Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Técnicas de Virtualização de Sistemas, Inverno de 2022/2023

Segunda série de exercícios

Resolva os exercícios propostos, colocando a resolução de cada um dos exercícios em diretorias separadas (`se2/ex1/`, `se2/ex2/`, etc.). Para as questões teóricas, escreva as respostas em ficheiros `.md` em *markdown*. Utilize o conteúdo do ficheiro zip em anexo como modelo para a resolução dos exercícios 3 e 4, adicionando os ficheiros ao repositório de grupo. Não adicione o ficheiro zip ao repositório.
No final coloque a tag **`SE2`** no repositório GitHub.

1. Considere o manual de referência de arquitetura do processador ARM64 disponível em:
   https://developer.arm.com/documentation/ddi0487/ia/
   Leia as secções B1.3.1, C6.2.365 e D1.1 (3 primeiros e 2 últimos parágrafos) do manual ARM e a documentação Linux disponível via "`man syscall`" e explique sucintamente como se executa uma chamada de sistema em sistemas Linux para processadores ARM64, em particular:
   - Como se indica qual a chamada de sistema a executar?
   - Como são passados os argumentos e onde é devolvido o valor de retorno?
   - Como se provoca a transição do código da aplicação para o código do *kernel* e que nível se altera no processador?

2. Considere o manual de programação de sistema da arquitetura AMD64 disponível em:
   https://www.amd.com/system/files/TechDocs/24593.pdf
   Consultando as secções 5.3.3, 5.4 e 5.6, explique como se aplicam as proteções de *Read/Write*, *User/Supervisor* e *No Execute* quando, na tradução de um endereço, a definição dos respectivos *bits* de proteção não é igual nas entradas das tabelas em todos os 4 níveis de tradução (por exemplo, as entradas das tabelas PML4 e PDP proíbem acessos de escrita, mas as entradas das tabelas PD e PT permitem-nos).

3. O programa fornecido em anexo, em `se2/ex3`, apresenta um sumário de informação sobre um ficheiro em formato ELF (por exemplo, um executável ou uma biblioteca de ligação dinâmica). Execute `make` e experimente o programa gerado para ver o que faz. Exemplo: `./elf-summary /bin/ls`
   Modifique **apenas** o ficheiro `isel-calls.s` para que todas as funções deste ficheiro consistam em chamadas de sistema realizadas com a instrução `syscall`. Para cada uma das funções, apague a instrução `jmp` que lá está colocada e substitua-a pela devida invocação ao *kernel* via `syscall`.
   Encontra uma lista com as chamadas de sistema e a respetiva convenção de chamada aqui:
   https://chromium.googlesource.com/chromiumos/docs/+/HEAD/constants/syscalls.md
   Note que a convenção de chamada é muito semelhante, mas não igual, à utilizada pelo compilador de C.

   [OPCIONAL] Modifique o programa em C para mostrar também a lista de secções do ficheiro ELF, indicando, para cada uma, o respetivo nome e o endereço virtual a que está associada.
   NOTA: *este passo opcional não requer mais alterações no ficheiro* `isel-calls.s`

   Informação útil:
   - System V Application Binary Interface [capítulo 4]
   - `man elf`

(continua)

4. Complete o programa do ficheiro `se2/ex4/prog.c` em anexo, nos pontos indicados, de modo a provocar os seguintes efeitos na memória atribuída ao processo, visíveis em `/proc/<pid>/smaps` :
    a. Aumento do *resident set* (`Rss`) em cerca de 3MB na região que mapeia dados não iniciados (`.bss`).
    b. Acesso a 256 *bytes* de dados iniciados (`.data`) com o máximo impacto em páginas `Private clean`.
    c. Redução do `Pss` dos dados não-iniciados (`.bss`) para cerca de 1MB por 30 seg., mantendo o `Rss`.
    d. Criação de região de dados com 96KB no espaço de endereçamento, com o respetivo `Rss` a zero.
    e. Aumento do `Rss` da nova região de dados para 96KB.
    f. Criação de duas novas regiões, uma de código e outra de dados, no espaço de endereçamento.
    g. Aumento de páginas `Private dirty` na região que mapeia a secção de dados criada em f).
    h. [OPCIONAL] O que acontece no espaço de endereçamento quando se cria uma *thread*?

       Informação útil:
       - Criação de *thread* com `pthread_create`
       - Bloqueio/desbloqueio de *thread* com `sem_wait`/`sem_post`
       - Observação de detalhes de operação com `strace` (ex.: `strace ./prog`)

    NOTA: Resolva o exercício 4 sobre a base definida no ficheiro `se2/ex4/prog.c` em anexo, adicionando outros ficheiros fonte e auxiliares quando necessário.

ISEL, 14 de outubro de 2022

Data limite de entrega: 29 de outubro de 2022