COPPE UFRJ


# Segurança

Pedro Cruz

EEL770 – Sistemas Operacionais

# Motivação

GTA / UFRJ

- Mundo digital e real não são separados
  - Problemas no mundo digital afetam mundo real
- Atacantes do mundo digital podem afetar mundo real
  - Criar problemas digitais para seus inimigos reais
    - Comandar máquinas
    - Efetuar pagamentos
    - Publicar informações sigilosas

# Motivação de atacantes

GTA / UFRJ

- Dinheiro
  - Extorsão
  - Espionagem industrial
  - Espionagem governamental
- Orgulho
  - Digdin
- Ativismo
  - "Do bem"
  - "Do mal"

# Segurança da informação

GTA / UFRJ

- Confidencialidade
  - Proprietário decide quem tem acesso à informação
- Autenticidade
  - É possível identificar quem tem acesso à informação
- Integridade
  - Modificações na informação são detectadas
- Disponibilidade
  - Informação está sempre disponível para usuários autorizados
- Não-repúdio
  - Operações são sempre mapeáveis em um usuário

# Código é bug

GTA / UFRJ

- Mais funcionalidades exigem mais código
- Mais código gera mais falhas de segurança

- Conclusão lógica:
  - Mais funcionalidades -> mais falhas de segurança

# Sistemas operacionais + redes

GTA / UFRJ

- Ataques costumam chegar pela rede
  - Entender como redes funcionam é importante
    - Alguns assuntos não são de SO
      - Devem ser observados durante esta aula

# Exemplos de ameaças e ataques


- Exposição de dados
  - Ataque à confidencialidade
- Fraude de remetente
  - Ataque à autenticidade
- Falsificação de dados
  - Ataque à integridade
- Negação de serviço (*Denial-of-service*)
  - Ataque à disponibilidade
- Falsificação de assinatura
  - Ataque ao não-repúdio
    - Integridade
    - Autenticidade

# Métodos de segurança

GTA / UFRJ

- Criptografia
  - Estudo de técnicas para comunicação segura entre duas partes na presença de uma terceira parte
- Encriptação
  - Conversão de informação legível para aparente absurdo
- Endurecimento
  - Adição de mecanismos que procuram fazer os softwares se comportarem de forma adequada

# Sistemas confiáveis

GTA / UFRJ

- Atendem a uma lista de requisitos de segurança
- Possuem uma Base Computacional Confiável (*Trusted Computing Base* – TCB)
  - Base mínima para fazer valer as regras de segurança
    - TCB comprometida é computador comprometido

# *Trusted Computing Base* – TCB

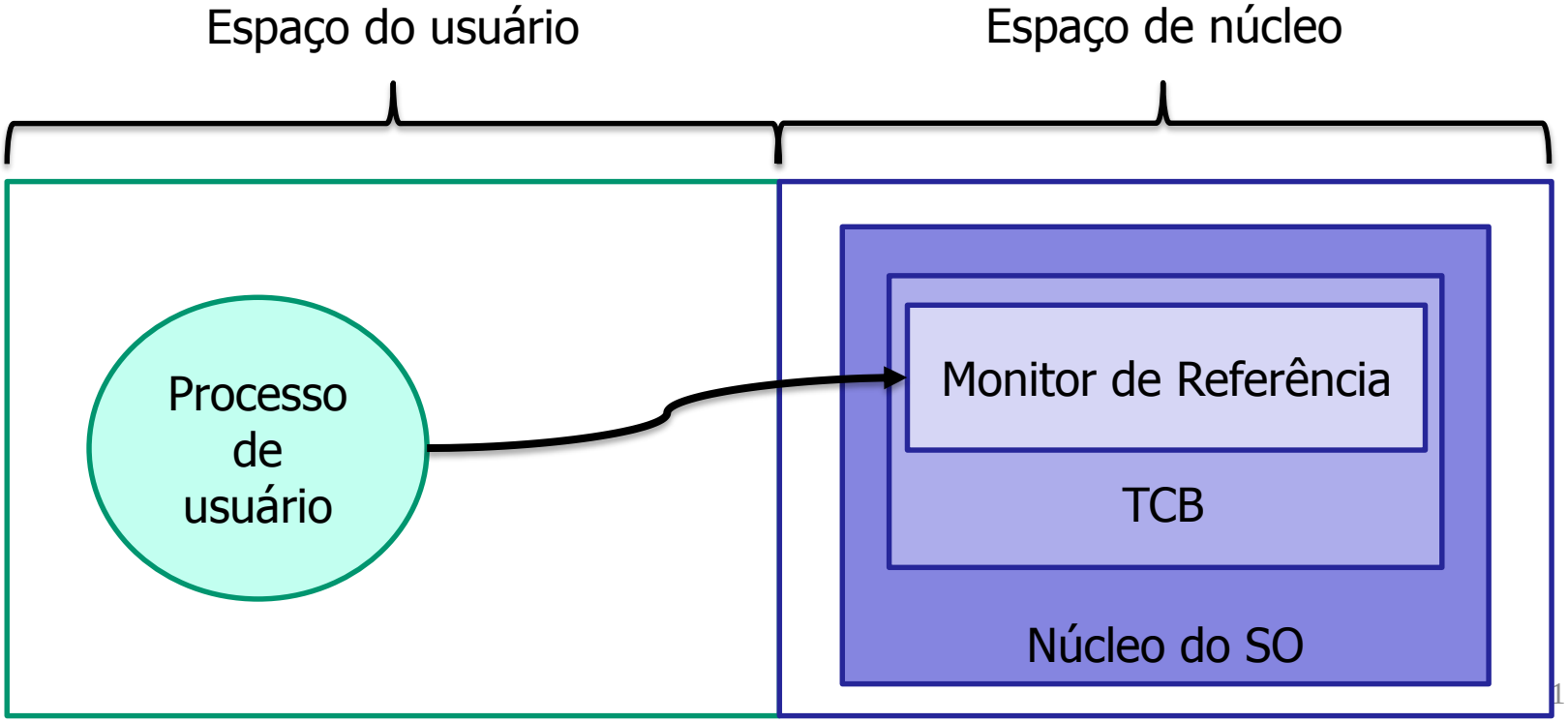
GTA / UFRJ

- Hardware
  - Quase tudo, exceto I/O
- Software
  - Parte do núcleo do SO
    - Criação de processos
    - Gerenciamento de memória
    - Gerenciamento de arquivos
  - Programas de usuário
    - Programas de usuário que tenham poder de superusuário

# Monitor de referência

GTA / UFRJ

- Módulo central de decisões de segurança
  - Chamadas de sistema passam pelo Monitor de Referência
  - Monitor de Referência decide sobre execução da chamada



# Domínios de proteção

GTA / UFRJ

- Recursos precisam ser protegidos
  - Abstraídos como objetos
  - Identificados por um nome único
- Processos podem executar operações
  - Operação é permitida para um objeto e para um processo
- Domínio
  - Conjunto de pares (objeto, direito)
  - Relacionado a um usuário ou grupo de usuários

# Exemplo de domínios

GTA / UFRJ

| Domínio | Arquivo1 | Arquivo2 | Domínio1 | Domínio2 | Domínio3 |
|---|---|---|---|---|---|
| **1** | Leitura | Escrita<br>Leitura | | Entrada | |
| **2** | Leitura<br>Escrita | | | | |
| **3** | | Execução<br>Leitura | | | |

# Princípio da menor autoridade

GTA / UFRJ

- *Principle of Least Authority – PLA*
  - Cada domínio deve ter somente os objetos e direitos mínimos para realizar suas funções

# Armazenando domínios

GTA / UFRJ

- Domínios são uma matriz esparsa
  - Pode ser armazenada como vetor de vetores
    - Número de células= |Domínios| * |Objetos|
      - Maior parte vazia
  - Pode ser armazenada como listas de listas
    - Por linha ou por coluna
      - Lista de controle de acessos
        - » Cada objeto guarda quem tem permissões sobre si
      - Listas de capacidades
        - » Cada processo guarda sobre quem tem permissões

# Lista de controle de acessos (*Access Control List* – ACL)


- A cada objeto é associada uma lista
  - Lista contém os domínios e suas permissões
    - Exemplo:
      - Arquivo1 -> UserA:RW; UserB:R
      - Arquivo2 -> UserB->RWX;UserC:X

# Grupos de usuários e papéis

GTA / UFRJ

- Grupos de usuários
  - Identificam vários usuários
  - Aparecem na ACL
    - Exemplo:
      - Arquivo1 -> UserA,G1:RW; UserB,G2:R
      - Arquivo2 -> UserB,G2->RWX;UserC,G1:X
      - Arquivo3 -> UserB,Gadmin->RWX
- Papéis
  - Contextos nos quais usuários estão inseridos
    - Afetam suas permissões

# Capacidades

GTA / UFRJ

- Cada processo possui uma lista
  - Quais operações podem ser executadas sobre cada objeto
    - **Capacidades**

| Processo | Capacidades |
|---|---|
| 1 | F1:RW;F2:R;F5:RWX |
| 2 | F2:RW;F3:RWX |
| 3 | F1:R;F3:RWX |

# Proteção de capacidades

GTA / UFRJ

- Capacidades devem ser protegidas dos usuários
  - Usuário que forje capacidades compromete a segurança

- Proteção das capacidades pode ser:
  - Arquitetura marcada (*tagged architecture*)
  - Capacidades em espaço de núcleo
  - Integridade de capacidades por criptografia

# Arquitetura marcada

GTA / UFRJ

- Cada palavra de memória possui duas partes
  - Dados
    - Podem ser lidos e executados
  - Marcação
    - Não são executados
    - Só podem ser alterados pelo operacional
- Marcação pode indicar se palavra é alterável pelo usuário
  - Se dados possuem uma capacidade, não são alteráveis

# Armazenando capacidades

GTA / UFRJ

- Espaço de núcleo
  - Apenas SO pode mexer
  - Processos de usuário as referenciam por índices
- Espaço de usuário
  - Entregam capacidade ao SO quando precisam realizar um pedido
    - Processos de usuário podem mexer
    - SO deve garantir que não haja alteração pelos processos

# Integridade de capacidades protegida por criptografia


- SO entrega ao processo uma tupla
  - (obj, direitos, conferência)
    - Só o SO sabe gerar a *conferência*

- Pedido pelo processo deve conter a tupla
  - SO confere se tudo faz sentido
    - Se houve alteração, a *conferência* não bate
      - **Função criptograficamente segura de mão única**

| Objeto | Direitos | f(objeto, direitos, segredo) |
|---|---|---|

# Função criptograficamente segura de mão única

GTA / UFRJ

- f(x) = y
  - Fácil de obter computacionalmente
- $x = f^{-1}(y)$
  - "Impossível" de obter computacionalmente

# Garantia de integridade

GTA / UFRJ

- SO sabe, inicialmente, a capacidade de um processo
  - (objeto, direitos)
- SO concatena uma senha secreta à capacidade
  - (objeto, direitos, senha)
- SO gera uma conferência aplicando a função à nova tupla
  - conferência = f((objeto, direitos, senha))
- SO entrega (objeto, direitos, conferência) ao processo
- Processo faz pedido com (objeto, direitos, conferênciaP)
- SO gera conferênciaS = f(objeto, direitos, senha)
  - Se conferênciaS == conferênciaP => tudo certo
  - Se não => processo mentiu

# Ataque por repetição de mensagem

GTA / UFRJ

- Sistema garante
  - Autenticação
    - Atacante não sabe gerar mensagens autênticas
  - Integridade
    - Atacante não sabe alterar mensagens existentes
- Atacante pode
  - Capturar mensagem que ele sabe o que significa
  - Replicar a mensagem quando lhe for adequado
- Exemplo
  - Atacante captura mensagens até que uma comporta se abra
  - Atacante envia mensagem ou sequência anterior à abertura

# Resolvendo ataque de repetição de mensagem


- Cada mensagem é numerada
  - Numeração é protegida por mecanismos de integridade
- Mensagens repetidas não são aceitas

# Autenticação

GTA / UFRJ

- Verificar se um usuário é ele mesmo
  - Algo que o usuário sabe
    - Senhas
  - Algo que o usuário tem
    - Objetos
  - Algo que o usuário é
    - Biometria
  - Algo que o usuário sabe fazer
    - Reconhecimento de voz

# Autenticação por desafio

GTA / UFRJ

- Procedimento de autenticação
  - Usuário possui uma senha
  - Serviço propõe um desafio para autenticação
    - Resposta depende da senha

- A cada nova autenticação, um novo desafio é proposto
  - Senha não pode ser "clonada"

# Ataques a software

GTA / UFRJ

- Transbordamento de buffer (buffer overflow)
- Ataques de reutilização de código
- Ataques por string de formato
- Transbordamento de inteiro
- Injeção de comando

# Buffer overflow

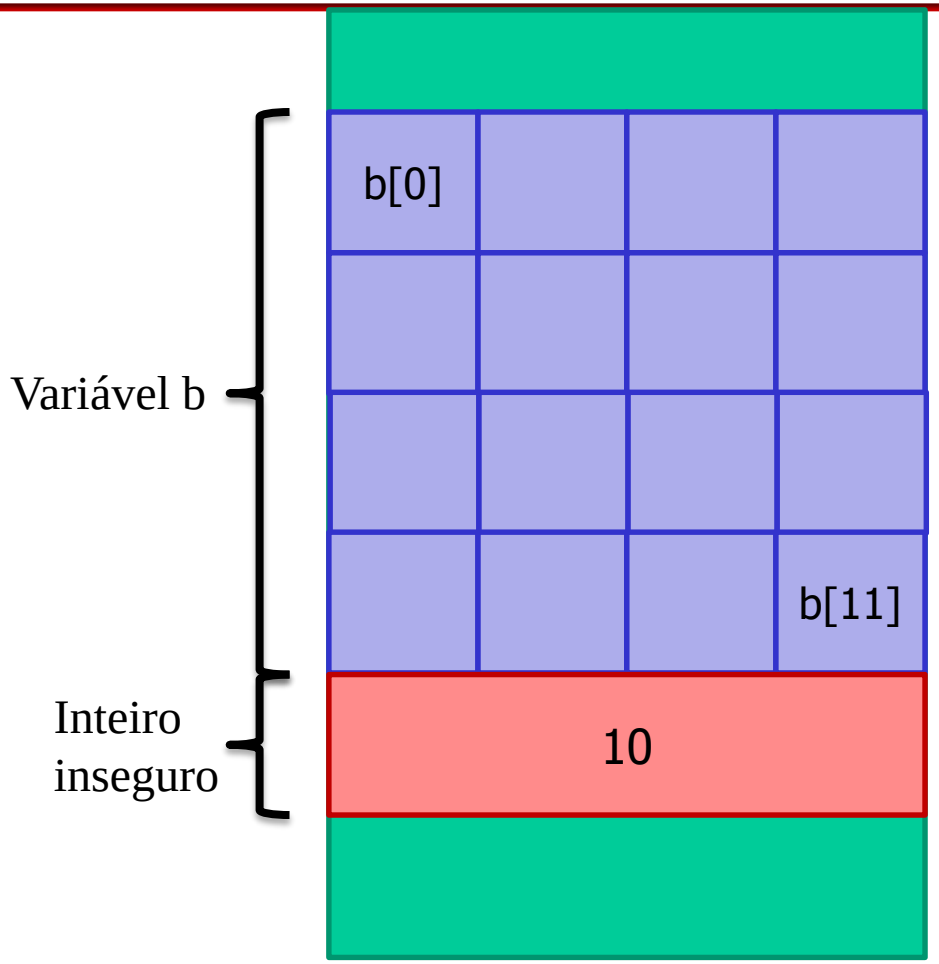
GTA / UFRJ

- Processo faz referência à memória
  - SO verifica se processo tem acesso à memória referenciada
  - SO não verifica se há bom uso da memória
- Entrada inesperada provoca funcionamento inadequado

# Transbordamento de buffer

GTA / UFRJ

```
#include <stdio.h>
int func()
{
    int unsafe_int = 10;
    char b[12];
    scanf("%s", buffer);
}
```

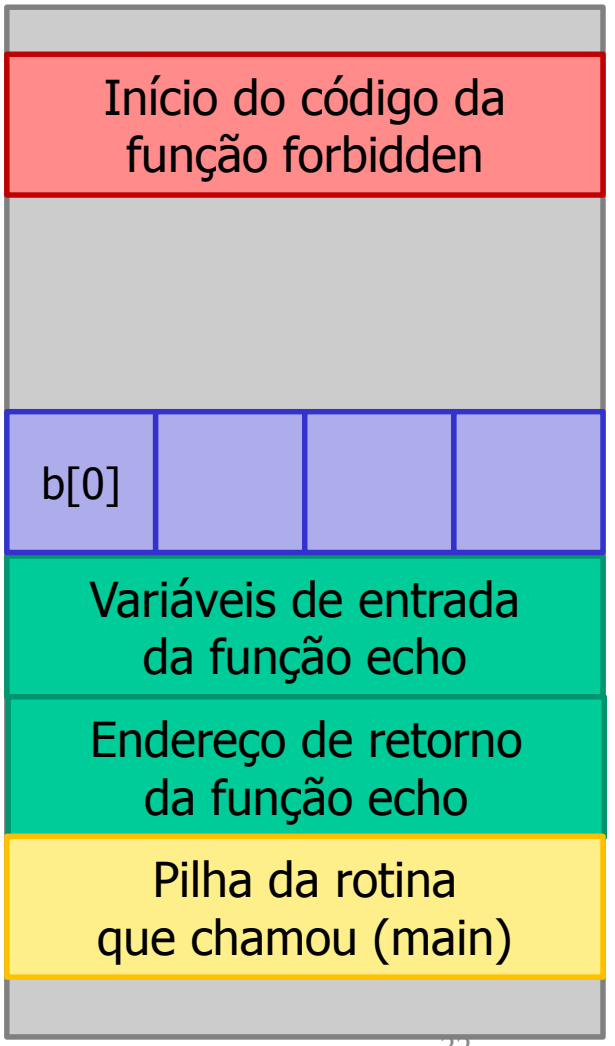



E se, ao invés de um inteiro, for um ponteiro de função?

# Transbordamento de buffer na pilha

GTA / UFRJ

```
#include <stdio.h>
void forbidden(){
    //segredo
}
void echo(int i){
    char b[4];
    scanf("%s", buffer);
}
int main(){
    echo(3);
    return 0;
}
```



# Proteção contra transbordamento de buffer

GTA / UFRJ

- Canários de pilha
  - Valor aleatório colocado adjacente aos endereços de retorno
    - A cada retorno, verificação é feita
      - Se valor for igual ao colocado, segue
      - Se valor for diferente do colocado, para
- Randomização de endereço de função
  - Funções não são guardadas de forma contígua
- Marcação de memória
  - Prevenção de execução de dados
    - *Data execution prevention – DEP*
      - W XOR X

# Canários de pilha

GTA / UFRJ

- Valor aleatório colocado adjacente aos endereços de retorno
  - A cada retorno, verificação é feita
    - Se valor for igual ao colocado, segue
    - Se valor for diferente do colocado, para

Não são infalíveis

# Ataques de reutilização de código

GTA / UFRJ

- Retorno à *libc*
  - Atacante utiliza a biblioteca padrão a seu favor
  - Exemplo
    - Utilizar transbordamento de buffer
    - Chamar a função *system(*char)*
      - » Função executa **char* na linha de comando
    - Argumento deve ser o comando que se deseja executar
- Atacante pode retornar para qualquer fragmento de código
  - Não precisa ser o início de uma função
    - *Return-oriented Programming ROP*

# Ataques por string de formato

GTA / UFRJ

```
char s[100], g[100] = “ola “;
gets(s);
strcat(g,s);
printf(g);
```

- g será formatado por printf
  - “%n” conta quantos caracteres tem na string até o “%n” e escreve em uma variável; “%08x” escreve em hexadecimal
  - Se variável não existe, o próximo endereço é usado =s
- Atacante usa “%08x %08x .... %08x %n” para alterar um endereço de memória a qualquer distância do printf

# Transbordamento de inteiro

GTA / UFRJ

- Aritmética de inteiros não confere transbordamento
  - Se um inteiro transbordar, pode até ficar negativo
- Atacante pode forçar um transbordamento
  - Utilizar consequências do transbordamento e causar danos

# Ataques por injeção de comando

GTA / UFRJ

- Programador preguiçoso executa comando no shell utilizando entrada do usuário
- Atacante realiza entrada maliciosa
- Exemplo:
  - Código:

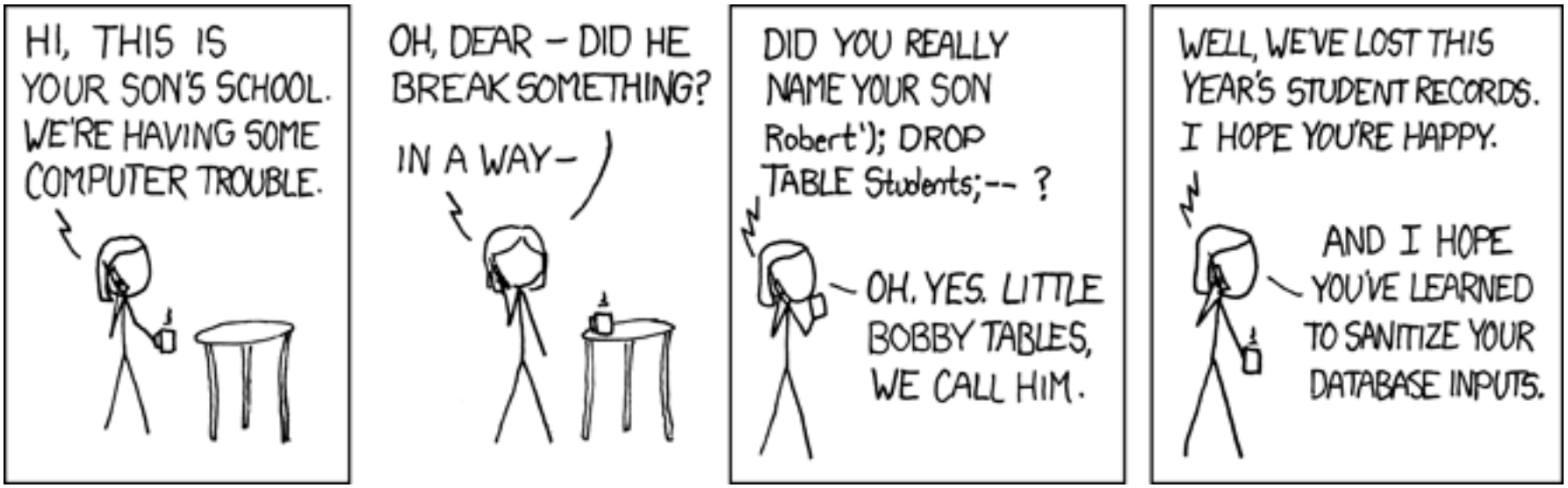
```
input_file << std::in;
system( “cp  “+input_file+ “ tmp_file” )
```

  - Entrada:

```
input.txt cp.txt; rm -rf /
```

# Injeção de SQL

GTA / UFRJ



COPPE UFRJ


# Segurança

Pedro Cruz

EEL770 – Sistemas Operacionais