



Observação de Ataques contra a Memória do Kernel Android: Desafios e Soluções

Cláudio Torres Júnior, Jorge Correia, João
Pincovsky, Marco Zanata, André Grégio

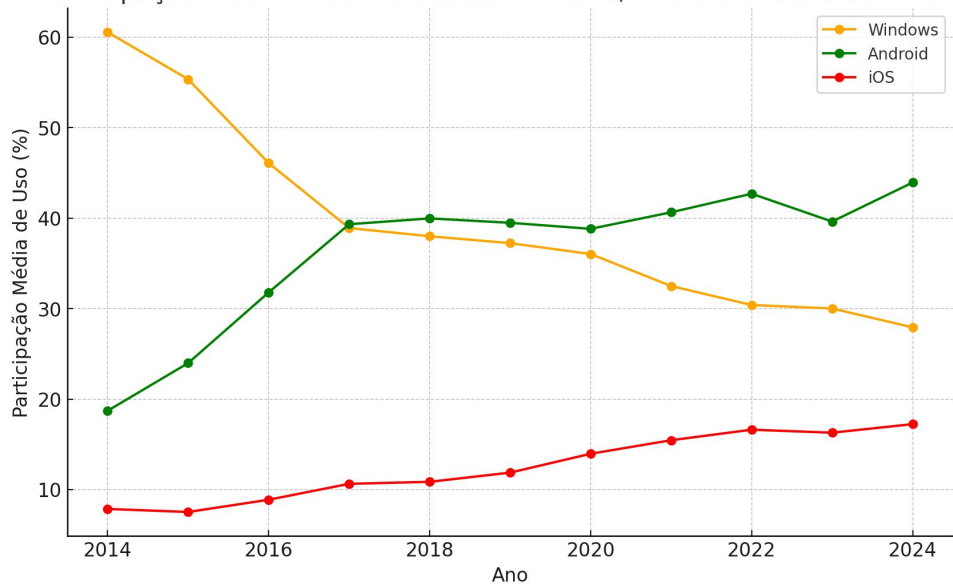
Universidade Federal do Paraná (UFPR)



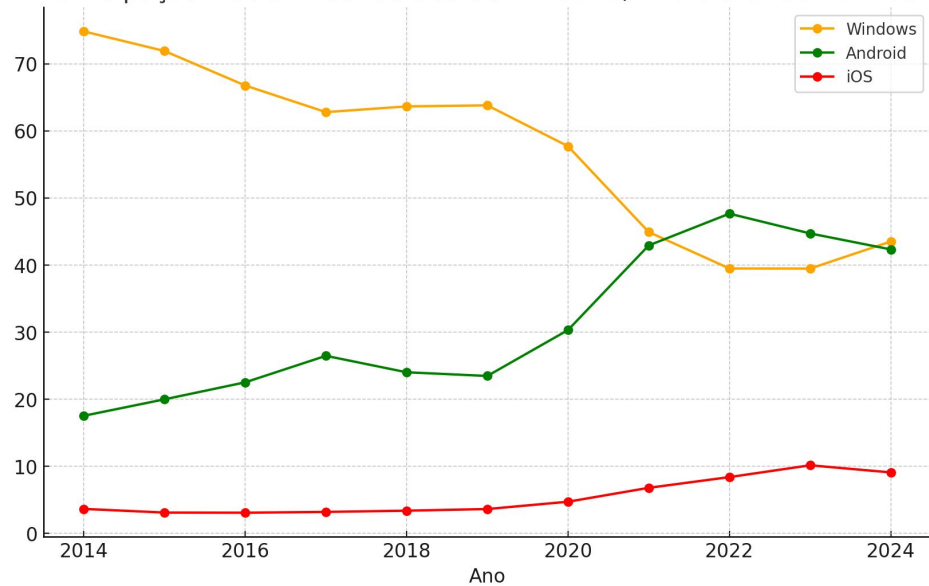
Motivação

● Prevalência do Android no Mercado Global e brasileiro

Participação Média Anual de Uso de Windows, Android e iOS Globalmente




Participação Média Anual de Uso de Windows, Android e iOS no Brasil

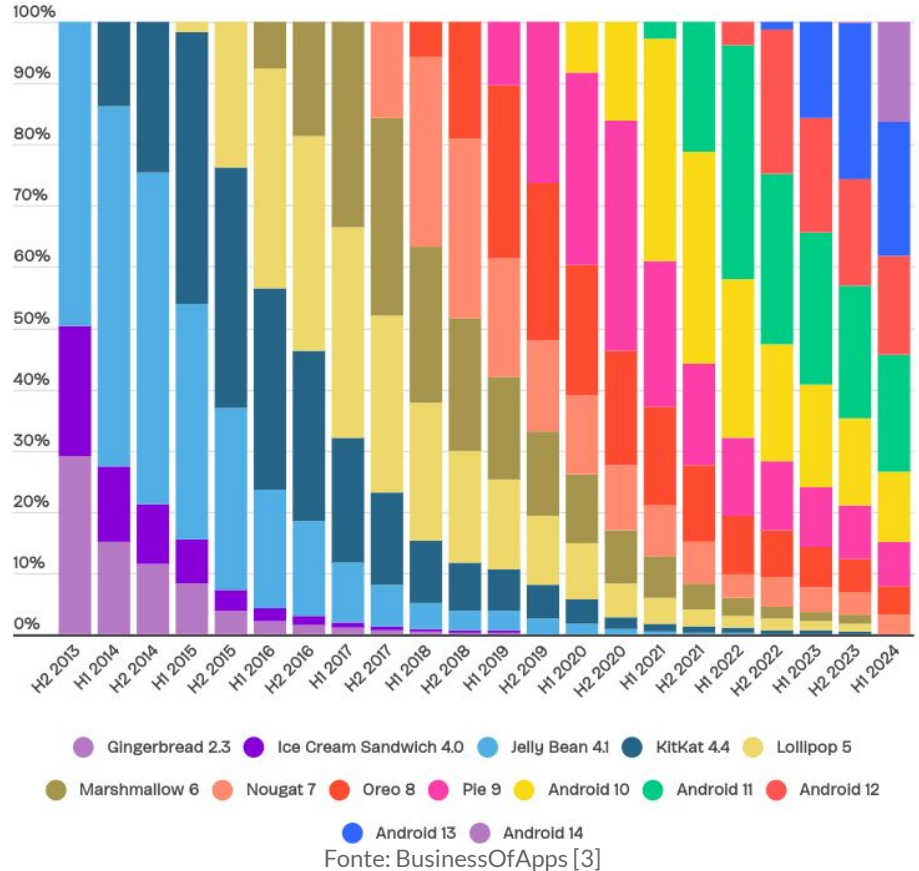
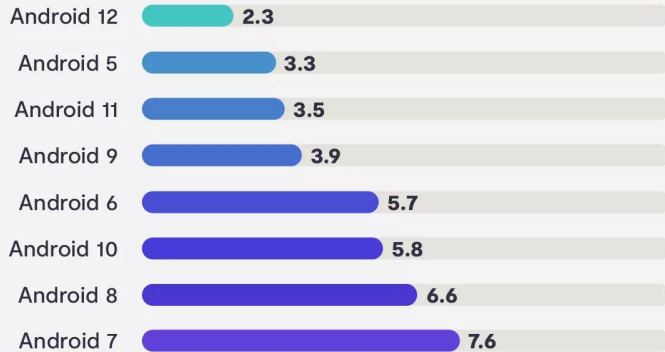


Fonte: StatCounter [1]

● Pontuações de Vulnerabilidade do Android ao Longo das Versões e Distribuição dos Dispositivos ao Longo do Tempo

Android: Vulnerability Scores

Vulnerability scale
Most secure  Least secure



- **Produtos com Maior Número de Vulnerabilidades Distintas nos últimos 10 anos**

	Product Name	Vendor Name	Product Type	Number of Vulnerabilities
1	Debian Linux	Debian	OS	8796
2	Android	Google	OS	7152
3	Linux Kernel	Linux	OS	5301
4	Fedora	Fedoraproject	OS	5116
5	Ubuntu Linux	Canonical	OS	4093
6	Windows Server 2016	Microsoft	OS	3624
7	Chrome	Google	Application	3493
8	Iphone Os	Apple	OS	3402
9	Mac Os X	Apple	OS	3206
10	Windows Server 2019	Microsoft	OS	3181

Fonte: cvedetails [2]

● Produtos com Maior Número de Vulnerabilidades Distintas em 2023

	Product Name	Vendor Name	Product Type	Number of Vulnerabilities
1	Android	Google	OS	1422
2	Windows Server 2022	Microsoft	OS	572
3	Windows Server 2019	Microsoft	OS	548
4	Fedora	Fedoraproject	OS	545
5	Windows 11 21h2	Microsoft	OS	516

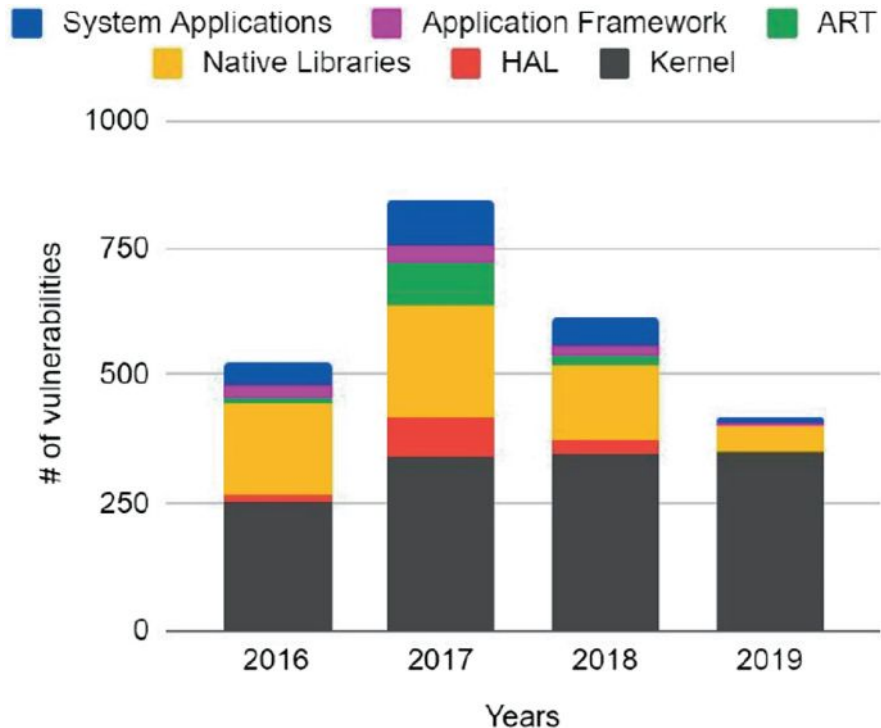
● Em 2024

11	Macos	Apple	OS	309
12	Windows 10 1607	Microsoft	OS	293
13	Android	Google	OS	287
14	Kernel	Linux	OS	283
15	Windows Server 2012	Microsoft	OS	278

Fonte: cvedetails [2]

Problemas

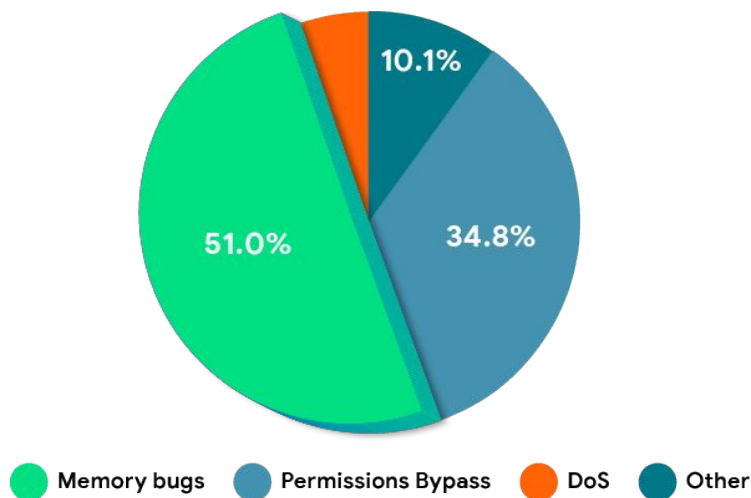
- Vulnerabilidades na Pilha do Android



Fonte: Garg, S., & Baliyan, N. (2022) [5]

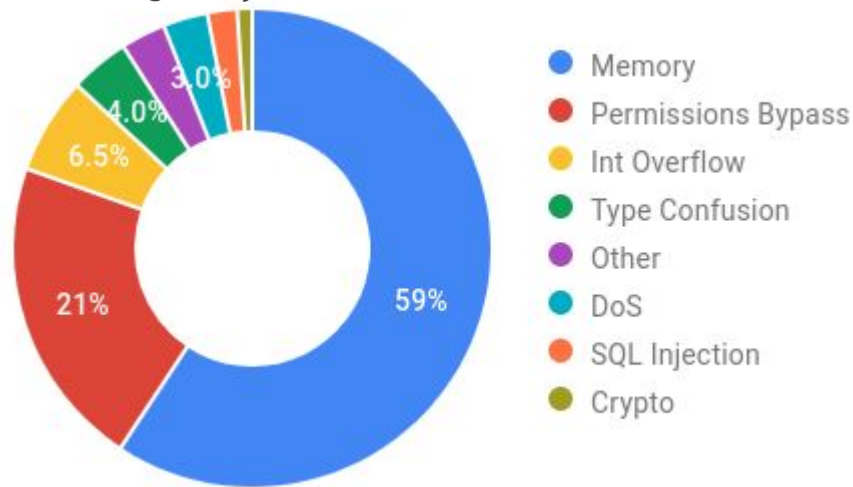
Problemas

Contribuição de falhas de segurança de memória para vulnerabilidades no Android



Fonte: Android memory bugs. (2024) [6]

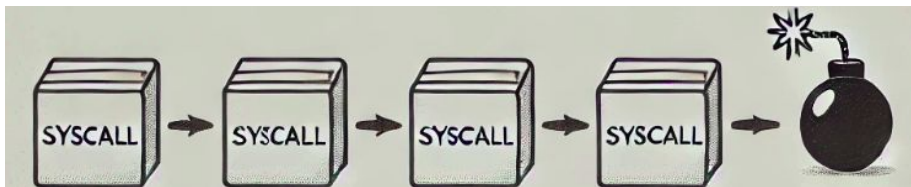
Tipos de vulnerabilidades críticas e graves corrigidas nos Boletins de Segurança Android em 2019



Fonte: Google Security Blog [7]

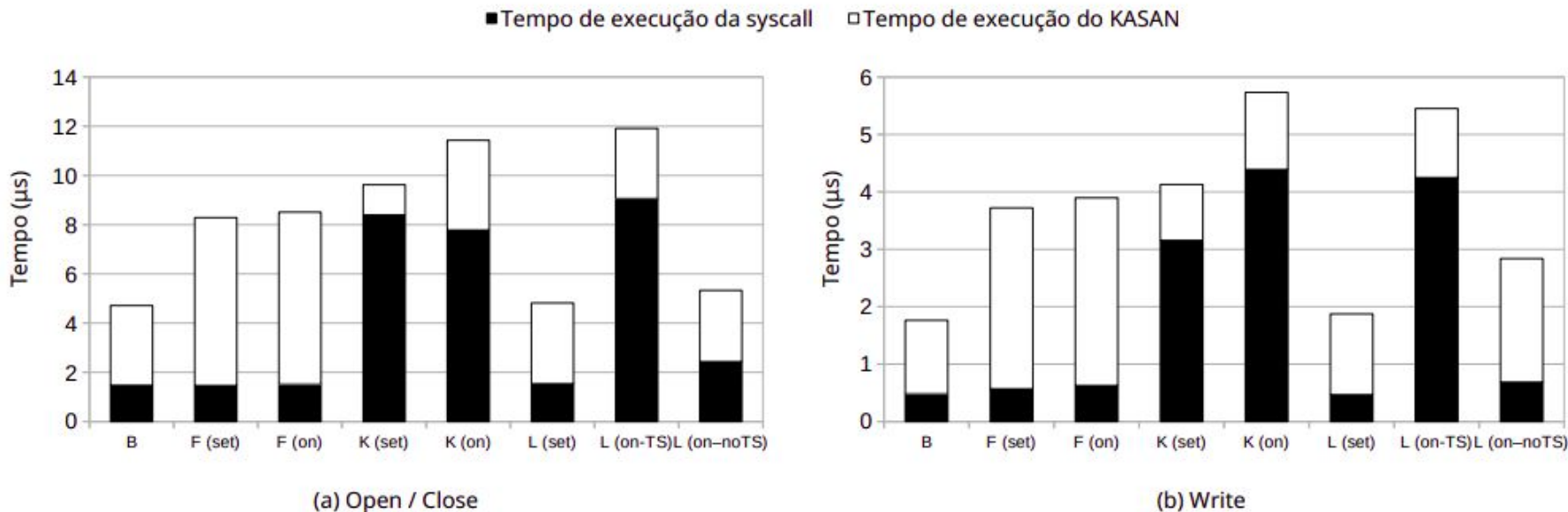
Soluções atuais

- Ferramentas de *tracing* para ataques à memória
 - Auxílio de detectores de erros na memória
 - KASAN - *Kernel Address Sanitizer*
 - *strace*, *ftrace*, *kprobes*, LSM*
- Geração de ‘caminhos’ do ataque
 - Sequências de *syscalls*/funções até a falha



Soluções atuais

- Tempo de execução de syscalls



Tempo de execução de chamadas de sistemas utilizando combinações de tracers com KASAN habilitado ou desabilitado, onde B é o baseline, F é ftrace, K é kprobe e L é LSM; (set) significa que a ferramenta foi apenas instalada, (on) que o tracing está habilitado

Soluções atuais

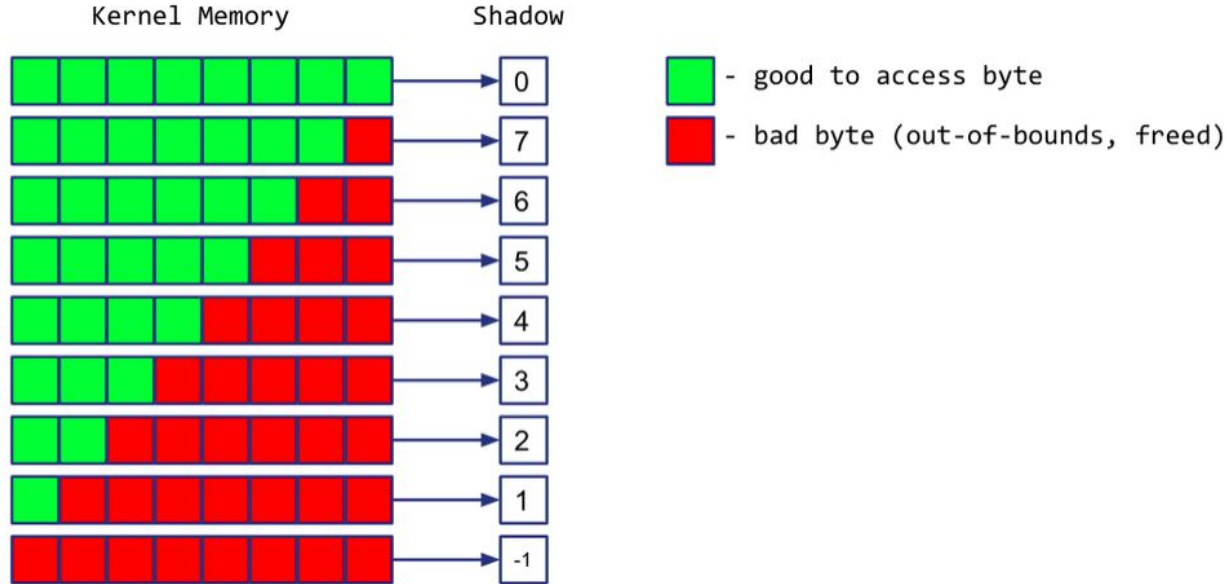
- Problemas
 - Alto overhead em todo o sistema
 - Quebra da execução na primeira falha
 - Visualizações do intuito de um *exploit* interrompidas

Solução proposta - Prova de Conceito

- Mas antes, um pouco de KASAN

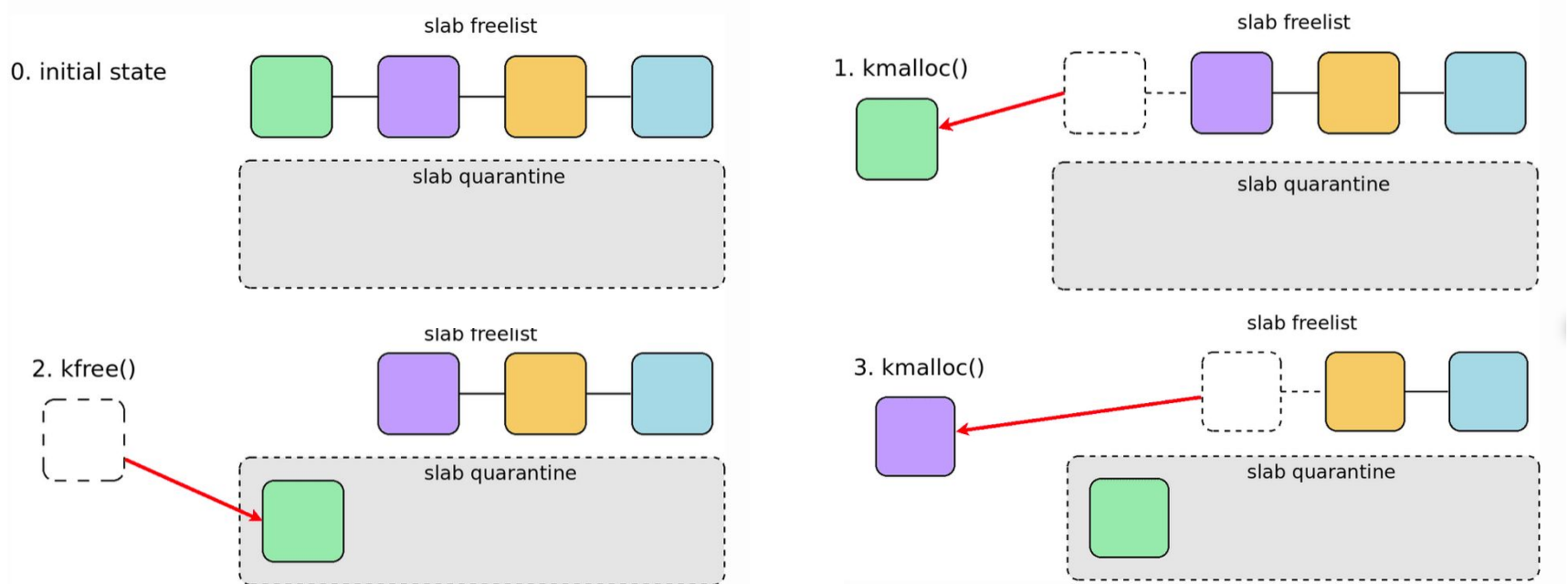
Solução proposta - Prova de Conceito

- Para cada 8 Bytes, usa-se 1 Byte shadow



Solução proposta - Prova de Conceito

- Memória previamente alocada vai para quarentena



Solução proposta - Prova de Conceito

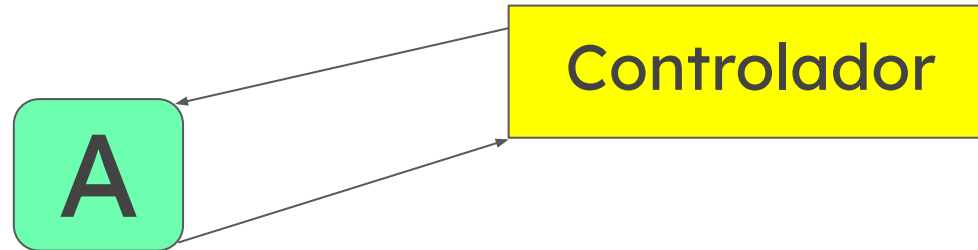
- Utilização de funcionalidades do KASAN
 - Instrumentação de *loads/stores*
 - Modificação do método de alocação
 - Novo formato de quarentena de memória

Solução proposta - Prova de Conceito

- Utilização de Kprobes
 - *Tracing* de *syscalls* e funções específicas
 - Controle dos argumentos das funções
 - Controlar endereços alocados/liberados
 - Permite criar o método de quarentena

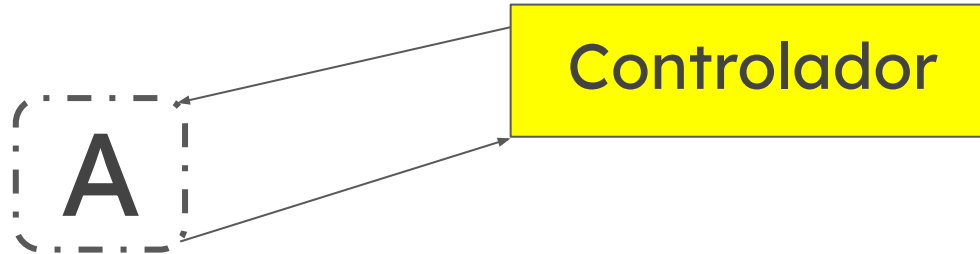
Solução proposta - Prova de Conceito

- Use-After-Free (UAF)



Solução proposta - Prova de Conceito

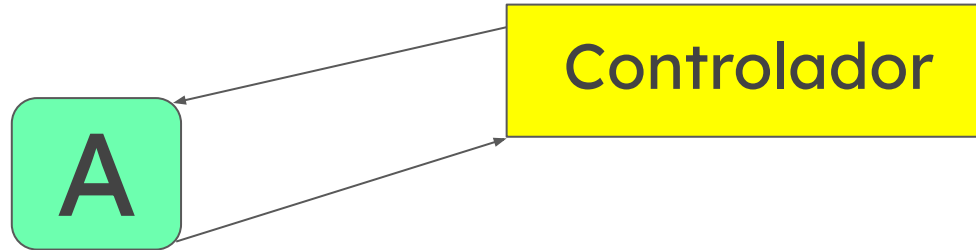
- Use-After-Free (UAF)
 - Endereço A é liberado mas não avisa o controlador



- Região A volta para a lista de endereços disponíveis

Solução proposta - Prova de Conceito

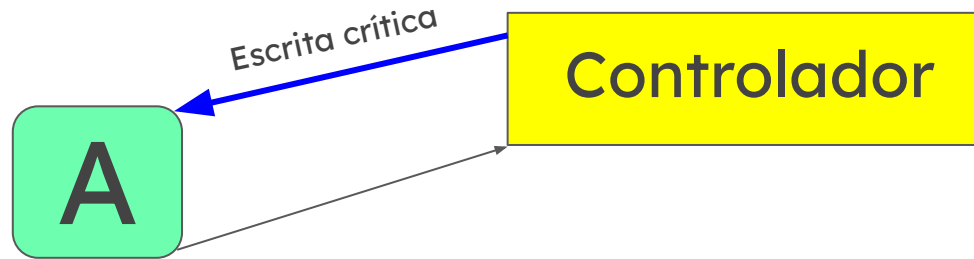
- Use-After-Free (UAF)
 - Um objeto é alocado com tamanho próximo à região A



- Por ter tamanho próximo à região A, o SO entrega novamente a região A

Solução proposta - Prova de Conceito

- Use-After-Free (UAF)
 - Atacante avisa controlador para ser atualizado (controlador ainda acredita que o objeto A antigo está nessa região)



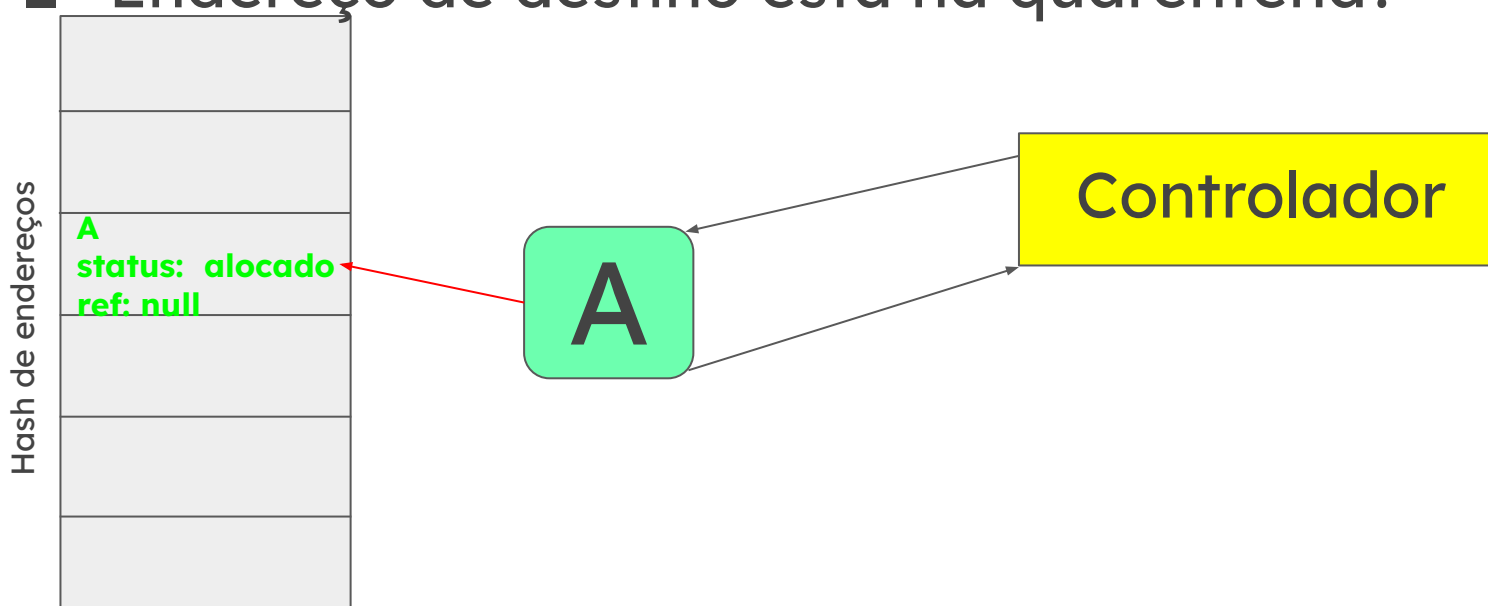
- Quem controla o novo objeto na região A controla o conteúdo crítico

Solução proposta - Prova de Conceito

- Identificando UAF
 - Toda escrita e leitura são verificadas
 - Endereço de destino está na quarentena?

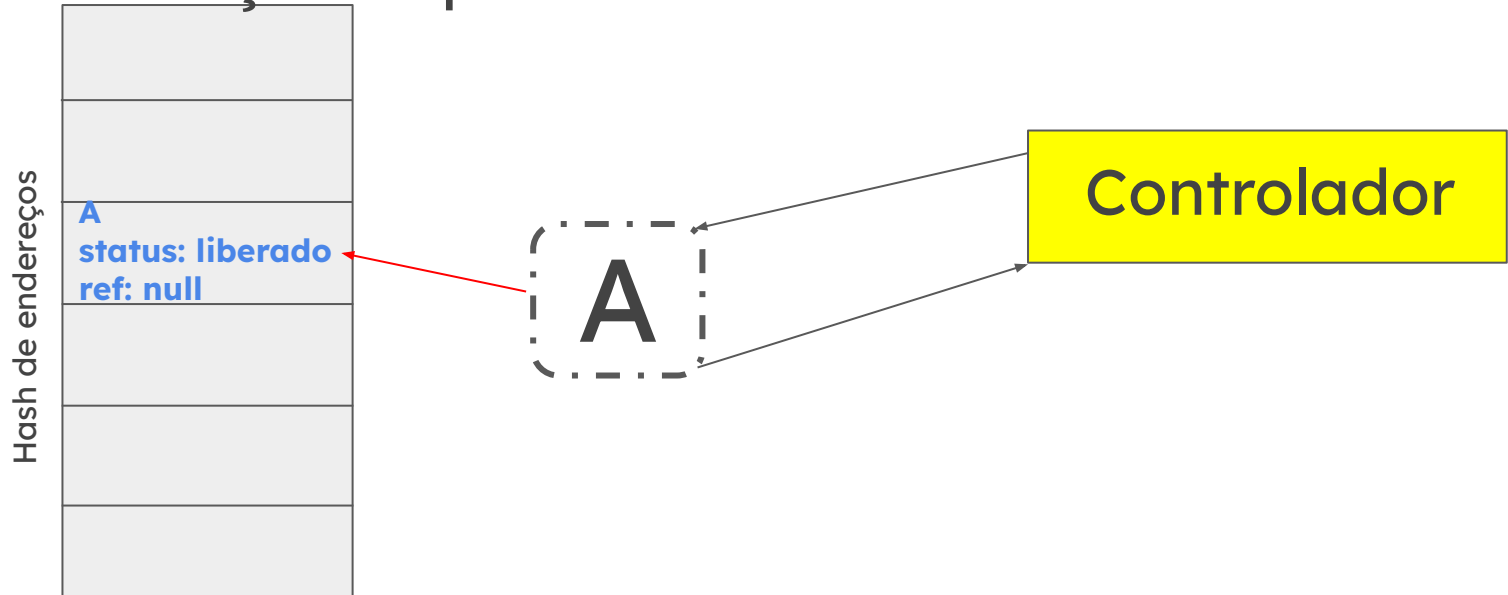
Solução proposta - Prova de Conceito

- Identificando UAF
 - Toda escrita e leitura são verificadas
 - Endereço de destino está na quarentena?



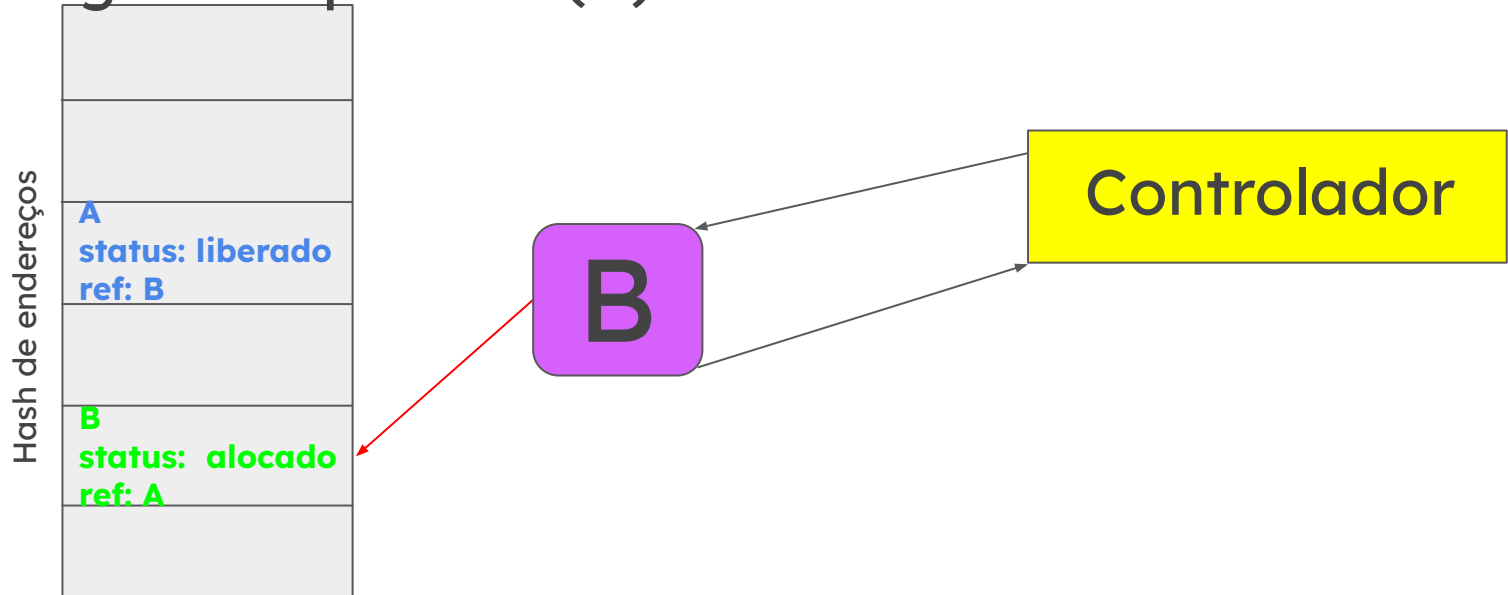
Solução proposta - Prova de Conceito

- Identificando UAF
 - **A** é liberada mas não volta para a lista de endereços disponíveis



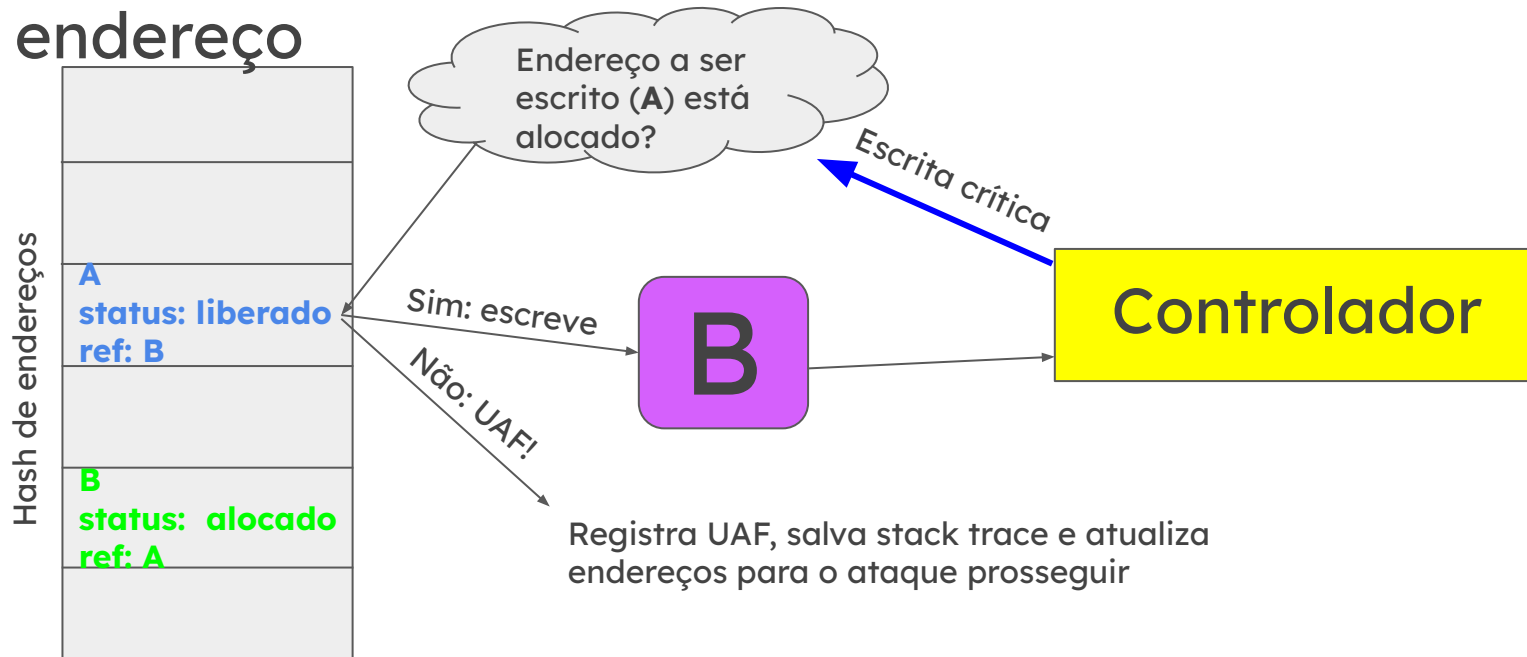
Solução proposta - Prova de Conceito

- Identificando UAF
 - O próximo objeto alocado pega a próxima região disponível (**B**)



Solução proposta - Prova de Conceito

- Identificando UAF
 - No momento da escrita critica, verificamos o endereço



Solução proposta - Prova de Conceito

- Exemplo de strace + KASAN

```
openat(AT_FDCWD, "/dev/binder", O_RDONLY) = 3
epoll_create1(0) = 4
epoll_ctl(4, EPOLL_CTL_ADD, 3, {EPOLLIN, {u32=0, u64=0}}) = 0
ioctl(3, BINDER_THREAD_EXIT, 0) = 0
mprotect(0x75cb3dacc000, 4096, PROT_READ|PROT_WRITE) = 0
mprotect(0x75cb3dacc000, 4096, PROT_READ) = 0
munmap(0x75cb3dacc000, 4096) = 0
exit_group(0) = ?
+++ exited with 0 +++
```

Solução proposta - Prova de Conceito

- Exemplo de strace + KASAN

```
[ 99.819717] =====  
[ 99.820087] BUG: KASAN: use-after-free in _raw_spin_lock_irqsave+0x2d/0x4e  
[ 99.820087] Write of size 4 at addr ffff8880587be5c8 by task cve-2019-2215-t/5602  
  
[ 99.820087] Call Trace:  
[ 99.820087] dump_stack+0x93/0xcd  
[ 99.820087] ? _raw_spin_lock_irqsave+0x2d/0x4e  
[ 99.820087] print_address_description+0x6d/0x22f  
[ 99.820087] __kasan_report+0x138/0x17e  
[ 99.820087] SyS_exit_group+0x21/0x21  
[ 99.820087] trace_clock_x86_tsc+0x11/0x11  
[ 99.820087] ? prepare_exit_to_usermode+0x22a/0x236  
[ 99.820087] entry_SYSCALL_64_after_hwframe+0x3d/0xa2
```

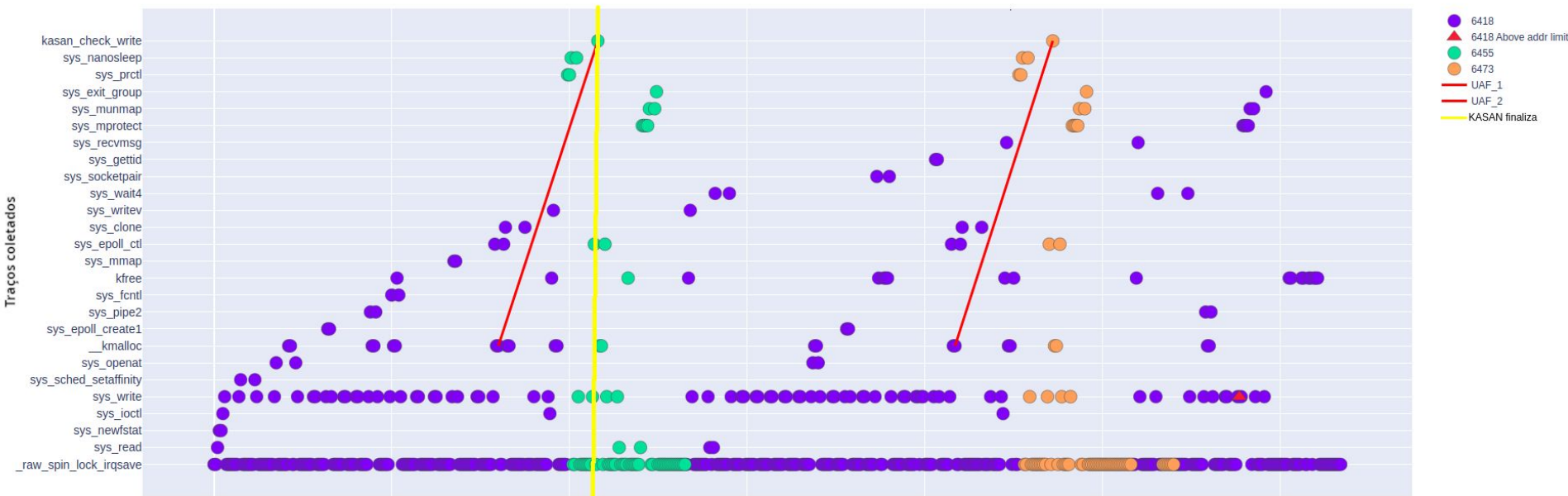
Solução proposta - Prova de Conceito

- Nossa solução

```
sys_openat;int dfd: 0000fffff9c;const char * filename: 000000480f0c;  
__kmalloc;size_t arg1: 000000000230;gfp_t arg2: 0000014080c0;;;;;  
ret__kmalloc;ffff888008038400;;;;;  
sys_epoll_create1;int flags: 000000000000;;;;;0  
sys_epoll_ctl;int epfd: 000000000004;int op: 000000000001;  
__kmalloc;size_t arg1: 000000000198;gfp_t arg2: 0000014080c0;;;;; 408 Bytes  
ret__kmalloc;ffff8880080c3e00;;;;;  
sys_ioctl;unsigned int fd: 000000000003;unsigned  
kfree;const void * arg1: ffff8880080c3e00;;;;; FREE!  
sys_mprotect;unsigned long start: 00007cb7c8b24000;  
sys_exit_group;int error_code: 000000000000;;;;;0  
_raw_spin_lock_irqsave;const void * arg1: ffff8880080c3e00;;;;; USE!  
kfree;const void * arg1: ffff88802a31cf00;;;;;
```

Solução proposta - Prova de Conceito

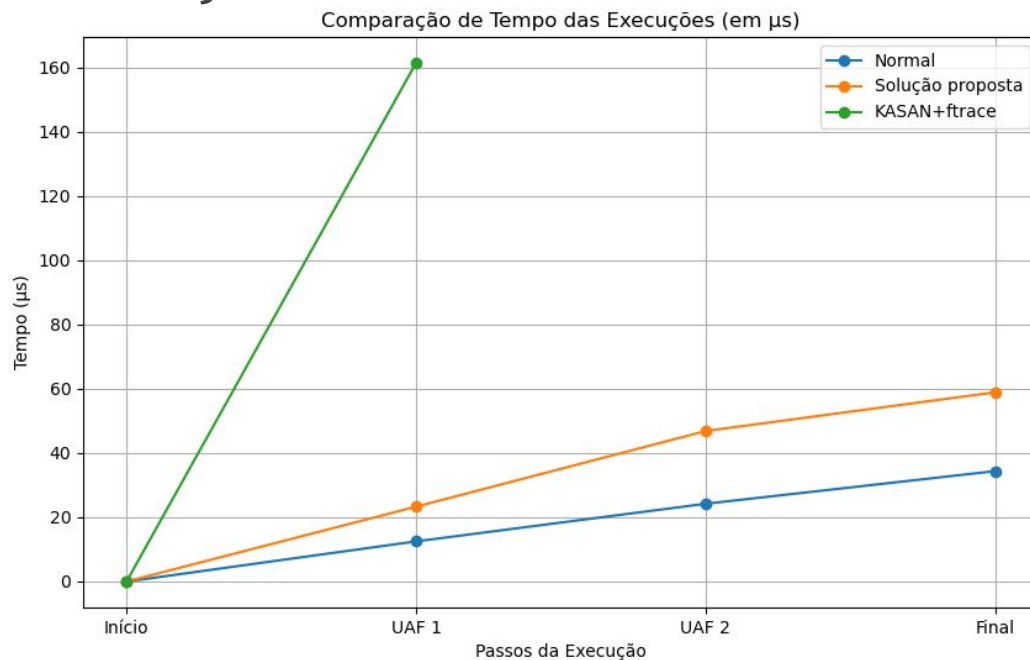
● Nossa solução



Visualização do traço coletado pela PoC em exploit da CVE-2019-2215.
Figura criada pelo autor utilizando Plotly.

Solução proposta - Prova de Conceito

- Nossa solução



Comparação de Tempo das Execuções (em μs) e Abrangência de Coleta do Traço do Exploit para a CVE-2019-2215.

Considerações finais

- Analisar e detectar erros em memória é custoso
- Disponibilidades de Exploits e configurações dos ambientes de teste são escassos

Trabalhos futuros

- Estudos de métodos mais eficientes
 - para quarentena e para atualização de regiões

Referências

[1] Mobile Operating System Market Share,

<https://gs.statcounter.com/os-market-share/mobile>

[2] Top 50 Products By Total Number Of "Distinct" Vulnerabilities,

<https://www.cvedetails.com/top-50-products.php>

[3] Android Statistics (2024),

<https://www.businessofapps.com/data/android-statistics/>

Referências

[4] iOS vs. Android: Which OS Is More Secure in 2022?,
<https://clario.co/blog/ios-vs-android-security/>

[5] Garg, S., & Baliyan, N. (2022). "Android Stack Vulnerabilities: Security Analysis of a Decade." In: Proceedings of the International Conference on Paradigms of Communication, Computing and Data Sciences, Algorithms for Intelligent Systems. Springer, Singapore.
DOI: 10.1007/978-981-16-5747-4_10.

Referências

[6] Android Memory -

<https://source.android.com/docs/security/test/memory-safety>

[7] Google Security Blog -

<https://security.googleblog.com/2021/01/data-driven-security-hardening-in.html>

[8] KASAN -

<https://naveenaidu.dev/kasan-kernel-address-sanitizer>

Obrigado!

ctjunior@inf.ufpr.br

claudiotorresjunior@gmail.com

Este trabalho teve apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES)

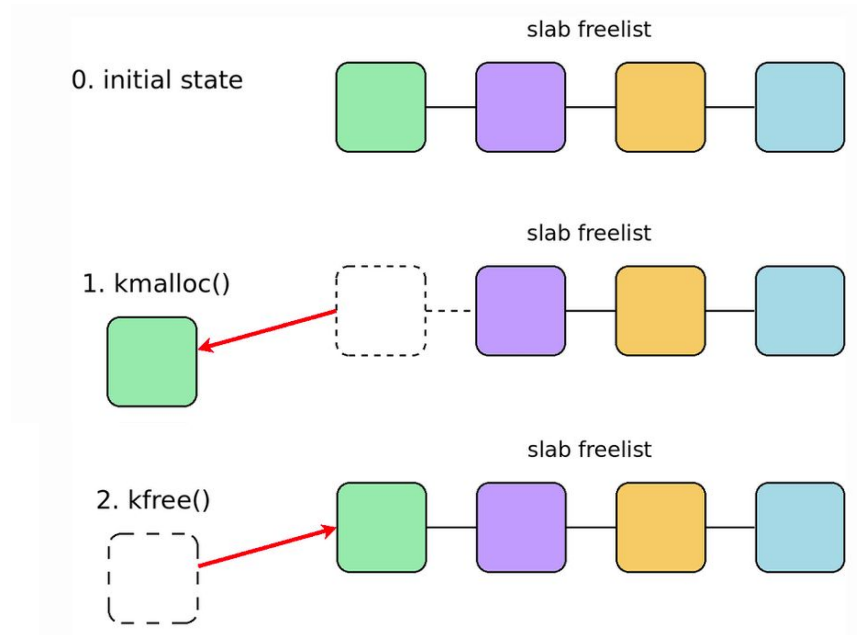


Solução proposta - Prova de Conceito

- Traços gerados
 - Syscalls/funções
 - "timestamp", "pid", "event", "arg0", "arg1", "arg2", "arg3", "arg4", "arg5", "above_addr_limit"
 - Stacktrace
 - "timestamp", "pid", "event", "stacktrace"
 - Acessos indevidos na memória
 - "timestamp", "pid", "event"

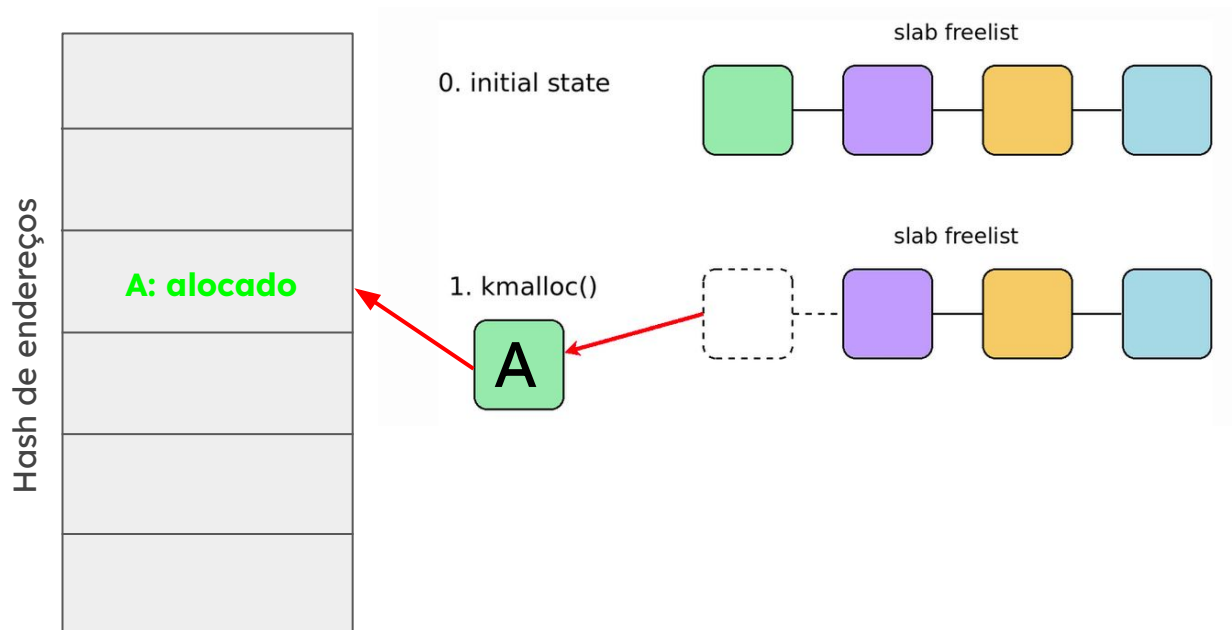
Solução proposta - Prova de Conceito

- Alocação e desalocação base



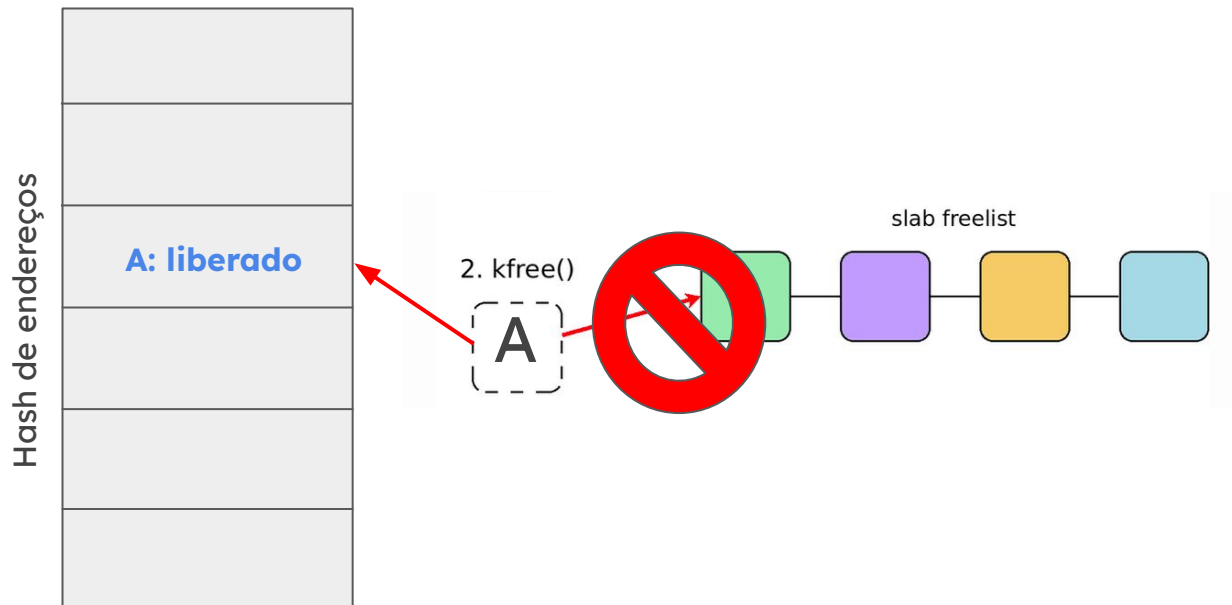
Solução proposta - Prova de Conceito

- Nossa solução para quarentena



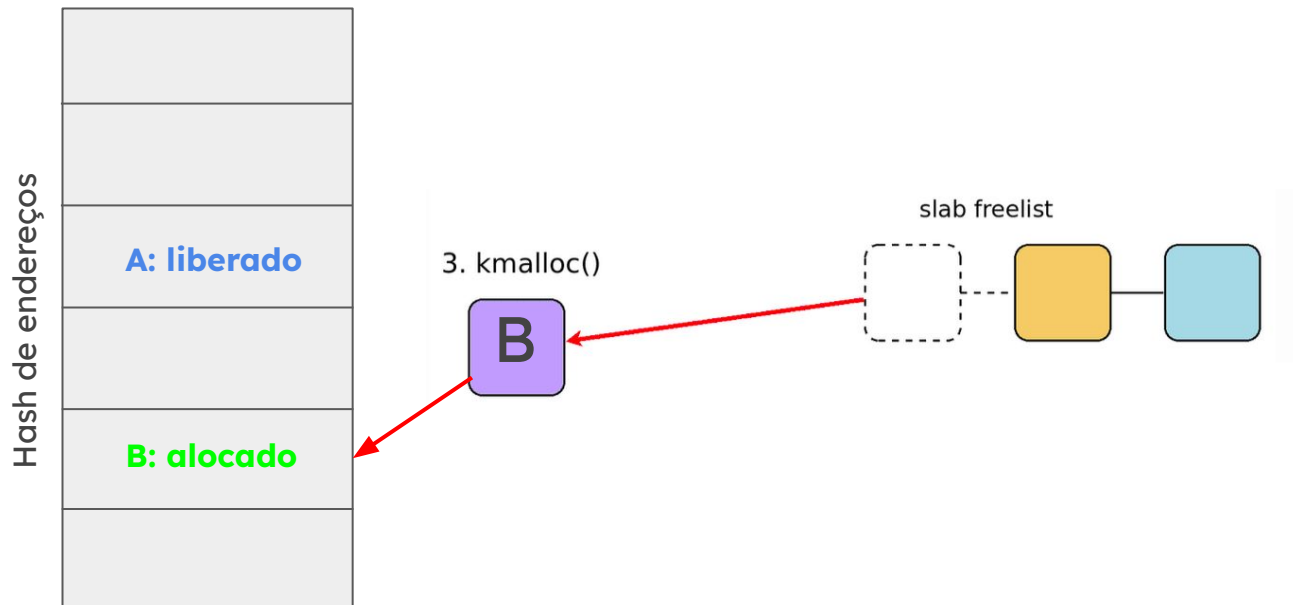
Solução proposta - Prova de Conceito

- Nossa solução para quarentena



Solução proposta - Prova de Conceito

- Nossa solução para quarentena



Tempo para strace

Tabela 1. Tempo em microssegundos e desvio-padrão da execução do *strace* com KASAN desabilitado e habilitado para as *syscalls* selecionadas.

Syscall	Sem KASAN	Com KASAN
open/close	155.911 ± 8.195	228.114 ± 23.174
write	87.761 ± 1.897	119.051 ± 24.875
read	85.254 ± 0.562	118.340 ± 13.888
stat	90.714 ± 2.182	140.018 ± 2.144
fstat	88.851 ± 0.815	118.262 ± 28.930
fork	322.087 ± 1.818	531.523 ± 90.882
mmap/munmap	171.113 ± 1.583	158.549 ± 54.768