




Mecanismos Básicos de Segurança de Software


Parte I: Enquadramento e Protecção

Segurança de Software
2019
Nuno Santos



Na aula passada

- ▶ O software tem **deficiências**
- ▶ Tornam-se vulnerabilidades quando um atacante explora essas **vulnerabilidades** para fins maliciosos
- ▶ Ataques cada vez mais frequentes e sofisticados, sendo efectuados por meio de **malware**



▶ 2 SS - Nuno Santos 2019



Nesta aula

- ▶ De que forma os sistemas actuais tentam evitar a existências de vulnerabilidades nos programas?
- ▶ De que forma é que os sistemas impedem danos causados por programas comprometidos ou maliciosos?



▶ 3

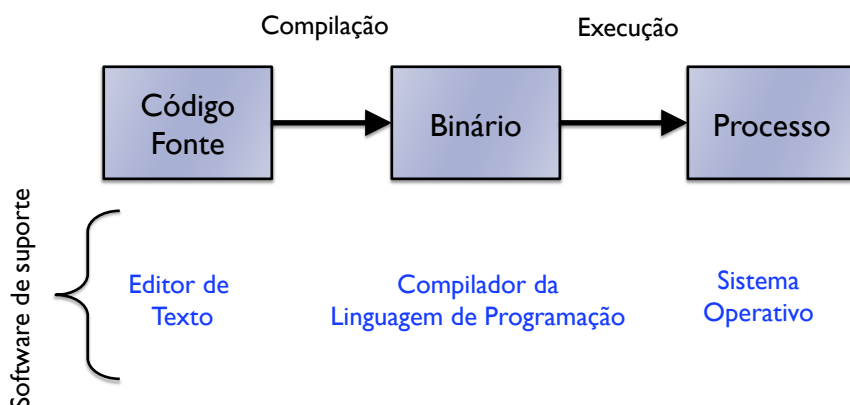
SS - Nuno Santos

2019



Ciclo de vida de uma aplicação

- ▶ Uma aplicação (um programa) passa por várias fases



▶ 4

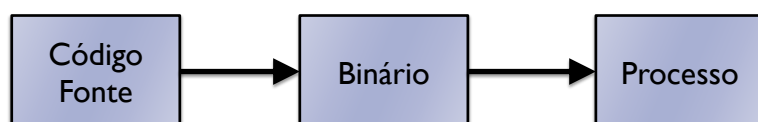
SS - Nuno Santos

2019



Mecanismos de segurança básicos

- ▶ Para evitar vulnerabilidades e prevenir violações de segurança por software malicioso ou incorrecto



Implementados nestes dois componentes:



Compilador da
Linguagem de Programação

Sistema
Operativo



Plano para esta aula

- ▶ Mecanismos de segurança nos sistemas operativos
- ▶ Mecanismos de segurança nas linguagens de programação

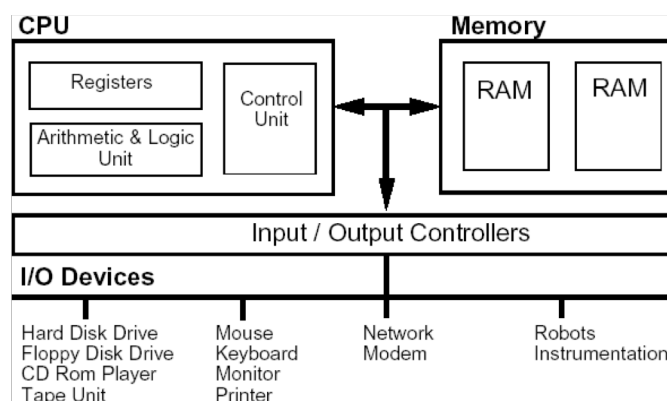
Mecanismos de segurança nos sistemas operativos

7

SS - Nuno Santos

2019

Arquitectura de um computador



- ▶ **Fluxo de execução de um programa:** programa é carregado para memória e depois o CPU executa as suas instruções: lê os dados, processa-os, e escreve resultados

▶ 8

SS - Nuno Santos

2019



Exemplo de um programa em C

► O que faz o programa?

- Soma dois números

► Instruções guardadas em memória

► Variáveis guardadas em memória

```
int main() {  
    int arg1 = 0;  
    int arg2 = 0;  
    int res = 0;  
  
    while(1) {  
        printf("Escreve parcela 1:\n");  
        scanf("%d", &arg1);  
  
        printf("Escreve parcela 2:\n");  
        scanf("%d", &arg2);  
  
        result = arg1 + arg2;  
        printf("Resultado: %d\n\n", res);  
    }  
    return 0;  
}
```

► 9

SS - Nuno Santos

2019

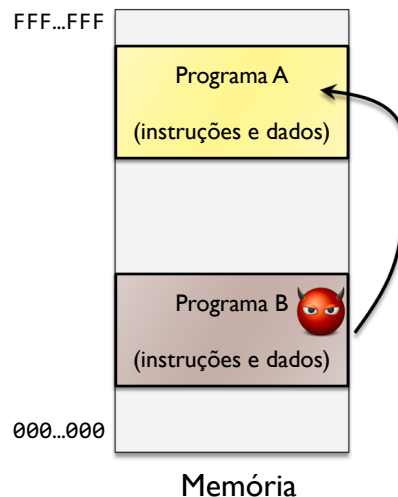


Desafio

► Dois programas executam-se no computador:

► Como garantir que um programa malicioso não acede à memória de outro?

- Mesmo se aplica a outros objectos, p. ex. ficheiros
- E entre diferentes utilizadores?



► 10

SS - Nuno Santos

2019



Protecções feitas pelo sistema operativo

- ▶ **Sistema operativo:** software privilegiado que gere e protege os recursos do sistema (ex. Windows, Linux)
- ▶ Implementa protecções entre:
 - ▶ Utilizadores legítimos e entre utilizadores legítimos e intrusos
 - ▶ O próprio sistema operativo e utilizadores legítimos / intrusos
- ▶ CPU funciona em dois **modos de operação** para proteger o próprio sistema operativo: núcleo e utilizador
- ▶ Dois tipos principais de protecção: **protecção de memória e controlo de acesso**

▶ 11

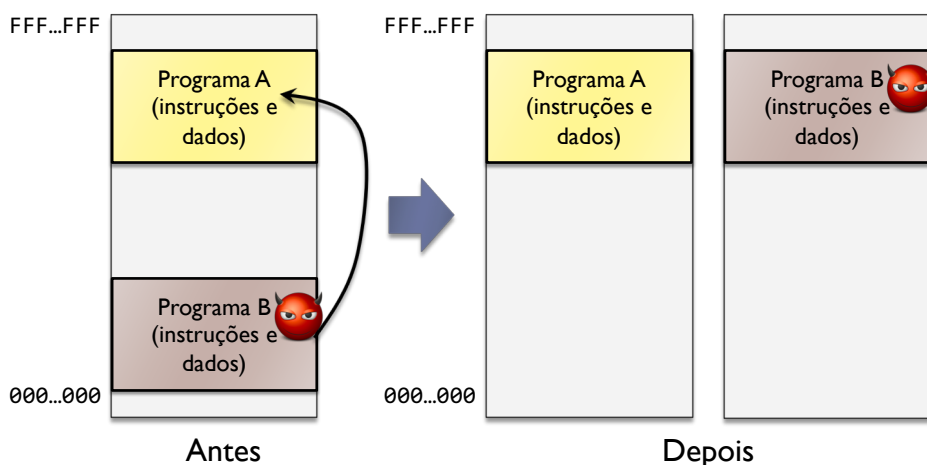
SS - Nuno Santos

2019



Protecção de memória

- ▶ Ideia chave: separar os espaços de endereçamento



▶ 12

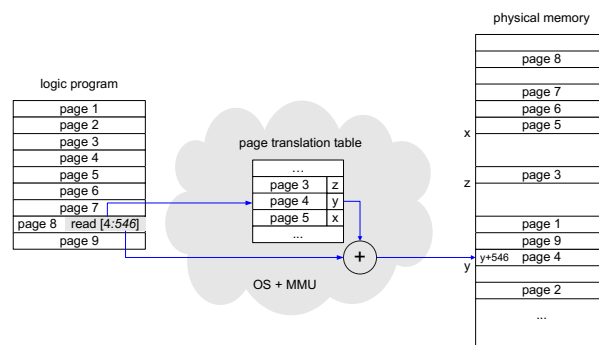
SS - Nuno Santos

2019



Memória virtual

- ▶ Cada programa executa-se num **espaço de endereçamento virtual** isolado de todos os outros
- ▶ Os endereços virtuais são traduzidos automaticamente para endereços físicos



▶ 13

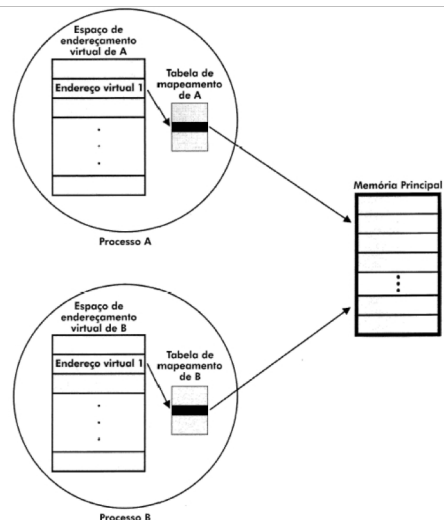
SS - Nuno Santos

2019



Processos

- ▶ Cada aplicação executa-se então no contexto de um **processo**
- ▶ Um processo tem um espaço de endereçamento virtual próprio
- ▶ Garante **isolamento e** protecção de memória entre aplicações



▶ 14

SS - Nuno Santos

2019



Controlo de acesso

- ▶ Como impedir que processos de utilizadores diferentes façam acessos de forma não autorizada?
 - ▶ Por exemplo, impede Alice de ler ficheiros privados do Bob?
- ▶ Modelo de controlo de acesso em Linux
 - ▶ Cada utilizador tem um **username**
 - ▶ Cada username tem um **user id (UID)** e um **group id (GID)**
 - ▶ Cada objecto (ficheiro, pasta, etc.) tem:
 - ▶ Um **dono UID** e um **grupo GID**
 - ▶ **Permissões de acesso**: rwx (read, write, execute) para dono, grupo, e mundo (9 bits)

▶ 15

SS - Nuno Santos

2019



Controlo de acesso

- ▶ Exemplo de permissões de um ficheiro:

```
# ls -l file
-rw-r--r-- 1 root root 0 Nov 19 23:49 file
```

Diagram illustrating the permissions string `-rw-r--r--`:

- File type**: Indicated by the first character `-` (regular file).
- Owner (rw-)**: Indicated by the next three characters `rw-`.
- Group (r--)**: Indicated by the next three characters `r--`.
- Other (r--)**: Indicated by the last three characters `r--`.

Legend:

- `r` = Readable
- `w` = Writeable
- `x` = Executable
- `-` = Denied

- ▶ Esta lista de permissões chama-se **Access Control List (ACL)**
- ▶ Existe um utilizador especial chamado **root**
 - ▶ Tem o UID 0 e é o administrador do sistema
 - ▶ Tem praticamente todos os direitos de acesso ao sistema

▶ 16

SS - Nuno Santos

2019

Mecanismos de protecção nas linguagens de programação

-



Objectivos de segurança numa LP

- ▶ **Type safety:** os dados são sempre manipulados segundo as convenções definidas pelo seu tipo
 - ▶ Ex. Uma variável *short* pode ser atribuído a um *int*, mas uma *string* não pode
- ▶ **Memory safety:** leituras e escritas num determinado objecto devem estar confinadas à memória desse objecto
 - ▶ Não devem ser permitidos overflows
- ▶ **Control flow safety:** saltos no programa só podem ser feitos para endereços válidos
 - ▶ Para o início de um método, ciclo, *else*, depois de um *if*, etc.

▶ 19

SS - Nuno Santos

2019

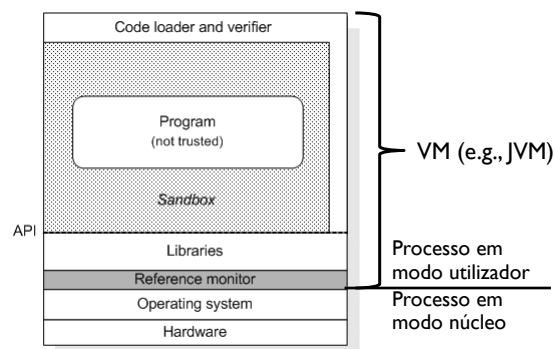


Sandboxes

- ▶ Como proteger o sistema de código móvel cuja proveniência não conhecemos?

- ▶ Em **sandboxes**

- ▶ Código não confiável corre dentro de um ambiente protegido no próprio processo
- ▶ Acessos são controlados por um monitor
- ▶ Controladas por uma máquina virtual (VM)
- ▶ E.g., Java VM



▶ 20

SS - Nuno Santos

2019



Conclusões

- ▶ Muitos ataques são possíveis devido a vulnerabilidades em código legítimo ou devido à execução de código malicioso
- ▶ O sistema operativo implementa medidas de segurança que visam isolar a memória entre aplicações e garantir acessos de acordo com permissões adequadas
- ▶ As linguagens de programação procuram evitar a introdução de erros (bugs) nos programas e a execução de operações não autorizadas por parte dos mesmos



Referências e próxima aula

- ▶ **Bibliografia**
 - ▶ [Correia17] Capítulos 3 e 4
- ▶ **Próxima aula**
 - ▶ Vulnerabilidades em software: Buffer overflows