S.H.I.E.L.D
Scudo Heap Implementation Exploits, Leaks, and Defenses

—

Tom Mansion

OFF-BY-ONE
2025

Quarkslab

Who I am

Tom Mansion

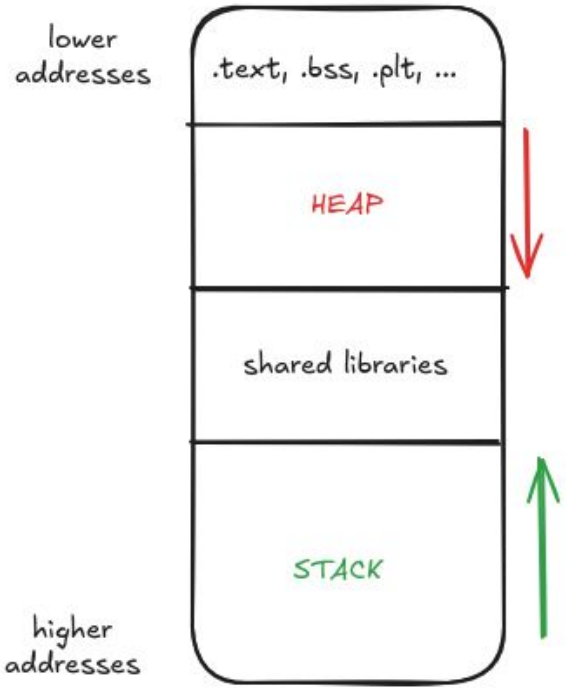
- R&D Engineer @Quarkslab
- CTF Enjoyer
- Heap exploitation enthusiast
- Focused on Android

Presentation outline

1. **Introduction to heap allocators**
   a. GLIBC's ptmalloc2 refresher
   b. How can we exploit it?
2. **Introduction to Scudo**
   a. Scudo security principles
   b. Scudo building blocks
3. **Scudo exploitation techniques**
   a. Use-after-frees and the quarantine
   b. Zygote to the rescue!
   c. Forged Commitbase
   d. Safe Unlink
4. **Conclusion**

Introduction - what even is a heap allocator?

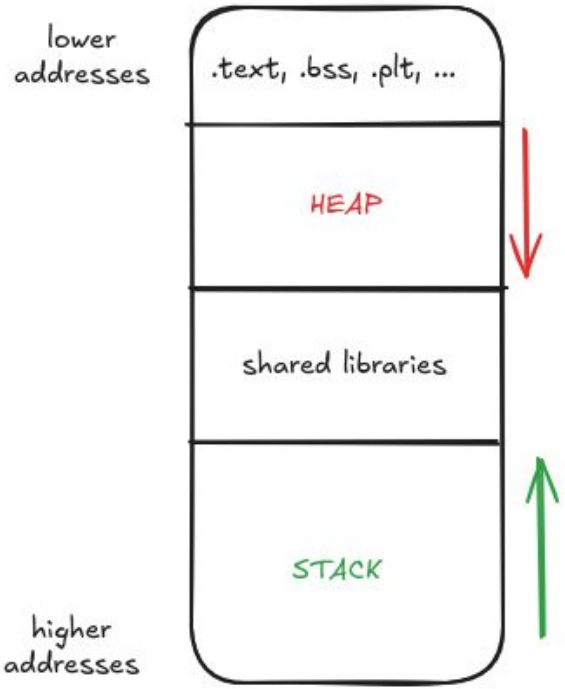
- Dynamic memory management runtime
- Lives in libc
- Used by almost every program
- Called very often



Introduction - what even is a heap allocator?

Properties of a good allocator:

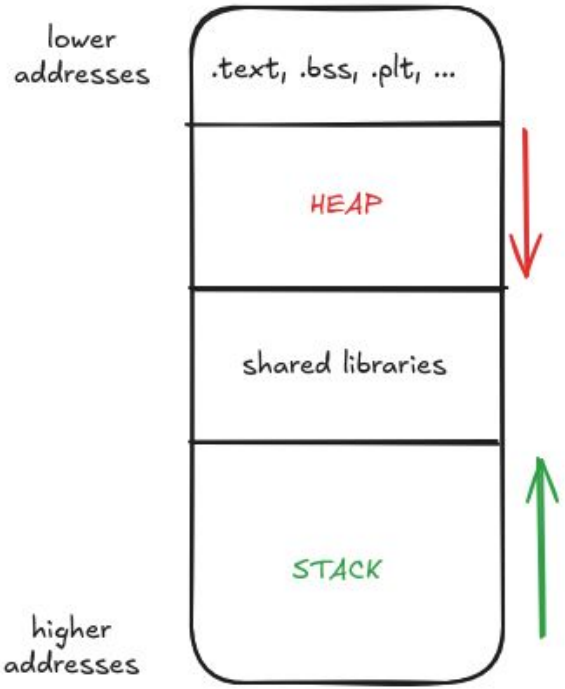
- Transparent



Introduction - what even is a heap allocator?

Properties of a good allocator:

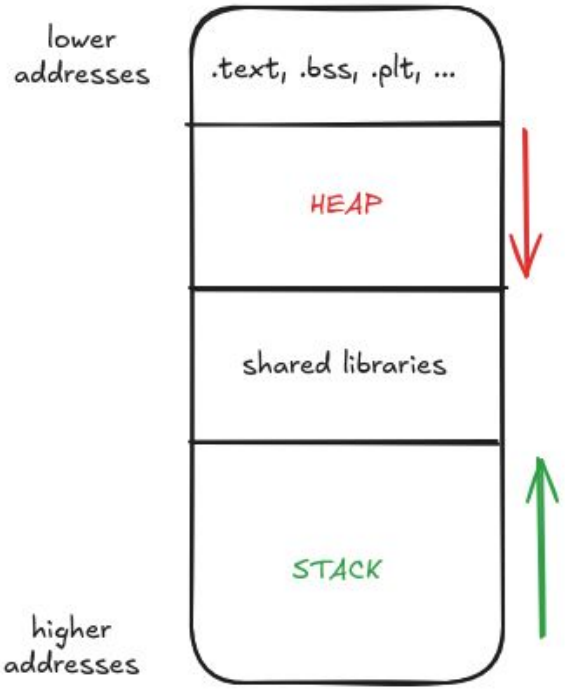
- Transparent
- Reliable



Introduction - what even is a heap allocator?

Properties of a good allocator:

- Transparent
- Reliable
- Efficient

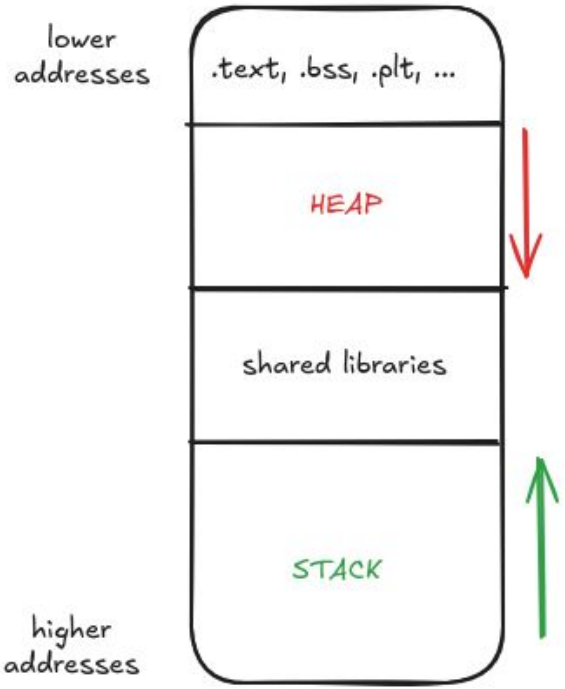



95
95
PISTON CUP

Introduction - what even is a heap allocator?

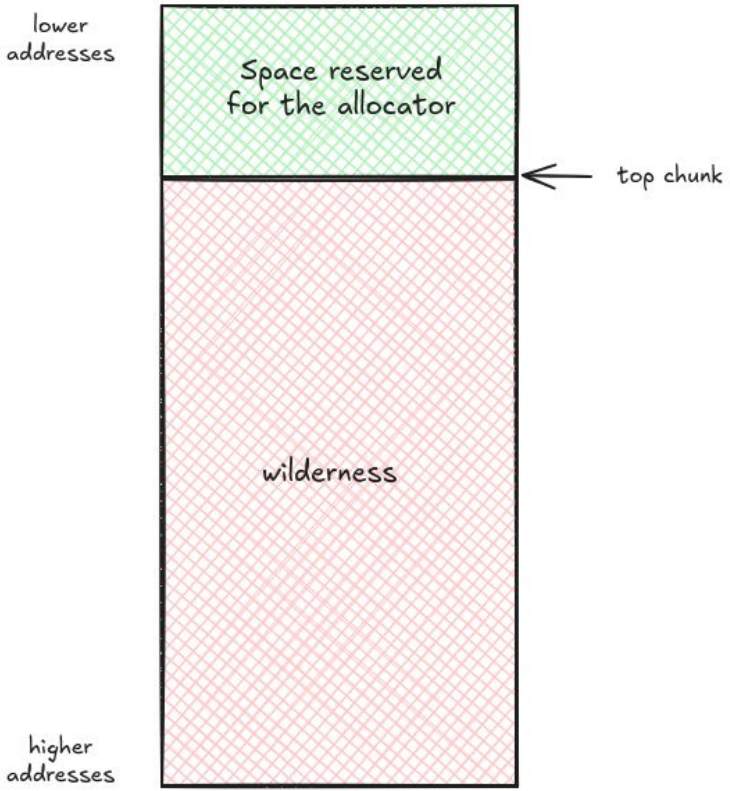
Properties of a good allocator:

- Transparent
- Reliable
- Efficient
- **Fast**
- (Secure)



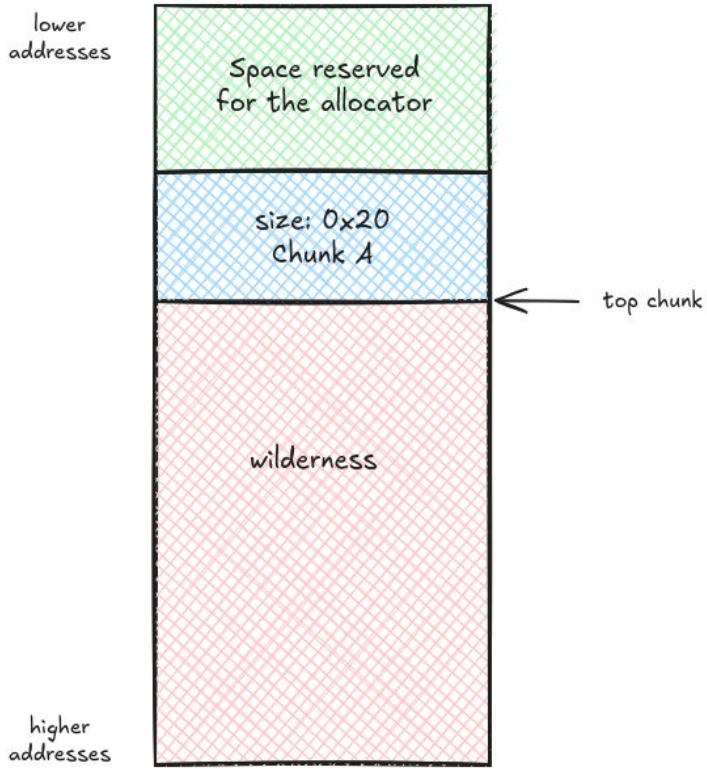
GLIBC's ptmalloc2 (refresher)

Properties of ptmalloc2



GLIBC's ptmalloc2 (refresher)

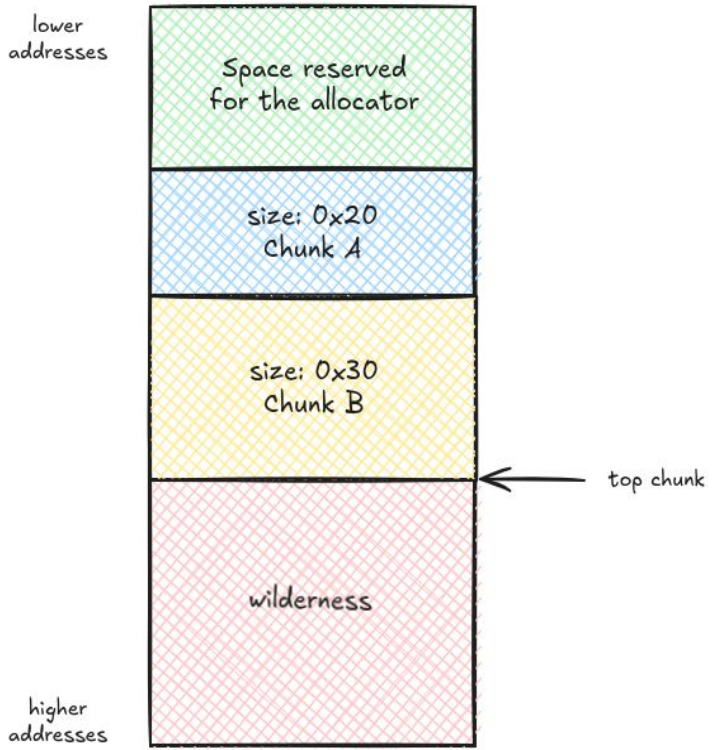
Properties of ptmalloc2



GLIBC's ptmalloc2 (refresher)

Properties of ptmalloc2

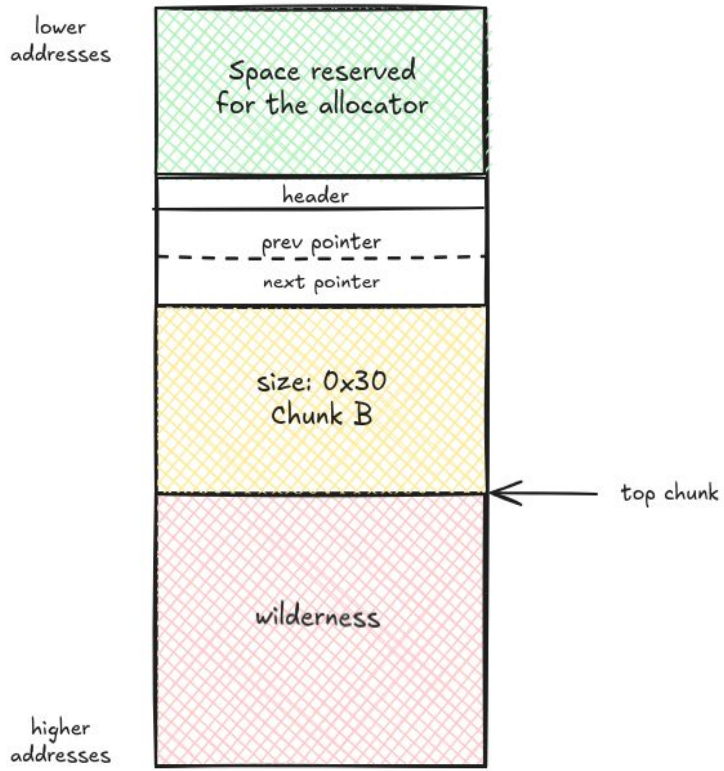
- Chunks of different sizes can be adjacent
- Deterministic



GLIBC's ptmalloc2 (refresher)

Properties of ptmalloc2

- Chunks of different sizes can be adjacent
- Deterministic
- Free chunks can be stored in freelists
- Freelist metadata is stored on the heap



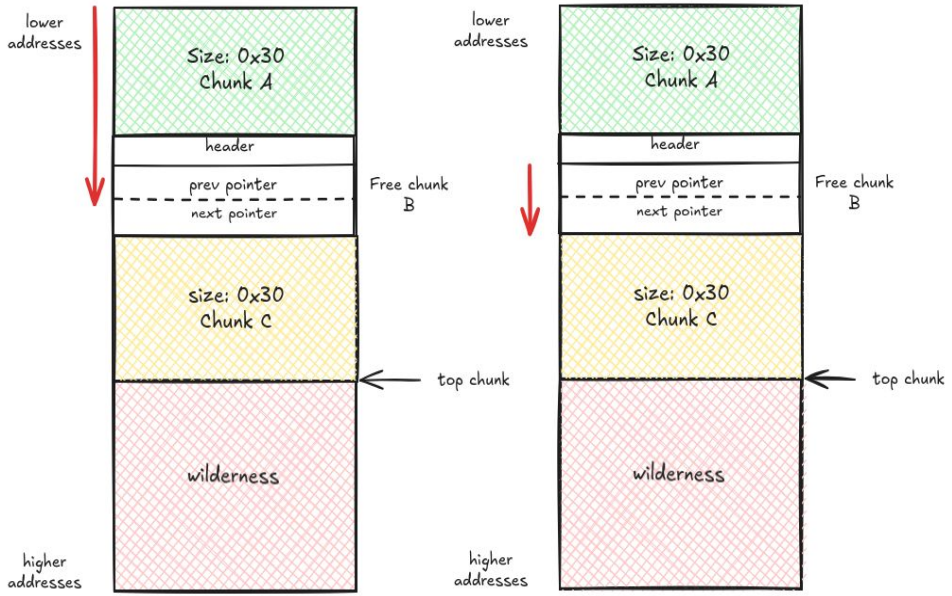
GLIBC’s ptmalloc2 - how can we exploit it?

Different classes of vulnerabilities (not exhaustive)

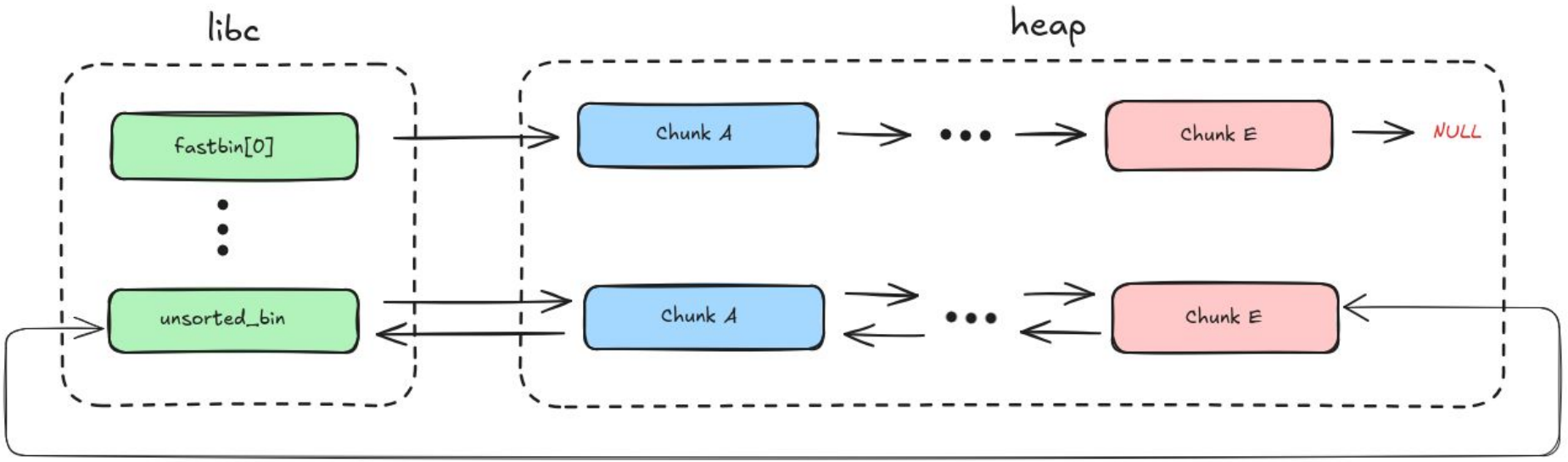
- Buffer overflow
- Use-After-Free
- Double-Free

What can we gain control over?

➔ User-controlled data
➔ Allocator control metadata

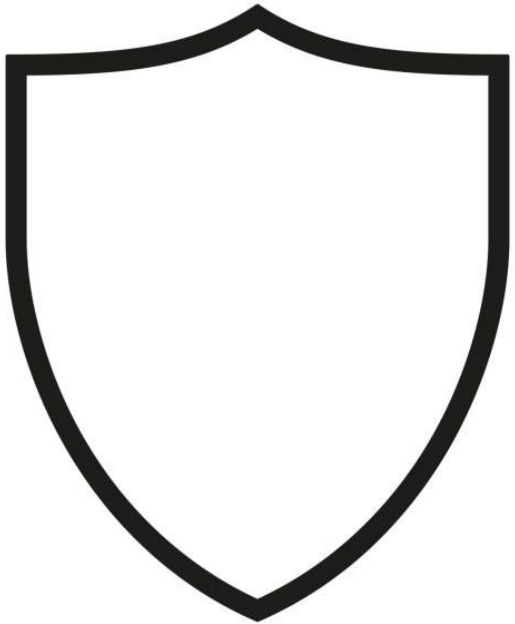


GLIBC’s ptmalloc2 - what metadata do chunks hold?



Introduction to Scudo

- Default allocator on Android
- **Hardened**
- Configurable
- Part of the LLVM project



Introduction to Scudo - Security principles

- **Separating allocation classes**
- **Introducing randomness wherever possible**
  - Randomizing the order of allocated chunks
- **Protecting the chunk headers with a checksum**
  - Prevents small buffer overflows
  - Based on a global secret stored in libc
- **Removing control metadata from the heap**
  - Prevents corruption of freelist pointers

Scudo building blocks - Primary and Secondary allocators

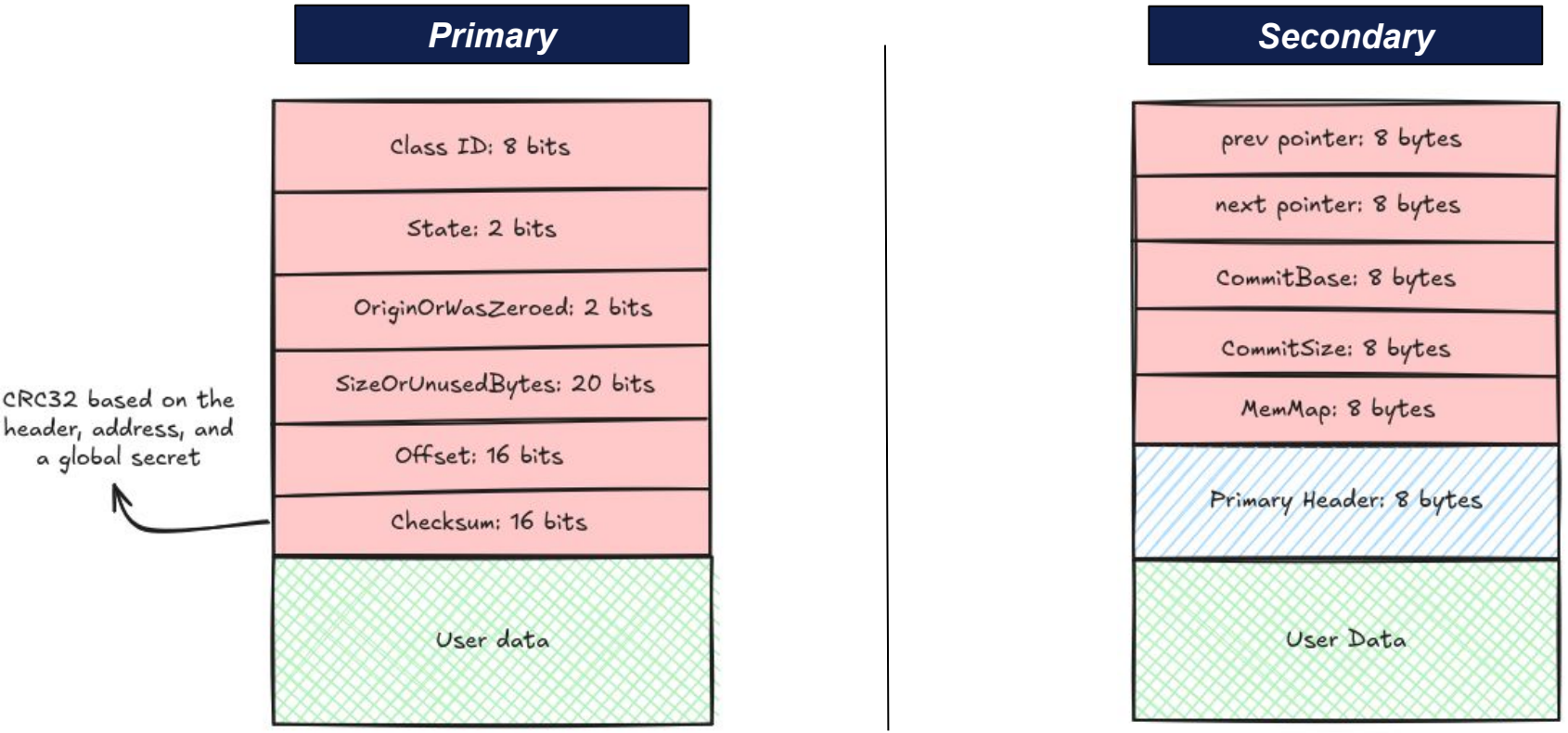
Primary

- **Small** allocations (< 64kB)
- One Region per chunk size
- Local cache used for **everything**
- Creates chunks with a primary header
- (Has a quarantine cache)

Secondary

- **Large** allocations (> 64kB)
- Creates chunks with mmap()
- Local cache for free chunks
- Creates chunks with a primary+secondary header
- (Has a quarantine cache)

Scudo building blocks - Chunk Headers



Scudo building blocks - Regions & Guard Pages

Scudo separates chunks by their size

- Mmaps a bunch of pages for the Primary
- Divides it into 32 “Regions”
- Random amount of 1-16 guard pages for each Region

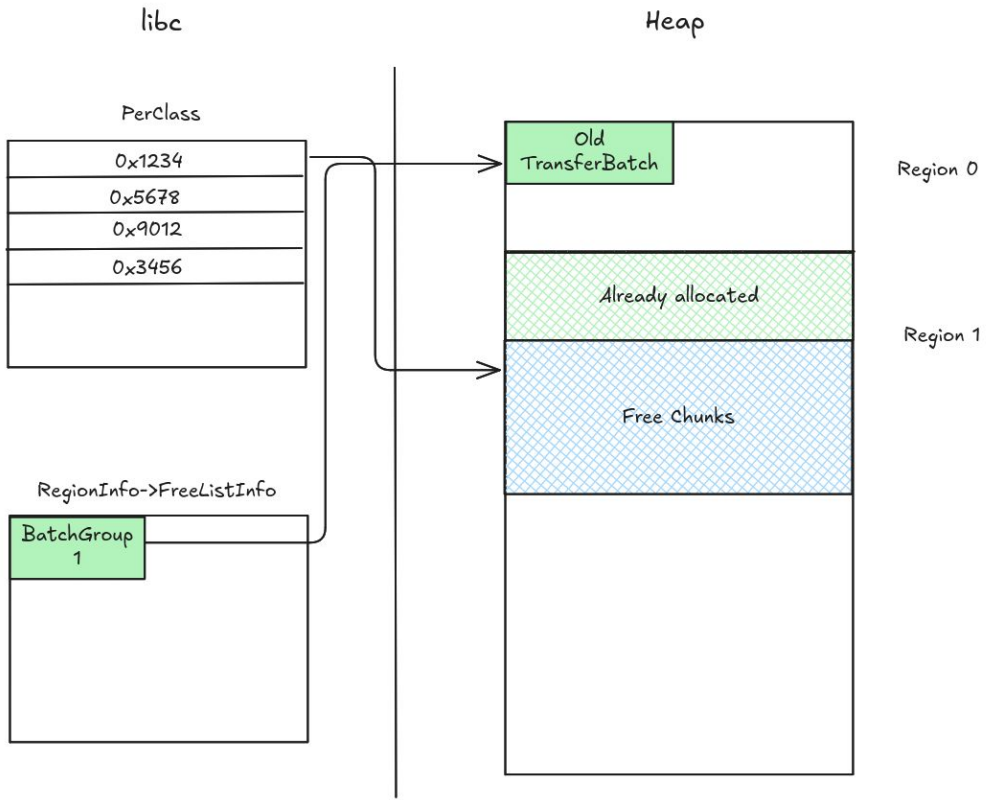
Scudo building blocks - Local cache

- Free list implementation
- One per Region (PerClass)
- Fully stored in libc
- Drain/Fill operations use TransferBatches

```
struct alignas(SCUDO_CACHE_LINE_SIZE) PerClass {
  u16 Count;
  u16 MaxCount;
  // Note: ClassSize is zero for the transfer batch.
  uptr ClassSize;
  CompactPtrT Chunks[2 * SizeClassMap::MaxNumCachedHint];
};
```

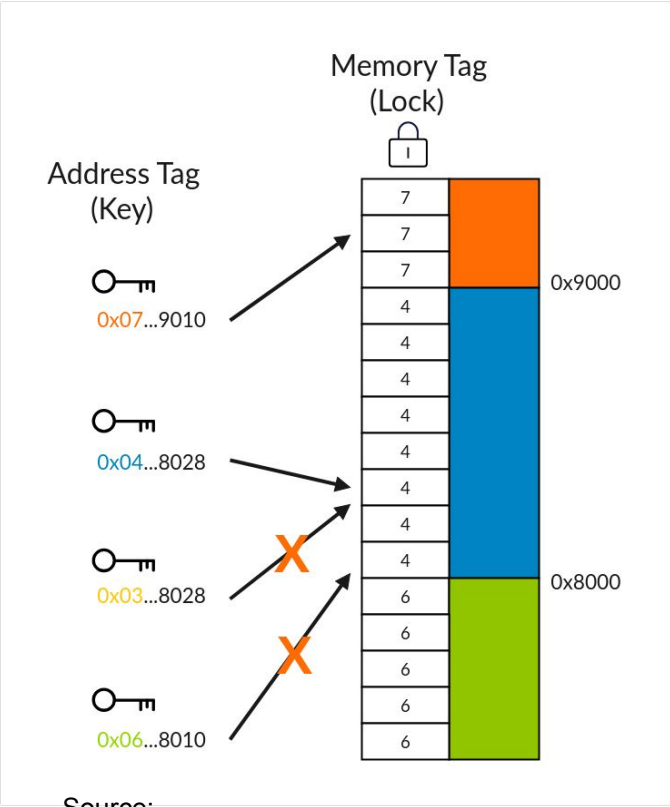
Scudo building blocks - Transfer Batches

- Part of a region's second freelist
- Cache layer between Region and local cache



Scudo building blocks - ARM MTE support

- Scudo developed to support it
- Different tag for header and contents
- Different tag for Primary/Secondary header
- Memory is re-tagged upon deallocation



Source:
https://developer.arm.com/documentation/108035/0100/How-does-MTE-work-?lang=en

And now, the moment you've all been waiting for

The case of Use-After-Frees

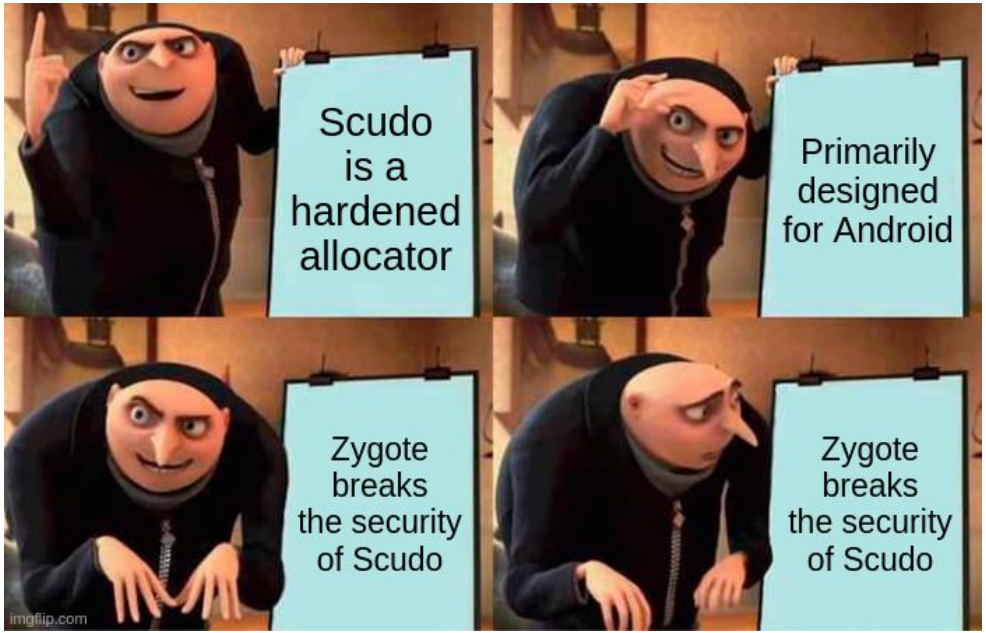
- Without the Quarantine and MTE, Scudo doesn't mitigate UAFs!
- No inline metadata to corrupt
- Type confusion exploits are the new target
- Example from a basic CTF challenge

```
struct Dog {
    char name[12];
    void (*bark)();
    void (*bringBackTheFlag)();
    void (*death)(struct Dog*);
};

struct DogHouse{
    char address[16];
    char name[8];
};
```

Scudo Exploitation

- Forged Commitbase
- Safe Unlink
- Zygote to the rescue!
- Patches 🥲



Forged Commitbase & Safe Unlink

Techniques published by P. Mao et al, in Usenix WOOT 24’

[Exploiting Android's Hardened Memory Allocator]

Exploiting Android’s Hardened Memory Allocator

Philipp Mao Elias Valentin Boschung Marcel Busch Mathias Payer
EPFL, Lausanne, Switzerland

Abstract

Most memory corruptions occur on the heap. To harden userspace applications and prevent heap-based exploitation, Google has developed Scudo. Since Android 11, Scudo has replaced jemalloc as the default heap implementation for all native code on Android. Scudo mitigates exploitation attempts of common heap vulnerabilities.

tacks, Scudo implements security measures to ensure the integrity of inline heap metadata and to prevent a predictable heap layout.

Exploitation techniques that target the allocator to escalate a heap-bound memory corruption vulnerability into an arbitrary memory write primitive or code execution have a long tradition. The security community has compiled a large

Forged Commitbase & Safe Unlink: Zygote to the rescue!

Both techniques share heavy prerequisites

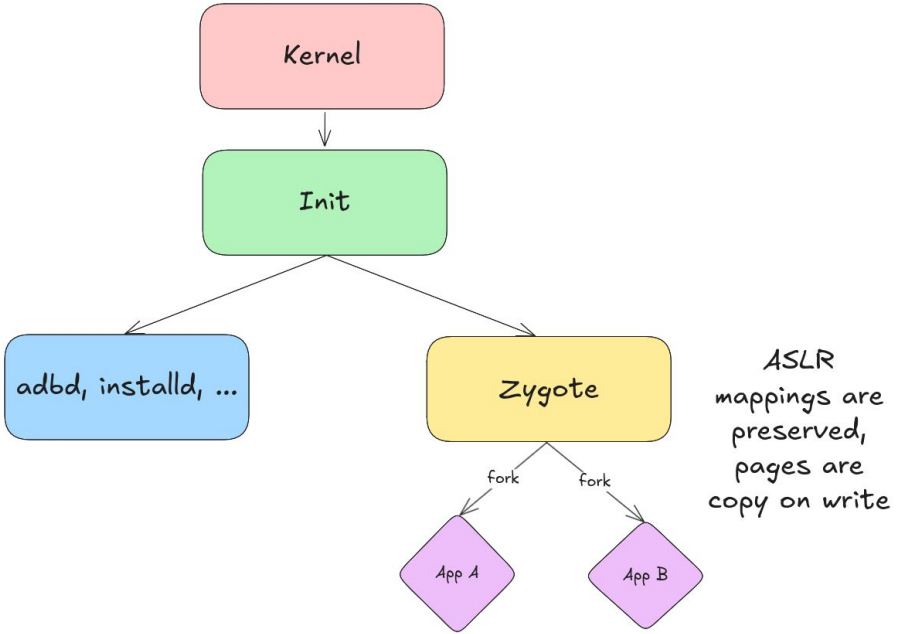
- Ability to guess chunks' addresses
  - Protected by the shuffling of chunks in TransferBatches
- Ability to forge a checksum
  - Protected by the security cookie stored in libc

But do these protections <u>always</u> work?

Forged Commitbase & Safe Unlink: Zygote to the rescue!

What is Zygote?

- Used to speed up app start time
- Loads framework code/resources
- Forked to spawn app processes
  - All user-installed apps
  - Some privileged services

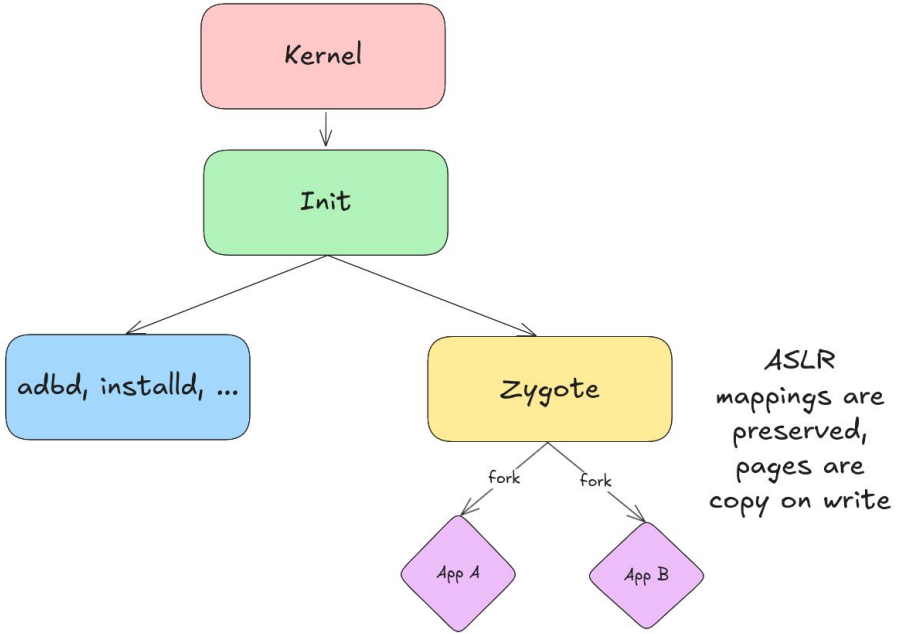


Forged Commitbase & Safe Unlink: Zygote to the rescue!

Great! What's the problem then?

- Zygote forks share their memory layout…
- Memory pages are copy on write…
- Zygote uses the heap

Scudo is initialized by Zygote



Forged Commitbase & Safe Unlink: Zygote to the rescue!

Both techniques share heavy prerequisites

- Ability to guess chunks' addresses
  - Protected by the shuffling of chunks in TransferBatches

⇒ Random seed is initialized by Zygote!

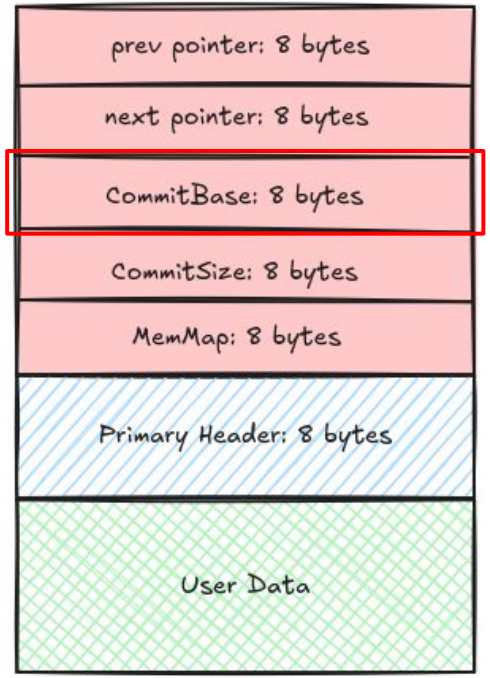
- Ability to forge a checksum
  - Protected by the security cookie stored in libc

⇒ Security cookie is initialized by Zygote!

Forged CommitBase

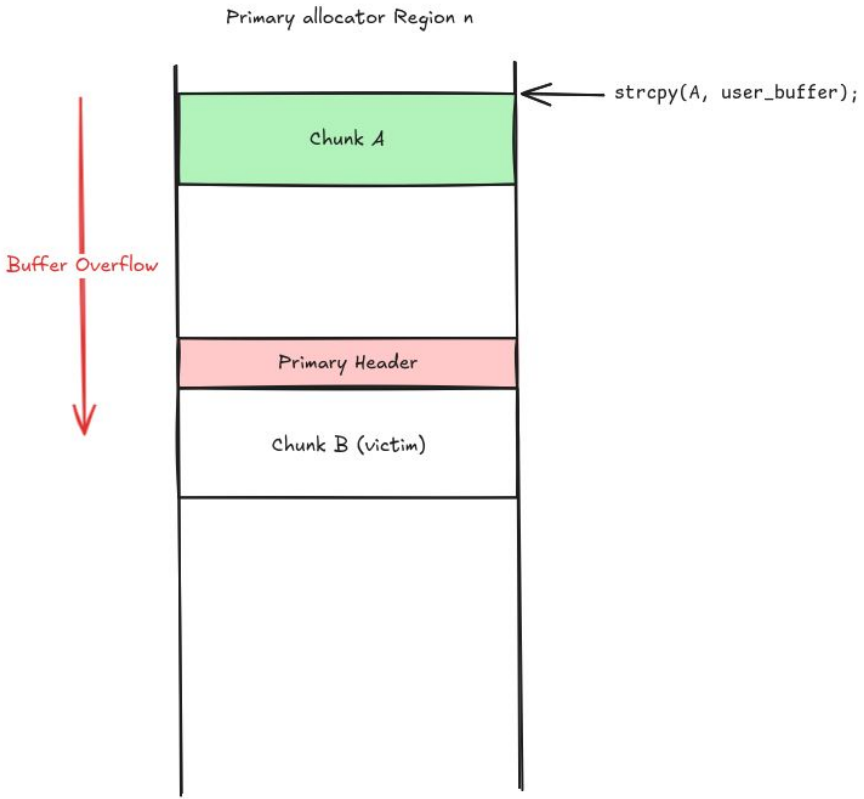
Force the **Secondary** allocator to return **one** arbitrary chunk

- By forging a fake secondary header
- A few constraints:
  - Predict a chunk's address
  - Forge a checksum
  - Write 0x28 bytes before a chunk
  - Free the victim chunk
  - Have a secondary chunk already allocated



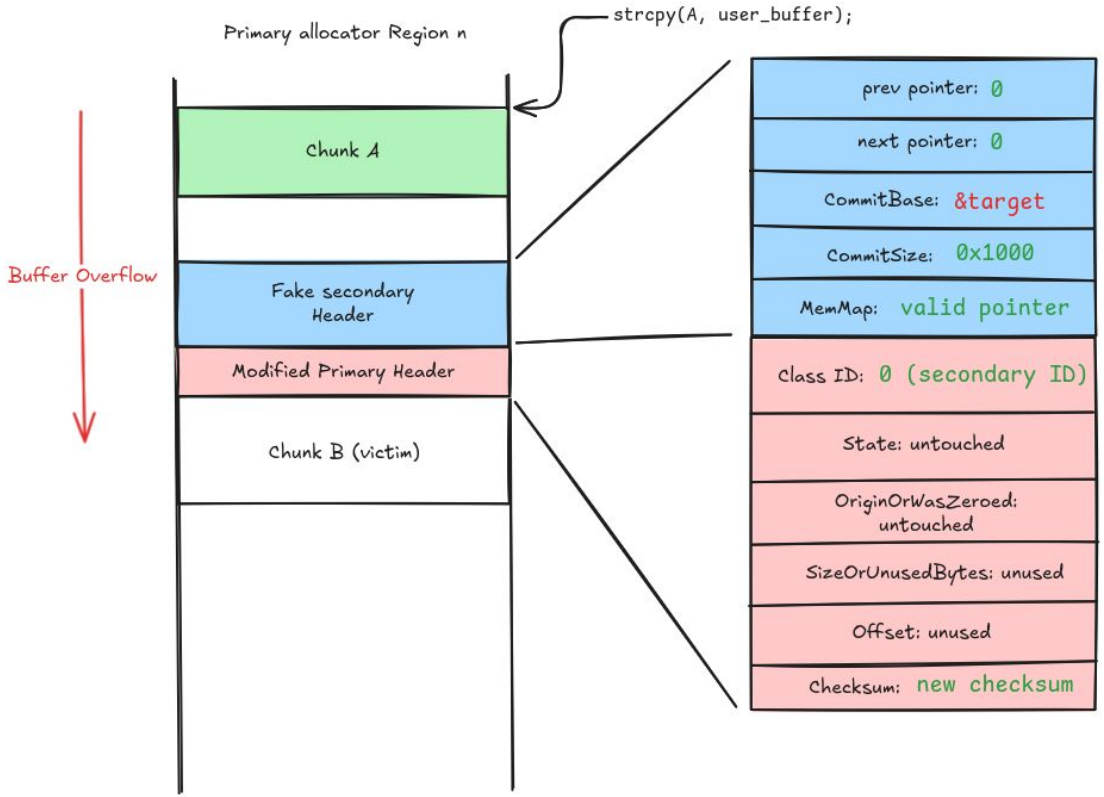
Forged CommitBase (cont.)

1. Buffer overflow from Chunk A to Chunk B



Forged CommitBase (cont.)

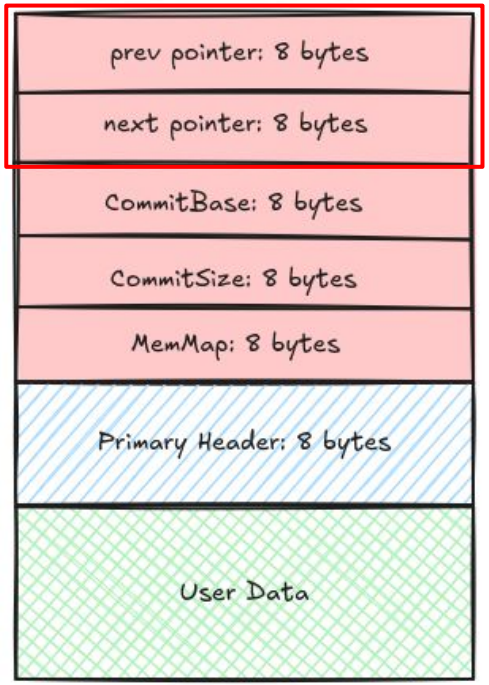
1. Buffer overflow
2. Forge fake secondary header
3. Modify Primary header
4. Free Chunk B
5. &target is placed in secondary cache
6. Allocate new secondary chunk & profit



Safe Unlink

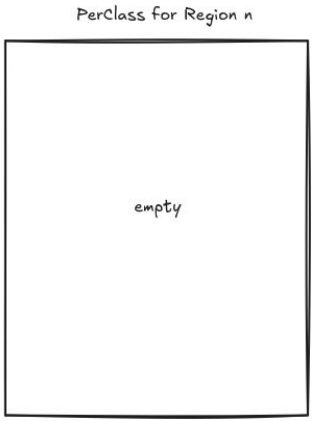
Force the allocator to return a chunk overlapping the Local Cache

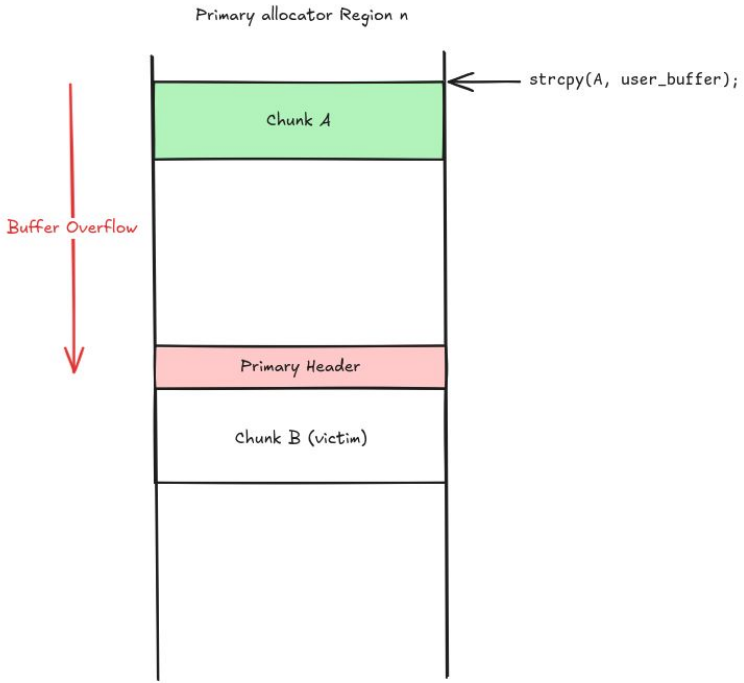
- Inspired by the Unsafe Unlink technique on GLIBC
- Unlink from the list of in-use chunks
- Once exploited, allows for **unlimited** arbitrary chunks
- Has a few more constraints:
  - Predict a chunk's address
  - Forge a checksum
  - Write 0x30 bytes before a chunk (buffer overflow)
  - Controlled free primitive that we can call 3 times
  - Free the victim chunk



Safe Unlink (cont.)

We start with a buffer overflow

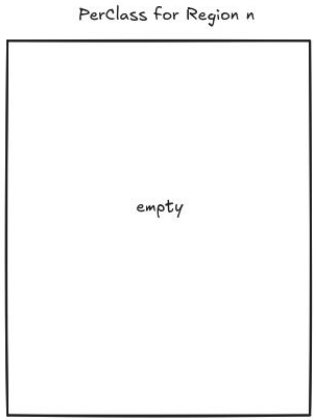


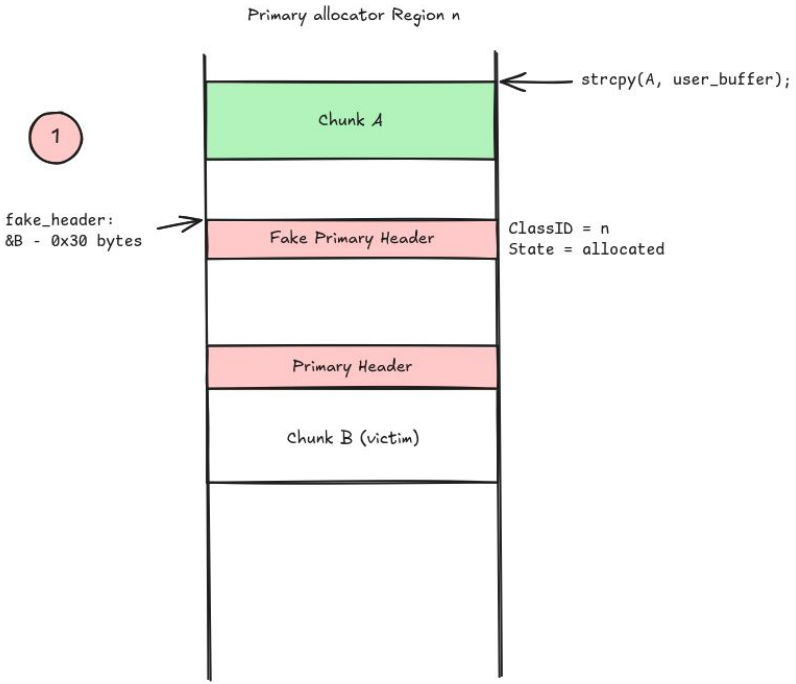


Safe Unlink (cont.)

1

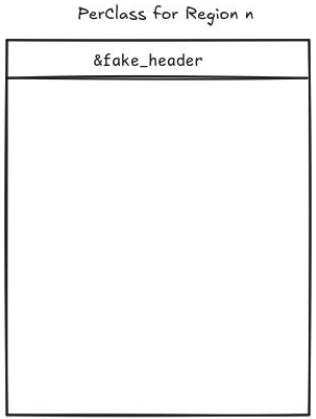
Forge fake primary header
0x30 before victim chunk

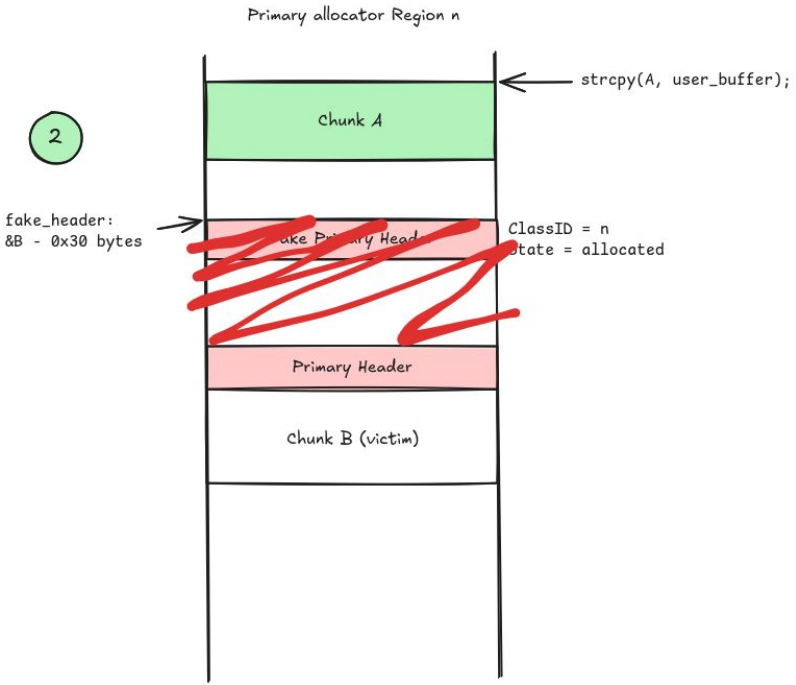




Safe Unlink (cont.)

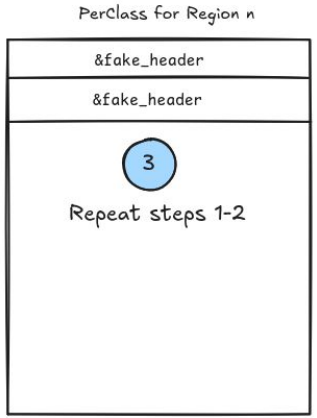
2 **Free the fake chunk**
Controlled free primitive

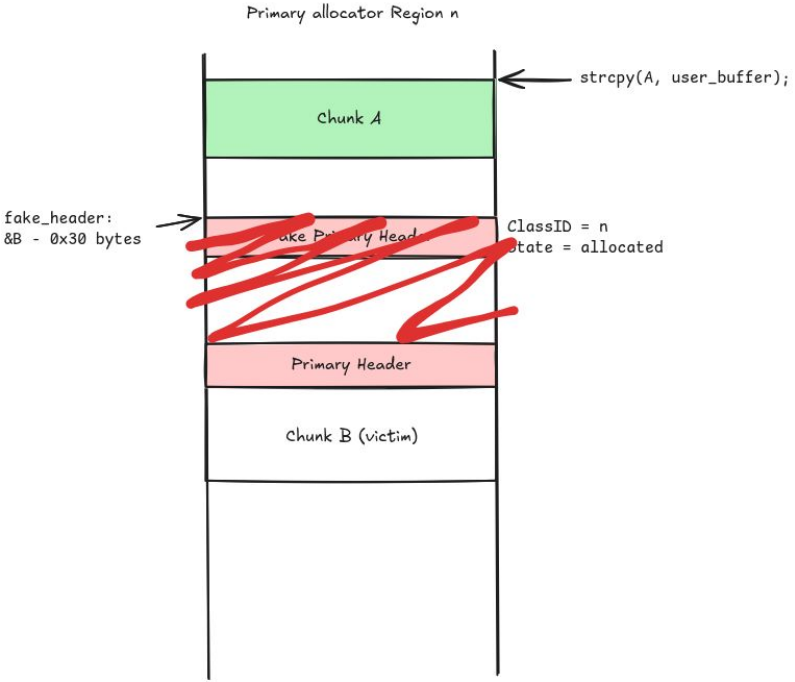




Safe Unlink (cont.)

3 **Do this twice**



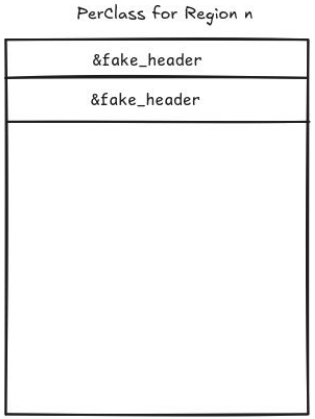


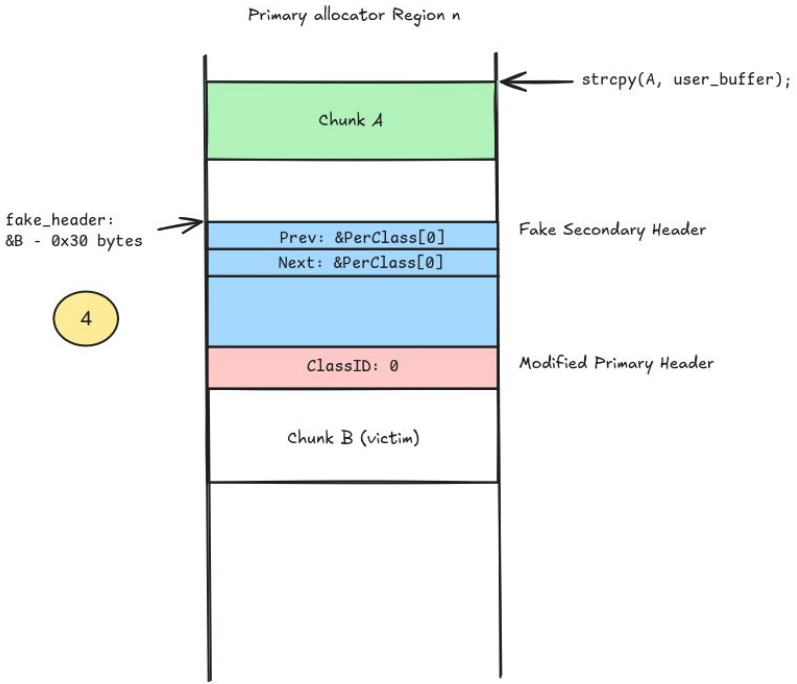
Safe Unlink (cont.)

4

Forge fake secondary header

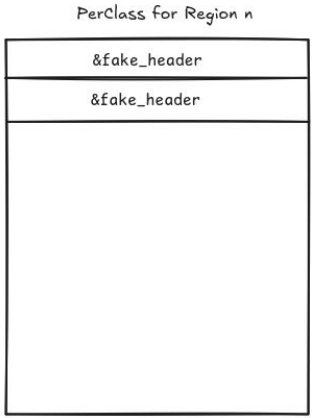
Setting up the unlinking attack

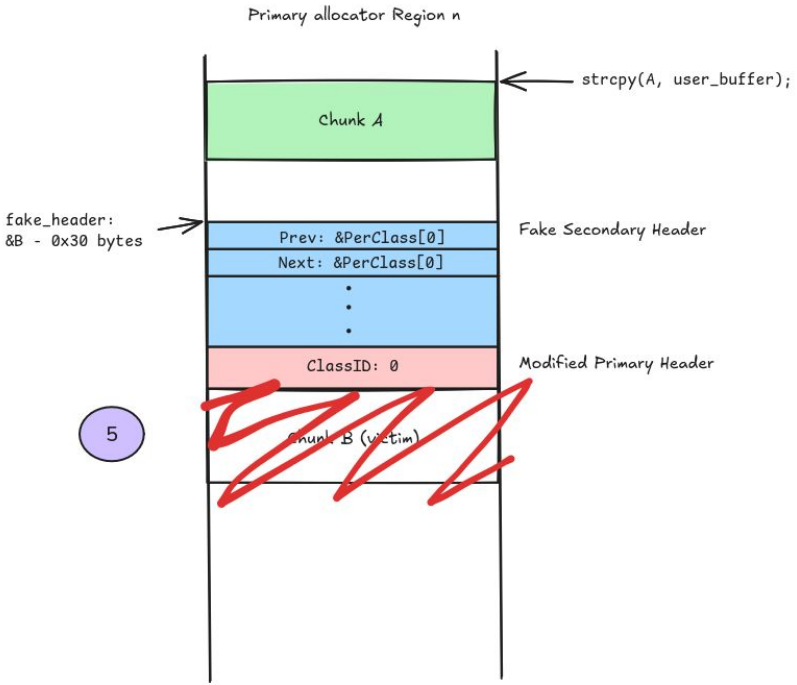




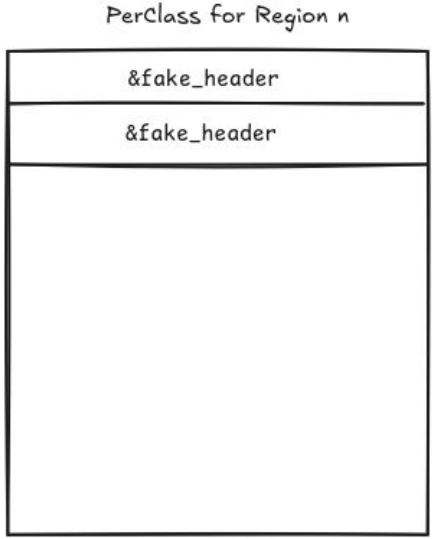
Safe Unlink (cont.)

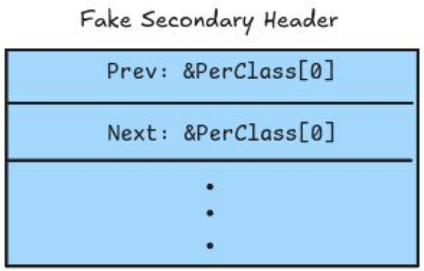
5 **Free the victim chunk**
Trigger the unlinking





Safe Unlink (cont.)






```c
typedef struct Node {
    struct Node* prev;
    struct Node* next;
} node_t;

void unlink(node_t* x) {
    node_t* prev = x->prev;
    node_t* next = x->next;

    prev->next = next;
    next->prev = prev;
}
```

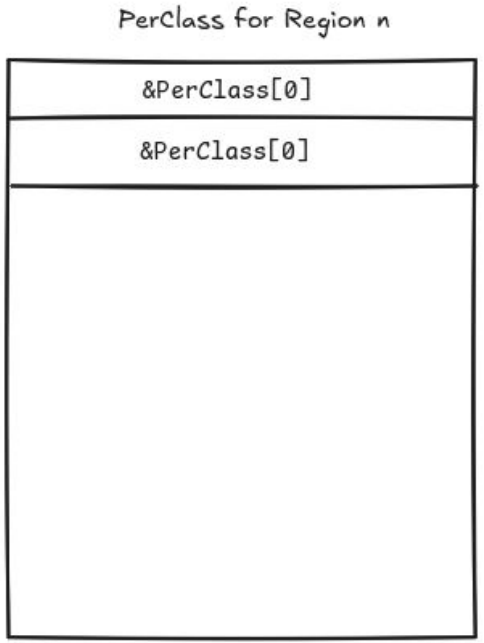
Safe Unlink (cont.)

6

Profit!

- The next allocation will return a pointer to PerClass
- We now have control over all subsequent allocations

6



Patches

- Safe Unlink was patched with Compact Pointers in the Local Cache
- CompactPtrs are offsets from the beginning of a Region
- This breaks step 5 of the exploit

```
struct alignas(SCUDO_CACHE_LINE_SIZE) PerClass {
  u16 Count;
  u16 MaxCount;
  // Note: ClassSize is zero for the transfer batch.
  uptr ClassSize;
  CompactPtrT Chunks[2 * SizeClassMap::MaxNumCachedHint];
};
```

Perspective - Combining the techniques?

What if we skip the unlinking step?

Assume we can store arbitrary CompactPtrs in the local cache

- Maybe with Forged CommitBase
- Or another exploit

```
static uptr decompactPtrInternal(uptr Base, CompactPtrT CompactPtr) {
  return Base + (static_cast<uptr>(CompactPtr) << CompactPtrScale);
}
```

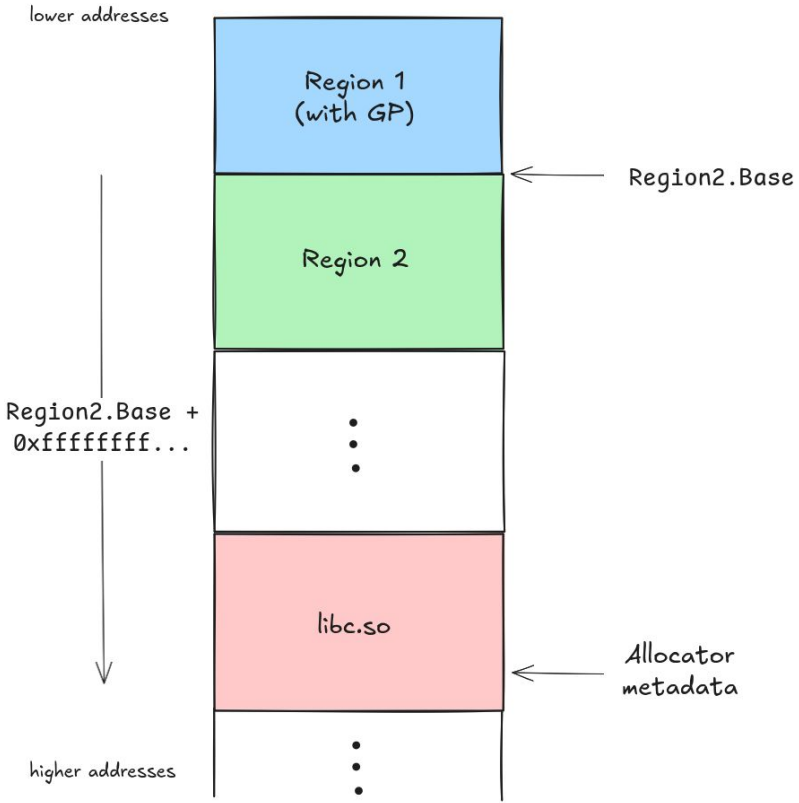
If we can control this too, we get the same primitive as Safe Unlink!

Perspective - Combining the techniques?

What if we skip the unlinking step?

We have control over the Local Cache for Region 2

1. Compute distance between Region2 and RegionInfo metadata
2. Write it in Local Cache
3. Allocate chunk over RegionInfo
4. We now control Region2.Base
5. Profit from arbitrary allocations



Conclusion

- **On Android, Zygote-forked processes can attack each other easily**
  - Forged Commitbase is still unpatched!
  - Safe Unlink has been patched :(
- Combining the techniques can yield interesting results
- In other contexts, Scudo is still very secure!
- There is not much left to exploit (very little inline metadata)
- ARM MTE will mitigate **all** of the exploits we've talked about
  - And many other classes of vulnerabilities
  - It is especially efficient for heap-based memory corruption

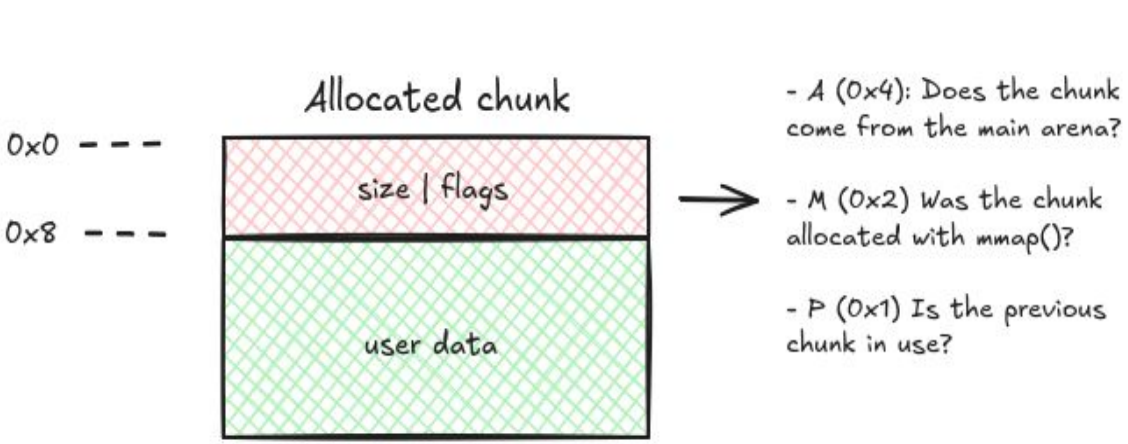
Thank you!

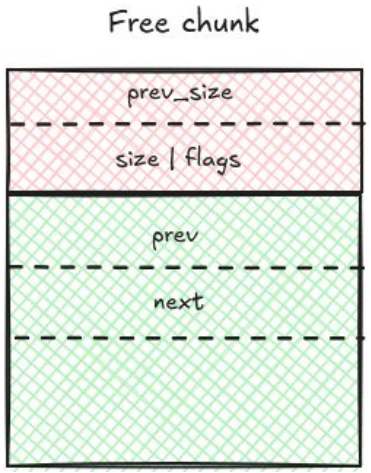
contact@quarkslab.com

@philipp0x90

Quarkslab

GLIBC's ptmalloc2 - what metadata do chunks hold?



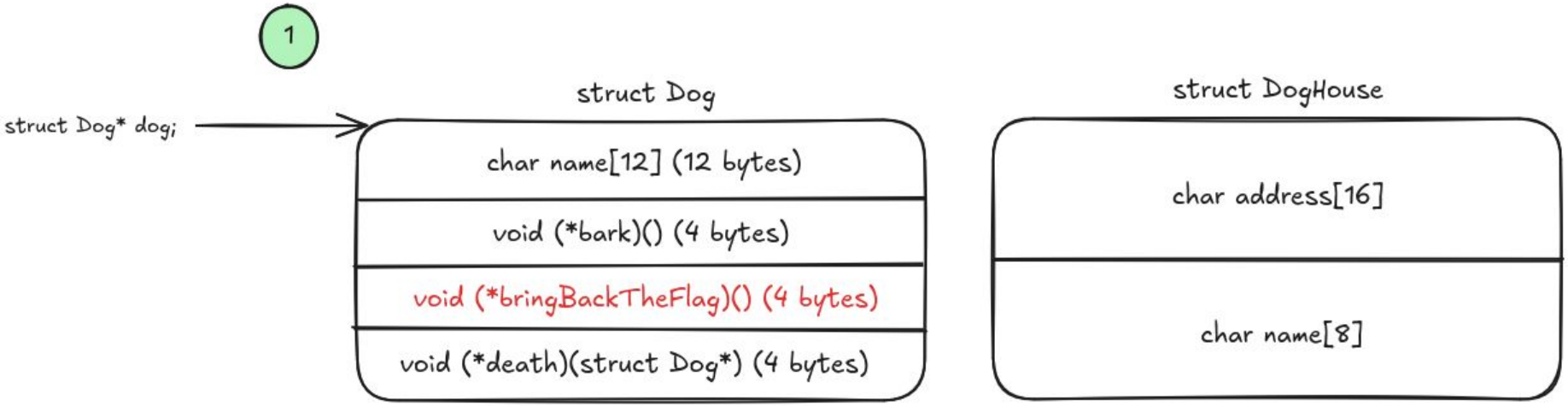


Targets for exploitation:

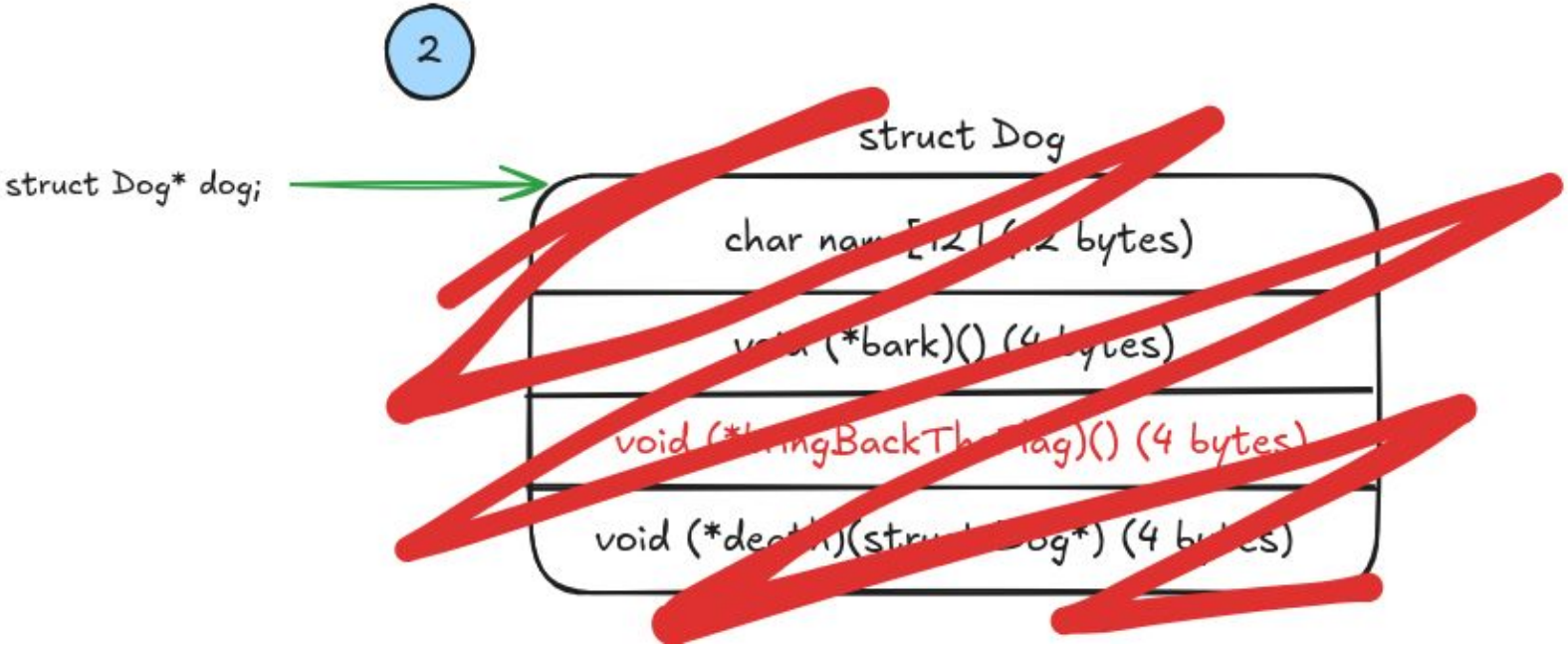
- Header
- User data

- Header
- Freelist pointers

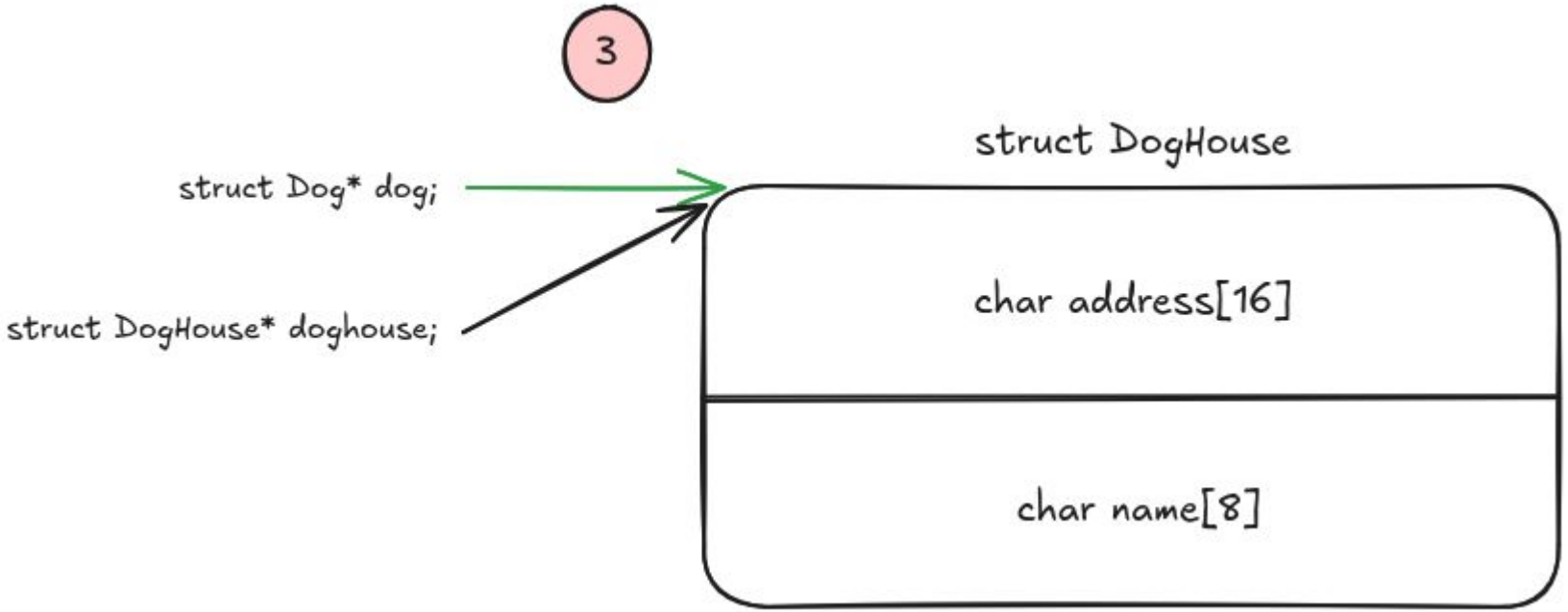
Use-after-free type confusion



Use-after-free type confusion



Use-after-free type confusion



Use-after-free type confusion