - Associazioni meccanismi di autenticazione --- servizi
 - Contenuti in **/etc/pam.d** o in **/etc/pam.conf**
- File di configurazioni aggiuntionali
 - Relativi ai singoli moduli

%PAM-1.0

auth	sufficient	pam_rootok.so	
auth	required	pam_unix2.so	nullok
account	required	pam_unix2.so	
password	required	pam_pwcheck.so	nullok
password	required	pam_unix2.so	nullok use_first_pass
use_authok			
session	required	pam_homecheck.so	
session	required	pam_unix2.so	debug # none or trace

- **Il primo campo descrive la classe di operazione**
 - **auth**: procedure per l'autenticazione dell'utente
 - **account**: per modificare gli attributi di un account
 - **password**: per modificare la password
 - **session**: per debug e logging su syslog
- **I vari moduli di una classe formano uno stack per classe**
- **I moduli vengono valutati in ordine**
 - Ogni modulo risponde con **grant** o **deny**
 - Il secondo campo definisce come deve essere valutato il valore di ritorno

■ Il secondo campo dei file di configurazione:

- **sufficient**
 - se il modulo ritorna grant, la risposta è positiva e i moduli successivi non vengono considerati
- **requisite**
 - se il modulo ritorna deny, la risposta è negativa e i moduli successivi non vengono considerati
- **required**
 - questo modulo deve ritornare grant affinché l'operazione abbia successo
- **optional**
 - questo modulo influisce sul risultato solo se è l'unico modulo presente

```
##PAM-1.0
```

```
auth required pam_unix2.so
```

```
auth required pam_nologin.so
```

```
auth required pam_env.so
```

```
account required pam_unix2.so
```

```
account required pam_nologin.so
```

```
password required pam_pwcheck.so
```

```
password required pam_unix2.so
```

```
session required pam_unix2.so
```

```
session required pam_limits.so
```

Se il file /etc/nologin esiste,
impedisce accessi non-root e
stampa il contenuto del file

Setta alcune variabili di ambiente

```
use_first_pass use_authok
```

```
none # trace or debug
```

Legge i limiti di sistema contenuti
in /etc/security/limits.conf

```
#%PAM-1.0
```

```
auth requisite pam_unix2.so nullok
```

Nega l'accesso di root a meno che
il terminale corrente non sia listato
in /etc/securetty



```
auth required pam_securetty.so
```

```
auth required pam_nologin.so
```

```
auth required pam_homecheck.so
```

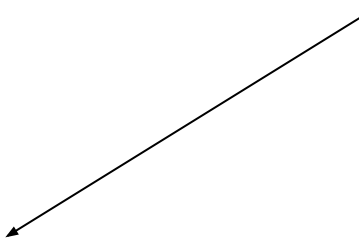
Verifica se la home esiste



```
auth required pam_env.so
```

```
auth required pam_mail.so
```

Effettua dei controlli di
validità sulla nuova
password



```
account required pam_unix2.so
```

```
password required pam_pwcheck.so nullok
```

```
password required pam_unix2.so nullok use_first_pass use_authtok
```

```
session required pam_unix2.so none # debug or trace
```



Controllo degli Accessi

- Dall'autenticazione all'autorizzazione
 - Una volta che i soggetti sono stati autenticati, il problema successivo da affrontare è l'**authorization** (o **access control**)

- L'**access control** è un elemento centrale della sicurezza informatica i cui obiettivi sono:
 - **Confidentiality** (riservatezza) — impedire a utenti non autorizzati di accedere alle risorse
 - **Integrity** (integrità) — impedire a utenti non autorizzati di modificare le risorse
 - **Availability** (disponibilità) — permettere agli utenti autorizzati di accedere alle risorse in modo tempestivo

- L'**access control** è un elemento centrale della sicurezza informatica i cui obiettivi sono:
 - **Confidentiality** (riservatezza) — impedire a utenti non autorizzati di accedere alle risorse
 - **Integrity** (integrità) — impedire a utenti non autorizzati di modificare le risorse
 - **Availability** (disponibilità) — permettere agli utenti autorizzati di accedere alle risorse in modo tempestivo
- **Trade-off** tra questi obiettivi
 - Ex: banale garantire **confidentiality** e **integrity** se si sacrifica la **availability** (negando qualsiasi accesso)

- L'**access control** si realizza attraverso un insieme di **policy** (politiche) e di meccanismi per applicarle, riguardo le operazioni effettuate dai **subject** (soggetti) sugli **object** (oggetti)
- Una **politica** di **access control** stabilisce quali accessi sono permessi, in quali circostanze e da chi

- **Discretionary Access Control (DAC):**
 - l'accesso si basa sull'identità dei **subject** e su regole che stabiliscono cosa i **subject** sono autorizzati a fare su determinati **object**
 - È detto *discrezionale* perché un **subject** può decidere di concedere (o negare) l'accesso ad altri **subject**
- **Mandatory Access Control (MAC):**
 - l'accesso si basa sul confronto di **security label** (etichette di sicurezza, che indicano quanto un **object** sia sensibile o critico) con i **security clearance** (livelli di autorizzazione) dei **subject**.
 - È detto *obbligatorio (mandatory)* perché le etichette e le autorizzazioni sono definite a livello di sistema e non possono essere modificate dai **subject**
- **Role-Based Access Control (RBAC):**
 - l'accesso si basa sui **ruoli (role)** che i **subject** ricoprono all'interno del sistema e su regole che definiscono quali accessi sono permessi ai **subject** in base ai ruoli che gli sono stati assegnati

■ Principio di *Accesso Mediato* (Reference monitor)

- Tutti gli accessi ad un oggetto devono essere controllati
- In pratica:
 - molti OS controllano i diritti solo al momento dell'apertura
 - non vengono controllate le operazioni successive

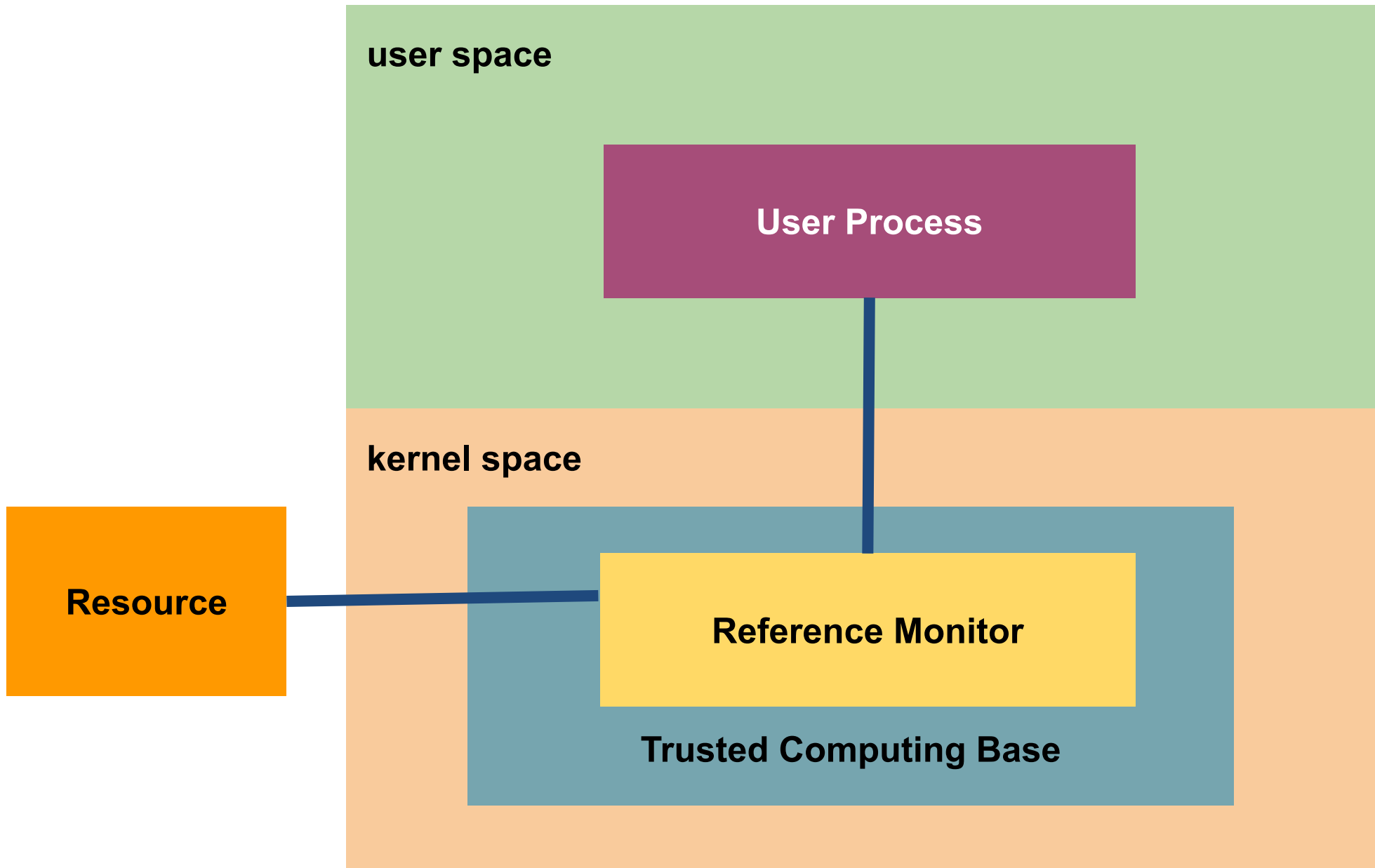
■ Principio di *Separazione dei privilegi*

- Un sistema non dovrebbe concedere permessi in base ad una singola condizione
- Esempio: UNIX permette ad un utente di diventare root se
 - conosce la password di root
 - fa parte del gruppo *wheel*

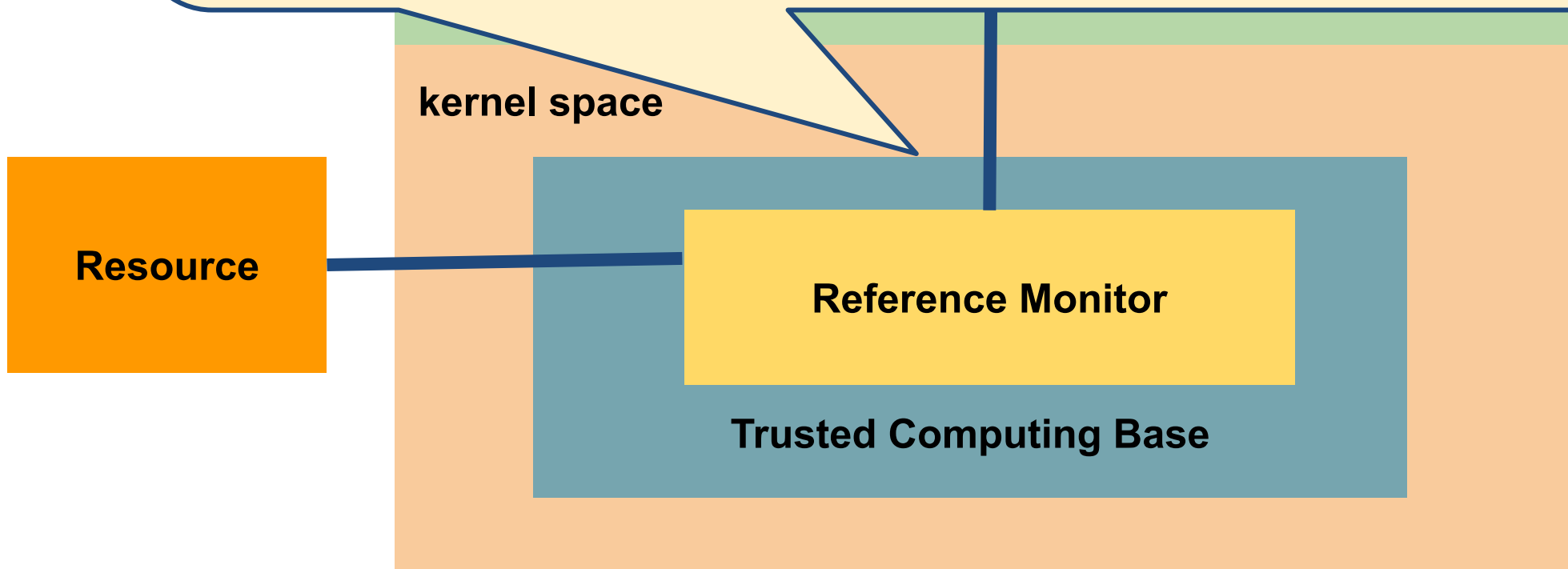
- **Principio di *Failsafe default***
 - nessun soggetto ha diritti per default
- **Principio di *Privilegio minimo* (Need-to-know)**
 - ogni soggetto ha, in ogni istante, i soli diritti necessari per quella fase dell'elaborazione
 - In pratica:
 - molti sistemi non hanno la granularità di privilegi e permessi richiesti per implementare questo principio
 - esempio: root/administrator vs utenti

Trusted Computing Base e Reference Monitor

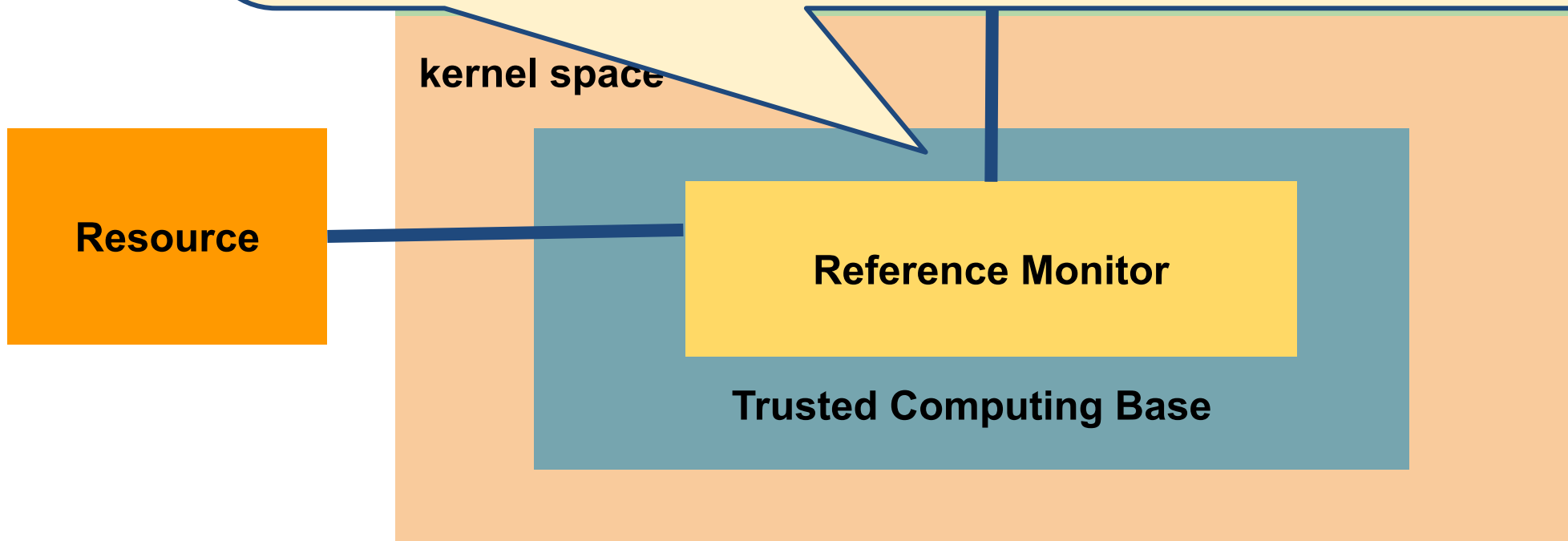
Controllo accessi



- Trusted Computing Base (TCB): l'insieme dei componenti hardware, firmware e software che sono critici per la sua sicurezza
- Vulnerabilità presenti all'interno della TCB potrebbero compromettere le proprietà di sicurezza dell'intero sistema



- Tutte le chiamate di sistema passano attraverso il **reference monitor** per il controllo della sicurezza
- il Reference monitor usa le regole di sicurezza implementate per capire chi ha accesso (ex: usando matrice di accesso, vedi prossime slide)



- **Insieme dei *soggetti (subjects)* S**
 - I soggetti sono le entità attive che eseguono azioni
- **Insieme degli *oggetti (objects)* O**
 - Gli oggetti sono le entità (passive/attive) su cui vengono eseguite azioni
 - Nota: $S \subseteq O$
- **Insieme dei *diritti di accesso (access rights)* R**
 - L'insieme delle azioni che possono essere eseguite
- **In un sistema operativo UNIX**
 - Soggetti: processi, thread
 - Oggetti: file, dispositivi, processi
 - Diritti di accesso: read, write, execute

■ Dominio di protezione

- Un insieme di coppie $\langle o, rs \rangle$, dove $o \in O$ e $rs \subseteq R$
- Informalmente, ogni coppia specifica un oggetto e un insieme di azioni che possono essere eseguite su tale oggetto

■ I subject (utenti) sono associati a un determinato protection domain in cui operano

- il dominio di protezione determina cosa l'utente può fare e cosa non può fare
- Ex; UNIX: dominio determinato da *user-id* e *group-id*

- **L'associazione soggetto / dominio può essere *statica* o *dinamica***
 - L'associazione può variare durante la vita di un soggetto
 - Può essere rappresentato con un diritto speciale nella matrice
- **Esempi di associazione dinamica**
 - Modalità user / kernel nell'esecuzione dei processi
 - Normalmente i processi operano in *user mode*
 - Quando eseguono una *system call*, passano in *kernel mode* e acquisiscono i privilegi necessari per eseguire la chiamata di sistema
 - Bit **SETUID** di Unix
 - Modalità “Run as” di Windows

- Un modello per il *Discretionary Access Control (DAC)*
- **Matrice di accesso**
 - una matrice domini/oggetti
 - l'elemento $A(i,j)$ contiene i diritti che il dominio D_i prevede per l'oggetto O_j
- **Quando viene creato un nuovo oggetto**
 - si aggiunge una colonna alla matrice di accesso
 - il contenuto di tale colonna è deciso al momento di creazione dell'oggetto

oggetti domini	File1	File2	File3	Stampante
D1	read			
D2		read write	read	
D3			read write	write
D4				write

- L'utente A nel dominio D2 modifica File2, l'utente B in D3 modifica File3
- Gli utenti A e B attivano la funzione di “correttore ortografico” basata su File4, che funge da dizionario
- Il dizionario è proprietario e **non** deve essere copiato

		oggetti			
		File1	File2	File3	File4
A	D1	read			
	D2		read write	read	read
B	D3			read write	read

- Ma ora A e B possono fare copie di File4

- Quando un processo passa da un dominio all'altro, si esegue un'operazione (**switch**) su un oggetto (il dominio)
- Il passaggio da un dominio all'altro si può controllare inserendo i domini tra gli oggetti della matrice d'accesso
 - Il passaggio è permesso se nella matrice è specificata la possibilità di **switch**

		oggetti				
		domini	File1	File2	File3	File4
A	D1	read				
	D2		read write	read		switch
	D3			read write		switch
	D4				read	

- Introduciamo un nuovo *dominio* D4, in modo che il dizionario possa essere letto solo all'interno di quel dominio
- Lo switch non solo cambia dominio, ma copia anche i diritti di accesso dal dominio di origine a quello di destinazione
- Poiché più utenti possono effettuare lo switch verso lo stesso dominio, essi vengono mantenuti logicamente distinti creando istanze multiple dello stesso domain
- Questo meccanismo implementa efficacemente il ***principle of least privilege***

	domini	oggetti				
		File1	File2	File3	File4	D4
A	D1	read				
	D2		read write	read		switch
B	D3			read write		switch
	D4				read	

<div>oggetti</div> <div>domini</div>		File1	File2	File3	File4	D4
A	D1	read				
	D2		read write	read		switch
	D3			read write		switch
	D4		read write	read	read	
	D4			read write	read	

- **Come tabella globale:**

- La matrice viene memorizzata come un array bidimensionale (tabella) con celle contenenti:

`<insieme_di_diritti_di_accesso>`

- **Vantaggi:**

- Implementazione semplice

- **Svantaggi:**

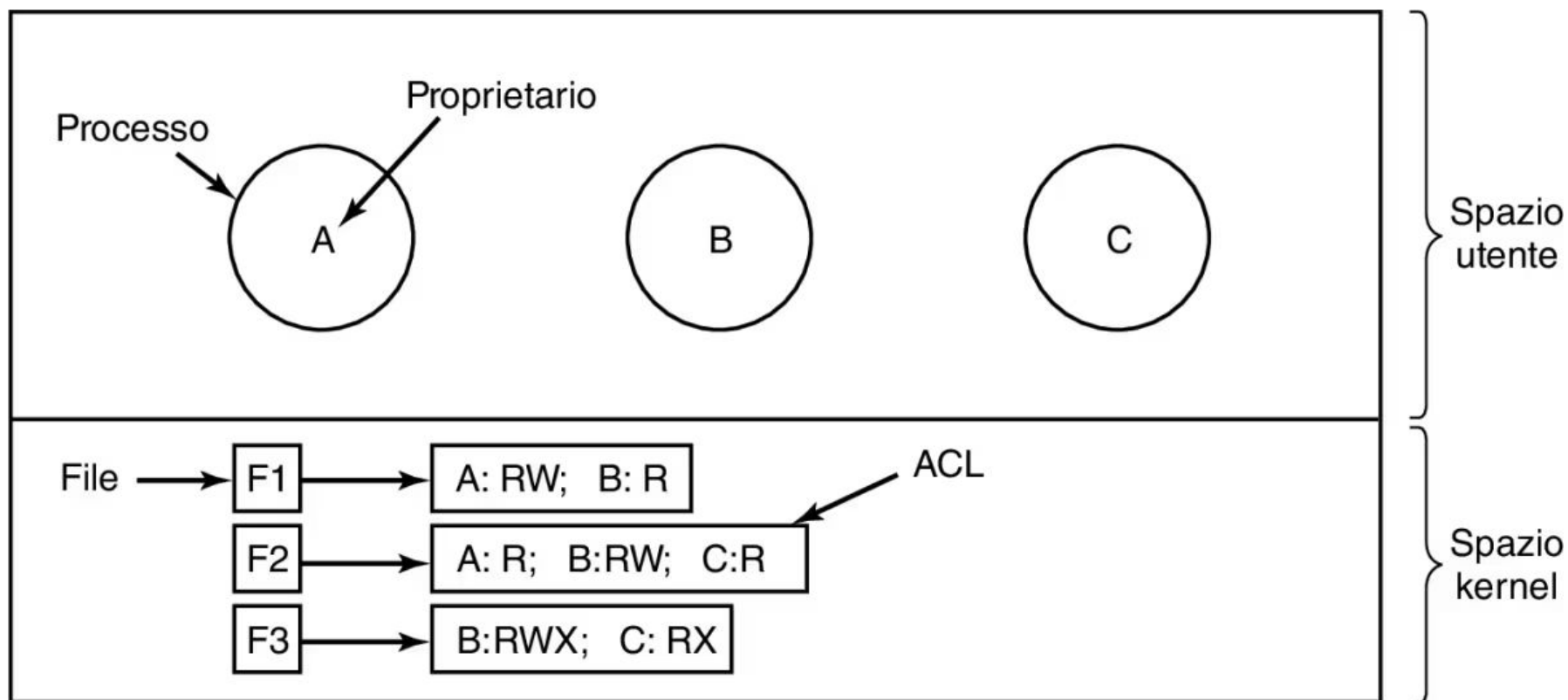
- La tabella può diventare enorme
- Difficile da gestire in un sistema dinamico, in cui domain e oggetti vengono aggiunti/rimossi e i diritti di accesso cambiano nel tempo

■ Access control list (ACL)

- la tabella viene memorizzata per colonna
- Ad ogni oggetto viene associata una lista di elementi *<dominio, diritti di accesso>*

■ Ottimizzazioni

- includi solo domini diversi da default (**no access**)
- raggruppa domini in insiemi (**gruppi**) e dai diritti per questi insiemi



- **Come Capability:**
 - NON le vediamo in questo corso ...

■ Come Capability:

- Ad ogni dominio viene associata una lista di *capability*, ovvero coppie
<oggetti, diritti di accesso>

■ Come avviene il controllo di accesso:

- I processi mantengono le capability e le presentano quali “credenziali” per accedere all’oggetto
- Sono una sorta di “chiave” per l’accesso alla “serratura” che protegge l’oggetto

■ Chi mantiene le capability?

- I processi che le “presentano” per esercitare i diritti di accesso sugli oggetti
- Le capability funzionano come chiavi che aprono le serrature che proteggono gli oggetti, oppure come inviti che convincono i “buttafuori” a far entrare qualcuno a una festa

- **Perché il meccanismo delle capability funzioni:**
 - le capability non possano essere “coniate”
- **Due possibili approcci:**
- **Capability mantenute nello spazio kernel associato al processo**
 - Protette dai meccanismi di protezione del kernel
 - Esempio: Hydra, Linux (parzialmente...)
- **Capability mantenute nello spazio utente**
 - Protette da meccanismi crittografici
 - Capability memorizzate dai processi, ma non modificabili
 - Approccio utilizzato nei sistemi distribuiti

■ La revoca può essere:

- immediata o ritardata (subito o si può attendere)
- selettiva o generale (per alcuni i domini o per tutti)
- parziale o totale (tutti i diritti o solo alcuni)
- temporanea o permanente

■ Revoca in sistemi basati su ACL

- E' sufficiente aggiornare in modo corrispondente le strutture dati dei diritti di accesso.

■ Revoca in sistemi basati su capability

- L'informazione relativa ai permessi è memorizzata presso i processi. Come si può allora revocare i diritti di accesso?

- **Capability a validità temporale limitata:**
 - Una capability “scade” dopo un prefissato periodo di tempo
 - Revoca ritardata
- **Doppia memorizzazione**
 - Ogni capability viene controllata prima di essere utilizzata
 - Si perdono alcuni dei benefici delle capability
- **Capability indirette.**
 - Vengono concessi diritti non agli oggetti ma a elementi di una tabella globale che puntano agli oggetti.
 - E' possibile revocare diritti cancellando elementi della tabella intermedia
- **Cambiamento dell'identità dell'oggetto (contatore nel nome):**
 - I processi devono chiedere nuovamente l'autorizzazione di accesso
 - Può essere pesante se ci sono frequenti variazioni



Controllo degli Accessi in Unix e Linux

■ Esempio: Unix

```
ls -l /etc/passwd
```

```
-rw-r--r-- 1 root root 3133 mar 25 09:31 /etc/passwd
```

- UNIX ha solo 3 domini: owning user, owning group, others
- I permessi Unix tradizionali sono una forma molto semplificata e rigida di ACL
 - con solo 3 possibili "soggetti" (owner, group, others) invece di una lista arbitraria di utenti e gruppi
 - Si parla di **DAC**: Discretionary Access Control

- Ogni **file** ha:
 - *un owner* — l'utente che ha creato il file
 - *un group* — un insieme di utenti
- Ogni file ha 9 bit di **diritti di accesso**, corrispondenti a:
 - **r**ead, **w**rite, **e**xecute per l'owner
 - **r**ead, **w**rite, **e**xecute per il group
 - **r**ead, **w**rite, **e**xecute per gli altri (other)
- Esempi:
 - `rw-r--r--` (644)
 - `rwxr-xr-x` (755)

- **Utenti e gruppi sono identificati tramite interi** presenti nel file delle password:
 - **user-id**
 - **group-id**
- **Esempi di righe nel file `/etc/passwd`:**

```
ferretti:x:501:1000:Stefano Ferretti:/home/ferretti:  
ghini:x:502:1000:Vittorio Ghini:/home/ghini:
```

- **Ogni processo possiede sei o più ID associate ad esso:**
 - *real user ID, real group ID*
 - identificano il vero utente e gruppo che esegue il processo
 - questi valori sono presi dalla entry nel passwd file
 - non cambiano durante la vita di un processo
 - *effective user ID, effective group ID, supplementary group IDs*
 - Impostati dinamicamente durante l'esecuzione del processo tramite il meccanismo **setuid**
 - sono quelle effettivamente utilizzate per determinare i nostri diritti di accesso al file system

■ Normalmente:

- *effective user id = real user id*
- *effective group id = real group id*
- ovvero, i diritti di accesso sono determinati in base a chi esegue il programma

■ In alcuni casi, un comportamento diverso è desiderabile:

- potrebbe essere necessario che un processo abbia un insieme di diritti maggiori di chi esegue il programma
- Esempio: comando **passwd**
 - Eseguito da chiunque
 - Deve cambiare il file `/etc/passwd`

Due meccanismi speciali di UNIX/Linux che modificano temporaneamente l'identità di chi esegue un programma

- ***set-user-ID (SUID) e set-group-ID (SGID):***

- nei bit dei permessi di un *file eseguibile*, il bit *set-user-ID* causa la seguente operazione

effective user id := owner del file

i.e. durante l'esecuzione, l'effective user ID diventa quello del proprietario del file (non dell'utente che lo ha lanciato)

- nei bit dei permessi di un *file eseguibile*, il bit *set-group-ID* causa la seguente operazione

effective group id := owning group del file

- nei bit dei permessi di una *directory*, il bit *set-group-ID* causa la seguente operazione
 - i nuovi file creati nella directory ereditano il group owner

- **E' possibile utilizzare questi due bit per risolvere il problema di passwd:**
 - l'owner del comando **passwd** è **root**
 - quando **passwd** viene eseguito, il suo *effective user id* è uguale a **root**
 - il comando può scrivere su **/etc/passwd**

- **Un utente normale vuole cambiare la propria password usando il comando passwd**
- **Problema: Il file `/etc/shadow` (che contiene le password cifrate) è leggibile solo da root**

```
$ ls -l /etc/shadow -rw-r----- 1 root shadow 1234 Sep  
30 10:00 /etc/shadow
```

■ Meccanismo SUID

- Quando un file eseguibile ha il bit **SUID impostato**, viene eseguito con i **privilegi del proprietario del file**, non dell'utente che lo lancia

```
$ ls -l /usr/bin/passwd
```

```
-rwsr-xr-x 1 root root 63960 /usr/bin/passwd
```

^

|

Bit SUID (s invece di x)

- **Impostare il Bit SUID**

```
chmod u+s file
```

```
chmod 4755 file # 4 = SUID, 755 = permessi base
```

- **Rimozione il Bit SUID**

```
chmod u-s file
```

```
chmod 0755 file
```

■ Alcuni dettagli:

- Per aprire un file per nome:
 - Necessario il diritto di esecuzione (search) su tutte le directory che fanno parte del path
 - Attenzione ai nomi relativi: current directory è implicita
- Per creare un nuovo file in una directory:
 - Necessari i diritti di esecuzione e scrittura sulla directory
- Per cancellare un file in una directory:
 - Necessari i diritti di esecuzione e scrittura sulla directory
 - Non abbiamo bisogno di diritti sul file

- **Il 12° bit dei permessi è detto sticky bit**
 - Il nome "corretto" è saved text mode
 - da cui la t nei permessi mostrati da ls
- **Nelle directory** (oggi usato solo in questo caso)
 - Impedisce ad un utente di cancellare file che non gli appartengono, nonostante abbia i diritti di scrittura sulla directory
 - Esempio di utilizzazione: /tmp
 - Tutti gli utenti possono creare file in /tmp (permessi 777)
 - Ma grazie al sticky bit, l'utente A non può cancellare i file dell'utente B
 - Ogni utente può gestire solo i propri file temporanei

■ Modalità di accesso ad un file:

- *effective uid, effective gid, supplementary gids* sono proprietà del processo in esecuzione
- *owner* e *group owner* sono proprietà del file a cui si vuole accedere

■ Algoritmo

- IF *effective uid* == 0 (root) THEN **access allowed**
- ELSE IF *effective uid* == *owner* AND appropriate user permission THEN **access allowed**
- ELSE IF *group owner* \in *supplementary gids* \cup {*effective gid*} AND appropriate group permission THEN **access allowed**
- ELSE IF appropriate other permission THEN **access allowed**
- ELSE **access denied**

■ **umask: maschera per la creazione dei file**

- Meccanismo di sicurezza, utilizzato per **rimuovere di default** dei permessi, tutte le volte che un processo crea un nuovo file
- Tutti i bit che sono accesi nella maschera, verranno spenti nell'access mode del file creato
 - In pratica: Permessi finali = Permessi massimi - umask

creazione di un file con diritti `rw- rw- rw-`

Mode	R	W		R	W		R	W	
Mask									
Final	R	W		R	W		R	W	

`umask 000`

Mode	R	W		R	W		R	W	
Mask					W		R	W	
Final	R	W		R					

`umask 046`

- testa comando **umask**
- vedi script **umask_demo_script.sh**

■ Alcune considerazioni:

- Il modello tradizionale di AC in UNIX è molto semplice
- La motivazione (originale) era l'efficienza: 12 bit per file per memorizzare la lista di controllo di accesso
- Spesso (ma non sempre) è sufficiente per implementare gli scenari di accesso che si presentano in sistemi UNIX

■ Esempio

- In ambiente universitario:
 - utenti lavorano su insieme di file separati
 - gruppi di lavoro su particolari progetti (studenti, personale)

- **In alcuni casi, sono necessari dei workaround per superare questo semplice modello**
 - organizzazione dei gruppi in modo non ovvia, che non riflette l'organizzazione strutturale
 - il meccanismo di set-user-id può portare facilmente ad un sistema compromesso
 - violazione del principio del privilegio minimo
 - alcune applicazioni (e.g. FTP, Samba) implementano le proprie estensioni al meccanismo di accesso
- **Prezzo: incremento della complessità del sistema**

- **Il gruppo POSIX ha tentato di standardizzare numerose problematiche relative alla sicurezza:**
 - Access Control Lists (ACL) * (in Linux)
 - Capability * (in Linux)
 - Mandatory Access Control (MAC)
 - Information Labeling
 - Audit
- **Sfortunatamente**
 - Standardizzare aree così diverse e variegate era un progetto troppo ambizioso
 - Tuttavia, il lavoro svolto dal Working Group non è andato perso

■ **ACL Posix in Linux**

- Disponibili come patch per kernel Linux 2.4
- Di serie nelle attuali distribuzioni Suse, UnitedLinux
- Aggiunte definitivamente nel kernel 2.5-2.6
- Quali file system?
 - Ext2, Ext3, IBM JFS, ReiserFS, SGI XFS

■ **ACL Posix in FreeBSD**

- Nella standard release a partire da 01/2003
- Sviluppate dal progetto TrustedBSD

■ In un file system

- I permessi di ogni oggetto vengono rappresentati come una **ACL**
- Una ACL consiste in un insieme di **ACL entry**

■ Da UNIX tradizionale ad ACL:

- Come sappiamo, esistono tre "user class"
 - owner, group, others
- Ognuna di queste classi è rappresentata da una AC entry
- I permessi per ulteriori utenti e gruppi occupano ACL entry aggiuntionali

■ Tipi di ACL entry

- **Owner** user::rwx
- **Named user** user:name:rwx
- **Owning group** group::rwx
- **Named group** group:name:rwx
- **Mask** mask::rwx
- **Others** other::rwx

■ **ACL minime**

- Contengono solo tre entry
- Sono equivalenti ai bit di permesso di UNIX tradizionale

■ **ACL estese**

- Hanno più di tre ACL entry
- Contengono una mask entry
- Possono contenere qualunque numero di named user e di named group entries

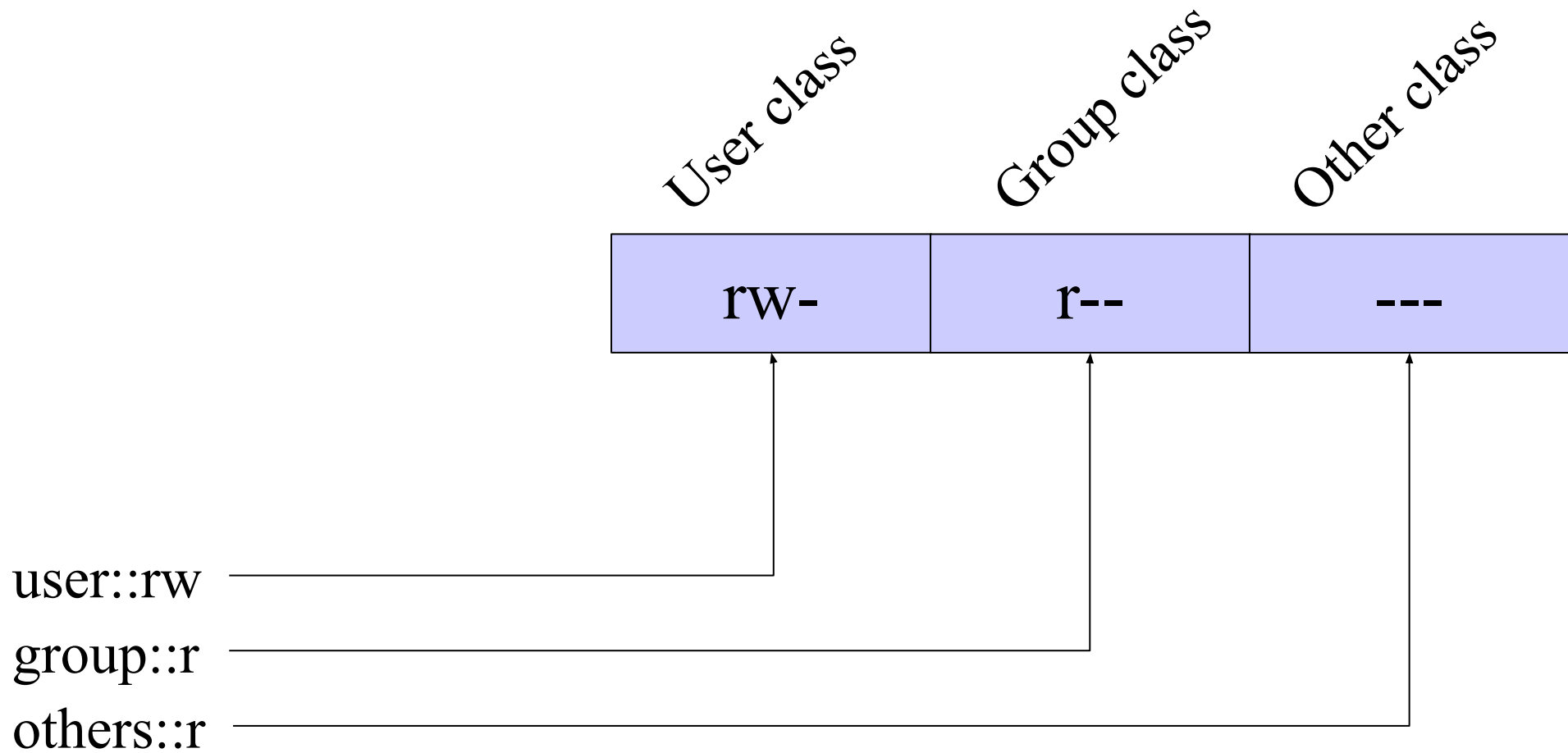
■ Come funzionano le ACL minime

- Come al solito!

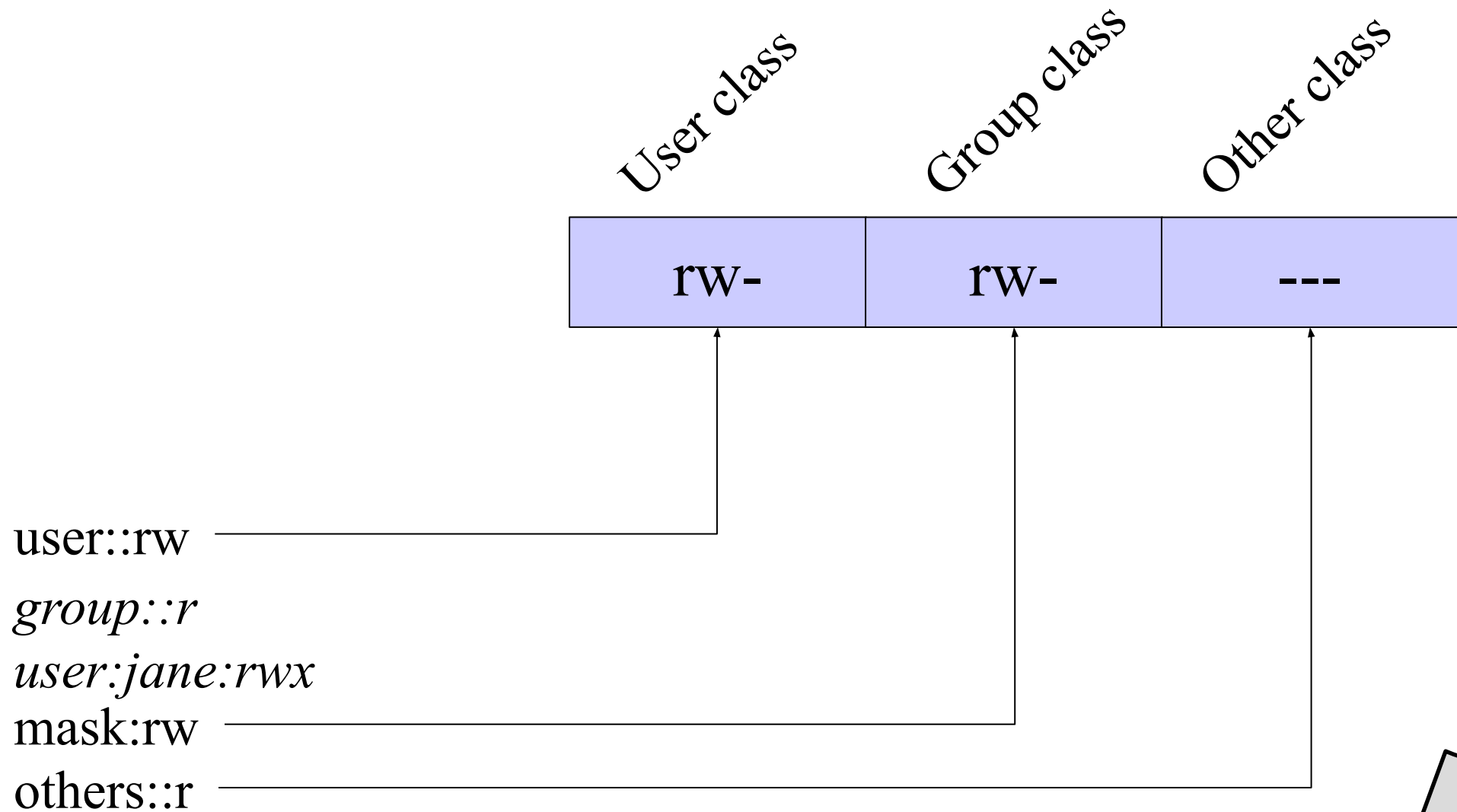
■ Come funzionano le ACL estese

- Tutti i named user e i named group entrano a far parte della classe group
- Bisogna distinguere fra:
 - permessi della classe gruppo
 - ACL entry della classe group
- I permessi della classe gruppo
 - sono contenuti nella maschera
 - rappresentano un "upper bound" ai permessi che possono essere ottenuti dalle entry della classe gruppo

- **ACL minime**



■ ACL estese



- **ACL estese**

*L'utente jane ha come
diritto di accesso rw-*

User class

Group class

Other class

rw-	rw-	---
-----	-----	-----

user::rw

group::r

→ user:jane:rw

→ mask:rw

others::r

- **ACL estese**

*L'owning group ha come
diritto di accesso r--*

User class

Group class

Other class

rw-	rw-	---
-----	-----	-----

user::rw

group::r

user:jane:rwX

mask:rw

others::r

■ Access ACL

- Ogni file (regolare e non) è associato ad una access ACL
- Le access ACL determinano i diritti di accesso

■ Default ACL

- Le directory sono associate anche ad una default ACL
- Le default ACL di una directory determinano i diritti di accesso dei file creati in quella directory

■ Come vengono ereditate le default ACL?

- la default ACL viene copiata nella access ACL dei nuovi file
- la default ACL viene copiata nella default ACL delle nuove directory

■ Ulteriori dettagli

- quando un file viene creato
 - possono essere specificati dei diritti di accesso
 - viene effettuata l'intersezione tra questi e la default ACL
- se una directory non ha default ACL, si utilizza il meccanismo tradizionale UNIX (umask, etc.)

- **Algoritmo di access control per un processo p che accede un file f con tipo di accesso t**
- **Primo passo:**
 - si seleziona la ACL entry di f che meglio identifica p
 - vengono cercati, nell'ordine:
owner, named users, (owning or named) groups, others
 - i gruppi vengono analizzati uno a uno, fino a quando non se ne trova uno che contiene t
- **Secondo passo:**
 - si verifica se la ACL entry selezionata contiene t

■ Come sono realizzate:

- Informazioni di lunghezza variabile associate agli oggetti dei file system
- Le ACL non sono le uniche informazioni "addizionali" che possono essere associate ad un file

■ Extended attribute (EA)

- Coppie *name=value* associate ad ogni file
- ACL sono implementate come EA

■ Nota:

- esistono system call specifiche per ACL (POSIX standard)
- EA non sono uno standard POSIX

■ EXT2/EXT3/EXT4

- EA memorizzate in 1 blocco separato, puntato dall'inode
- possibilità di eseguire sharing del blocco (per risparmiare memoria)
- limite di 1KB

■ JFS

- EA memorizzate in un extent di blocchi, puntato dall'inode
- se piccole: nell'inode stesso
- nessun limite alla dimensione

■ XFS

- EA piccoli memorizzati nell'inode
- EA più grandi memorizzati con una struttura B-tree
- molto efficiente, ma anche molto complicato

■ ReiserFS

- ReiserFS ha un supporto efficiente per piccoli file
- Gli EA vengono memorizzati in piccoli file, il cui nome è associato all'inode, in una directory di sistema nascosta

■ Dimensioni massime

- XFS, Ext2, Ext3: max 25-32 entry
- JFS, ReiserFS: max 8191 entry

■ Performance

- ACL molto grandi riducono le performance
- ma... ACL molto grandi non hanno senso, è meglio utilizzare al meglio i gruppi

■ **Compatibilità**

- File system diversi hanno supporti per ACL diversi
- In alcuni casi, si possono creare dei problemi
 - cosa succede se un ACL molto grande viene copiata in un file system con ACL limitate?

■ **Integrazione Unix-Windows**

- NTFS ACL e POSIX ACL non sono totalmente compatibili
 - ad es: possibilità di accesso differenti
- Tuttavia, le versioni correnti di SAMBA permettono una buona compatibilità

- **Tool standard (cp, ls, mv, etc.)**
 - tutti supportano le POSIX ACL
 - Molti tool aggiuntivi quali **tar** e **rsync** supportano ACM
 - in alcuni casi sono necessarie specifiche opzioni
- **Ma molte applicazioni ancora non le supportano ck**
 - Ad esempio, i file manager non le supportano
 - Il supporto non è completo, per es. **cpio** e **scp** non le supportano

■ Problema

- Un processo eseguito come root può accedere senza restrizioni a qualunque risorsa del sistema
 - apertura di porte riservate,
 - montare/smontare file system
 - accesso a tutti i possibili file, etc.

■ Osservazione

- Molti programmi eseguono come root solo per ottenere solo alcuni di questi privilegi

■ Posix capability:

- "Spezza" il ruolo del superutente in un insieme di sottoprivilegi

■ Alcune delle capability definite in Linux

- CAP_CHOWN
Allow for the changing of file ownership
- CAP_DAC_OVERRIDE
Override all DAC access restrictions
- CAP_DAC_READ_SEARCH
Override all DAC restrictions regarding read and search
- CAP_KILL
Allow to send signals to processes belonging to others
- CAP_SETUID
Allow changing of the UID
- CAP_NET_BIND_SERVICE
Allow binding to ports below 1024

- **Ogni processo ha tre bitmap che rappresentano capability**
 - *Effective Set*
 - Contiene le capability che il processo possiede ad un certo istante
 - E' contenuto nel Permitted Set
 - Permette Capability Bracketing
 - *Permitted Set*
 - Contiene il massimo insieme di capability che un processo possiede
 - *Inheritable Set*
 - Contiene il sottoinsieme di capability che un processo può lasciare in eredità ai suoi sottoprocessi

■ Considerazioni

- L'uso di capability sta prendendo pian piano piede all'interno dei programmi che normalmente usano privilegi di root
- Esempio: sendmail e bind utilizzano capability
- Hanno ricevuto un trattamento "riservato" nel kernel 2.6, in occasione dell'introduzione del concetto di Linux Security Module

■ **Discretionary Access Control (DAC)**

- I controlli di accesso sono basati sull'identità dei soggetti e sui permessi di accesso assegnati agli oggetti
- Il proprietario di un oggetto ha i diritti OWNER/COPY su di esso
- Da qui il nome “discrezionale”

■ **Modelli di accesso non discrezionali (Mandatory, MAC)**

- Il controllo degli accessi è basato su regole e informazioni associate ai soggetti ed agli oggetti
- L'informazione è chiamata livello di sicurezza / sensitività
- I livelli vengono implementati come etichette
- Il proprietario non ha privilegi speciali. Le etichette vengono associate esternamente e non determinate dal proprietario

■ **DAC è sufficiente quando:**

- il sistema non ha dati che debbano essere protetti da abusi del proprietario
- quando tutti i dati possono essere condivisi a discrezione del proprietario dei dati

■ **DAC è insufficiente quando:**

- Quando il sistema ha requisiti di protezione che eccedono quelli del proprietario
- Esempio:
 - Alice possiede il file A e autorizza Bob alla lettura del file A
 - Bob legge il file A e lo scrive sul file B
 - Bob può autorizzare Carl a leggere il file B

- **Livelli di sicurezza**

- In BLP, i livelli scelti riflettono la classificazione militare delle informazioni
- Top secret (TS), Secret (S), Confidential (C), Unclassified (U)
- Ordine lineare: $TS < S < C < U$

- **Il livello di sicurezza $L(S)$ di un soggetto S riflette le autorizzazioni che il soggetto ha sulle informazioni**

- livello massimo: security clearance
- livello corrente: può essere più basso

- **Il livello di sicurezza $L(O)$ di un oggetto O riflette i suoi requisiti di protezione**

- ogni oggetto ha un unico livello di sicurezza

- **“Information flows up, not down” (confidentiality)**
 - “Reads up” vietato, “reads down” permesso
 - “Writes up” permesso, “writes down” vietato
- **Simple Security Property (Versione 1)**
“No reads up”
 - Un soggetto S può leggere un oggetto O se e solo se:
 $L(O) \leq L(S)$ e S ha il permesso di leggere O
- ***-Property (Versione 1)**
“No writes down”
 - Un soggetto S può scrivere un oggetto O se e solo se:
 $L(S) \leq L(O)$ e S ha il permesso di scrivere O