Exploiting Android's hardened Memory Allocator

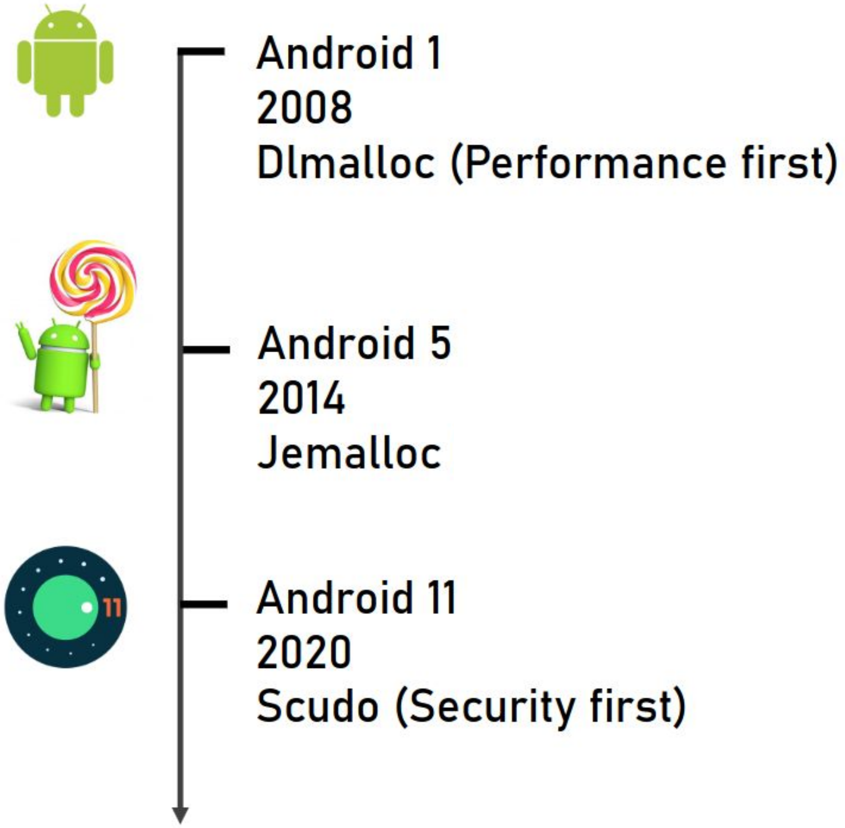
Philipp Mao, Elias Valentin Boschung, Marcel Busch, Mathias Payer
EPFL

Scudo: the Hardened Memory Allocator

Scudo is..

… a userspace memory allocator

… designed to prevent exploitation of heap-based memory corruption vulnerabilities



Motivation

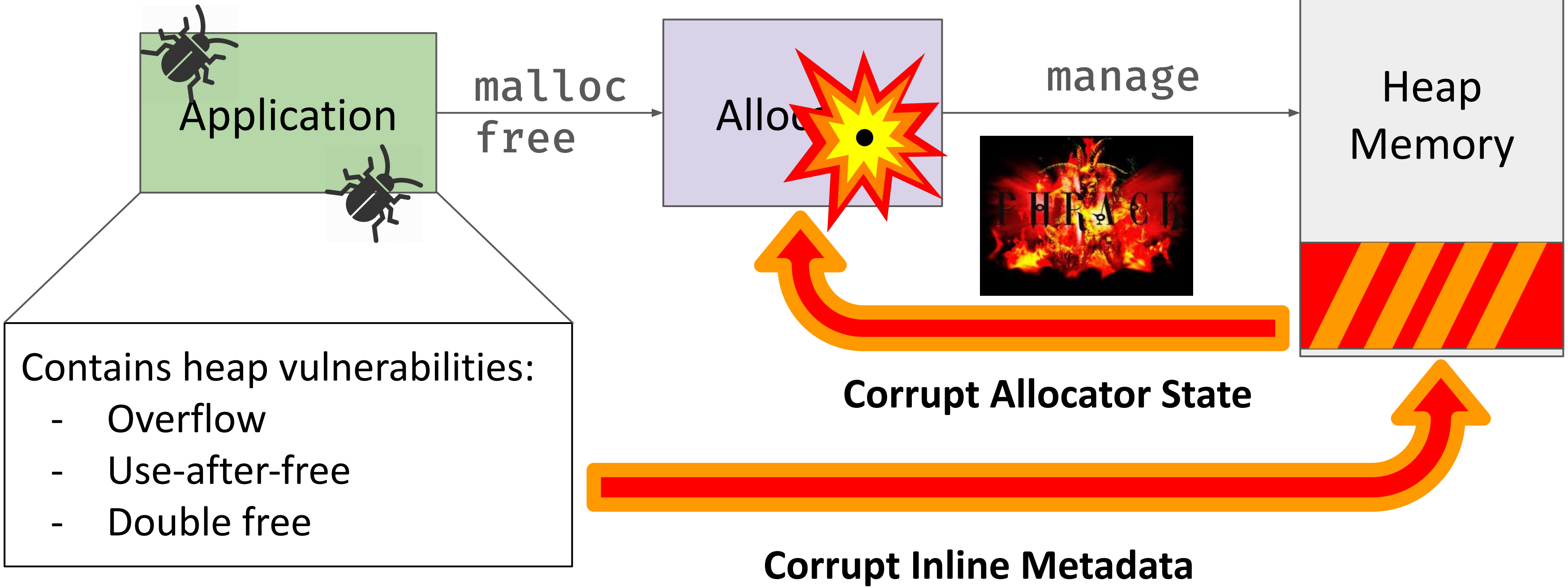
Can a heap memory corruption vulnerability in a program using Scudo be exploited?

1. Need to understand Allocator internals
2. Understand how vulnerability may be leveraged against the allocator

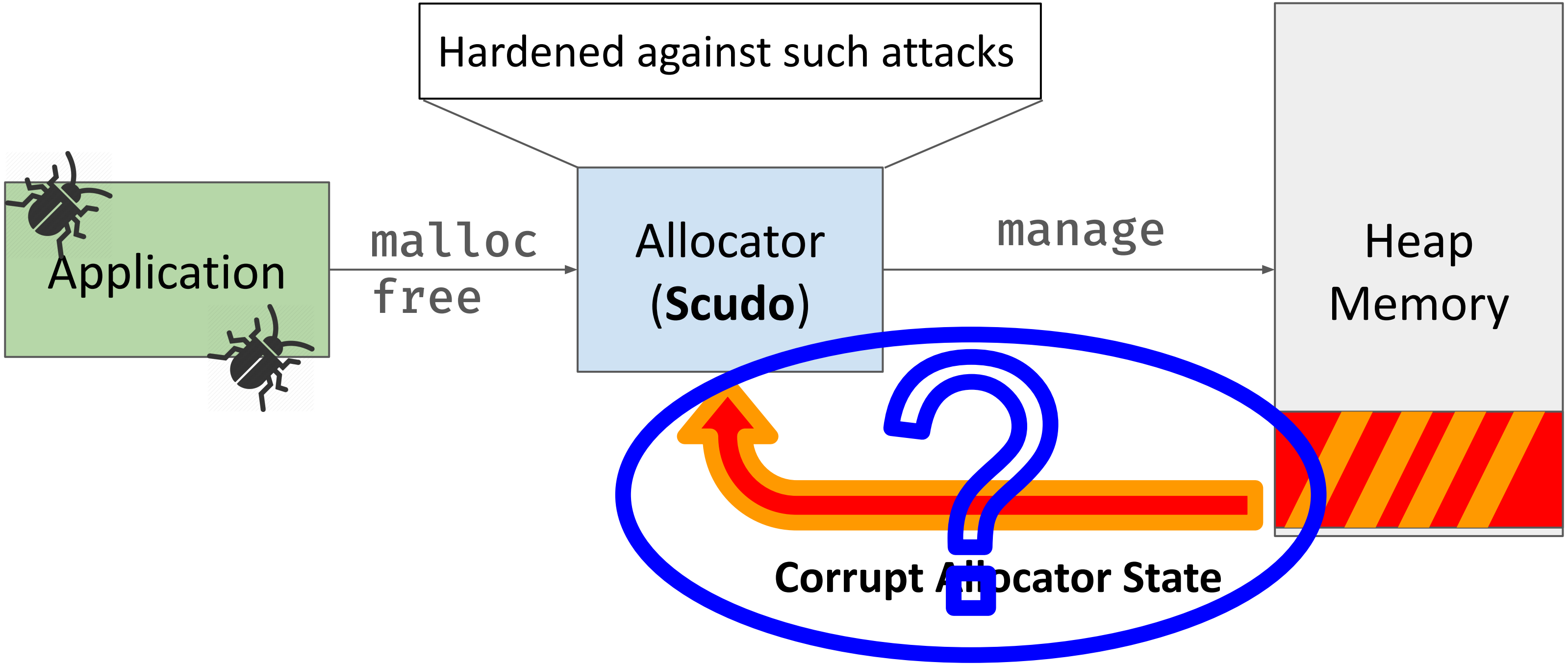
Outline

1. Exploiting the Allocator
2. Scudo Security Measures & Internals
3. Android & Scudo
4. Exploiting Scudo
   a. General Idea
   b. Case Study

Exploiting the Allocator



Is Exploiting the Allocator still possible for Scudo?



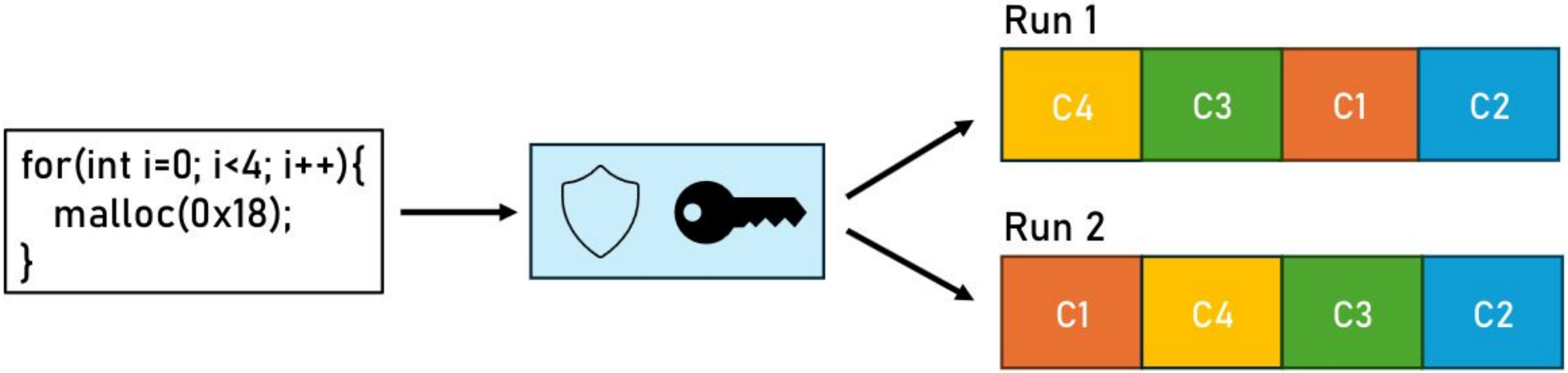
Threat Model: Able to corrupt heap memory

Scudo Internals & Security Measures

1. Randomization
2. Protection
3. Normal Chunks
4. Large Chunks

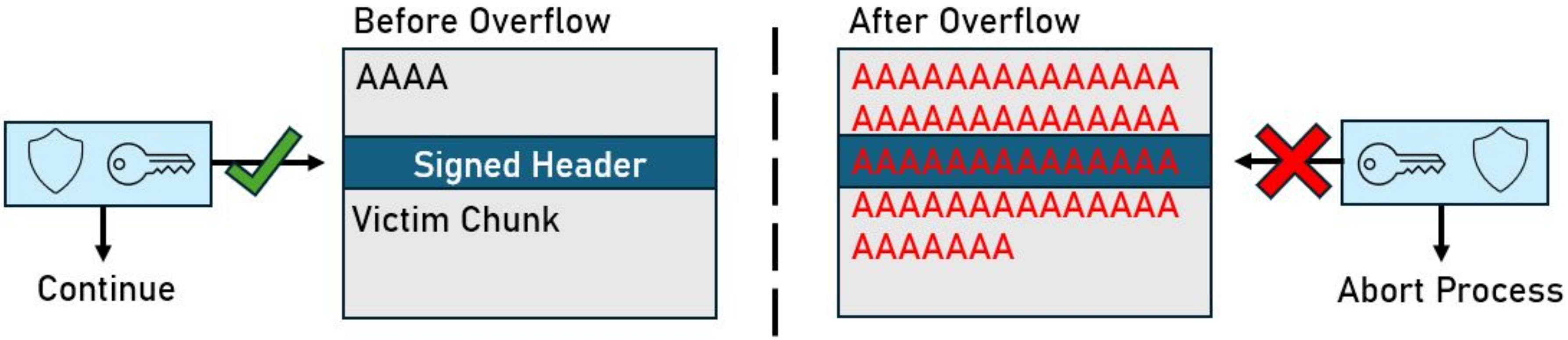
Randomization: Scudo Randomizes the Address of Allocations

Prevent attackers from arranging the heap in a particular layout.



Protection: Scudo protects inline Heap Metadata

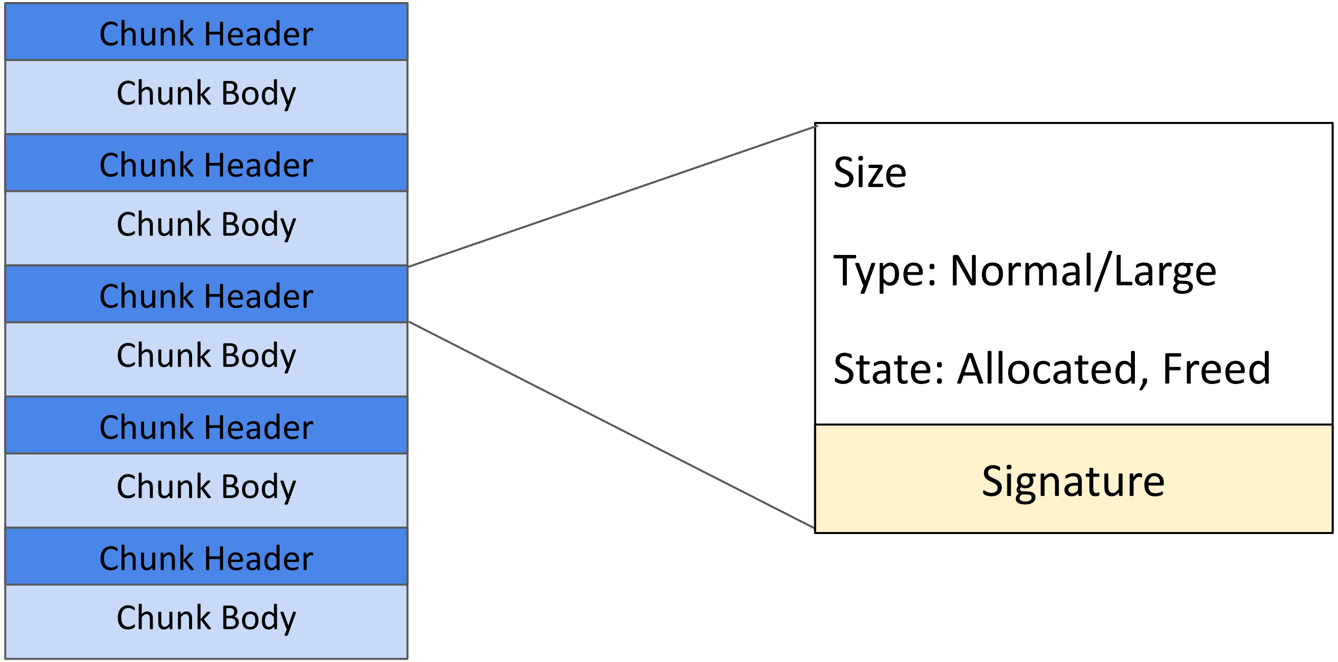
Chunk headers are signed, Scudo verifies the signature before parsing the metadata



Normal Chunks

For normal Chunks only the chunk header is stored inline

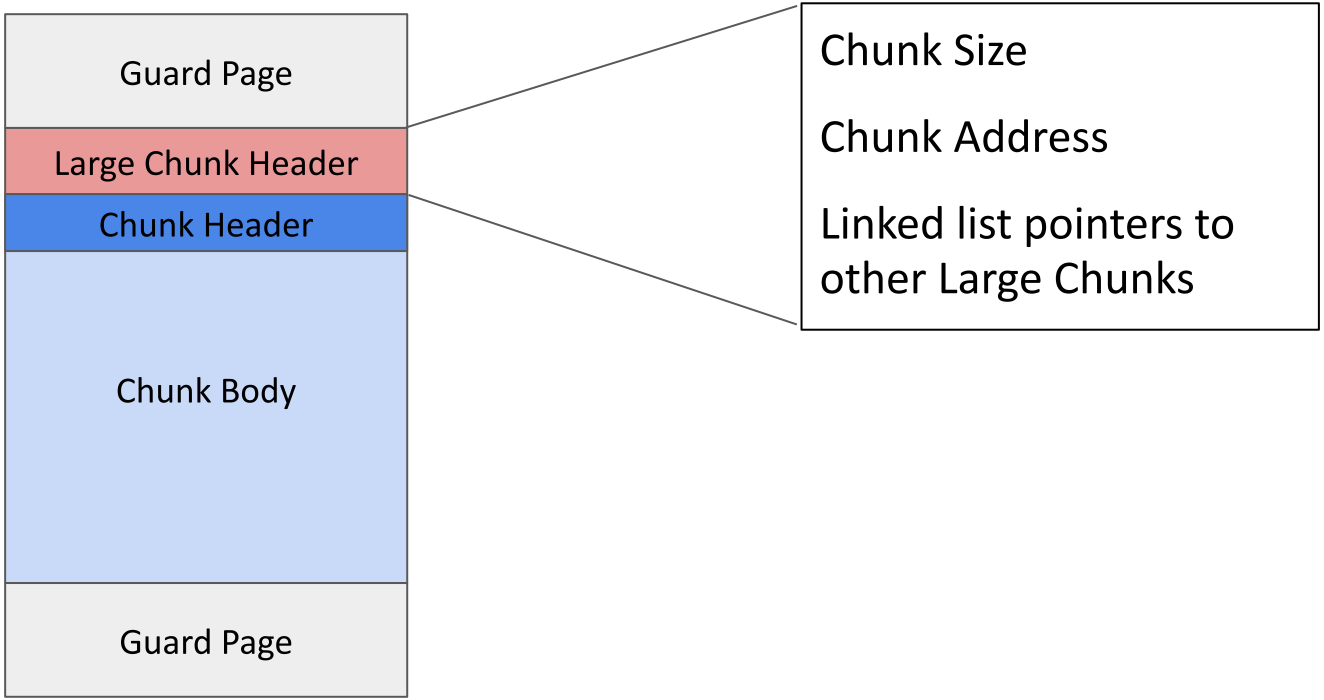
Normal Chunks (size < 65536 bytes)



Large Chunks

Only one instance where unprotected inline pointers are stored

Large Chunks (size > 65536 bytes)



Prospects of Exploiting Scudo are Looking Bleak

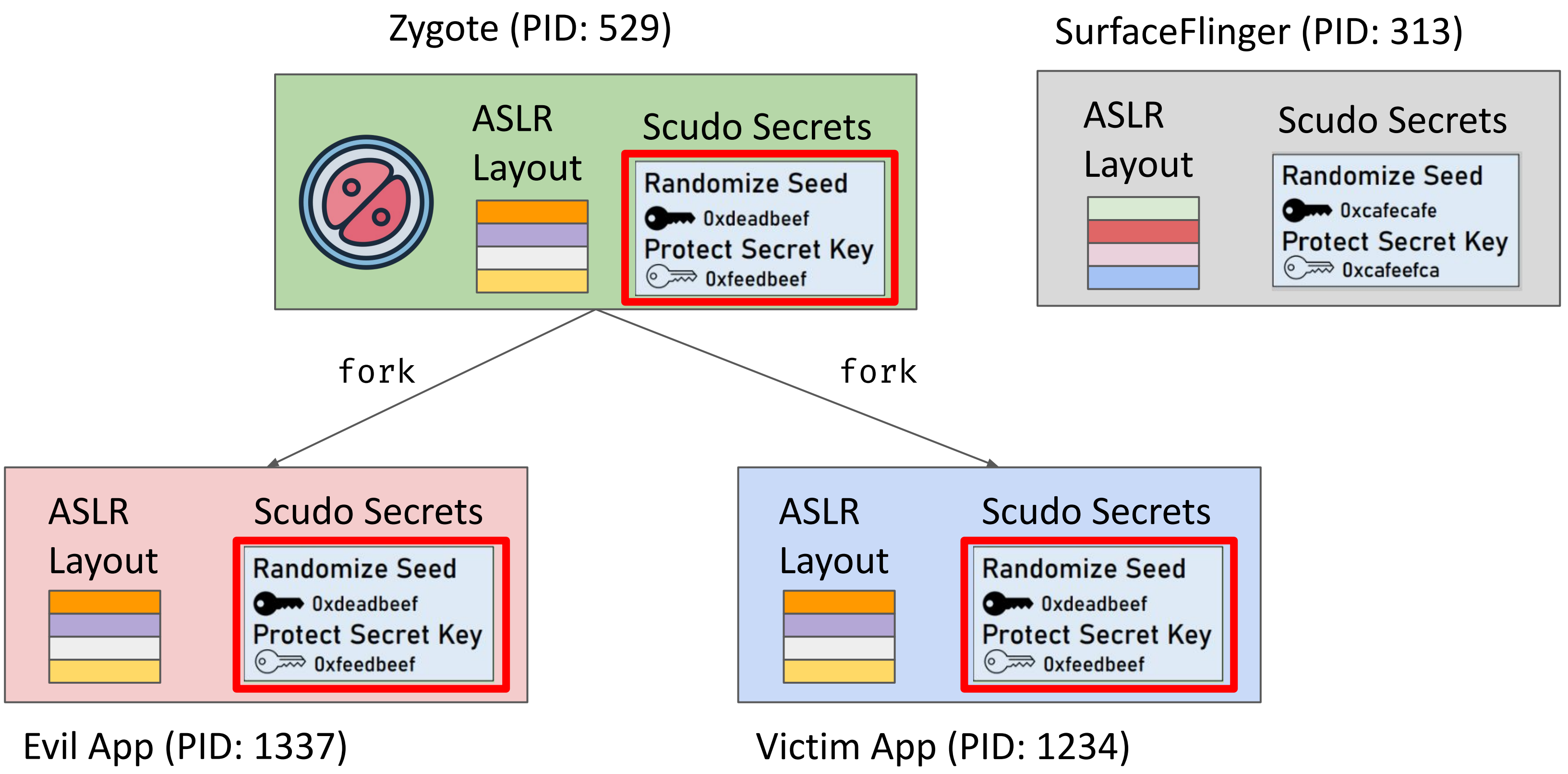
Only corruptible metadata is the normal chunk header…

… But the chunk header only stores a minimal amount of information

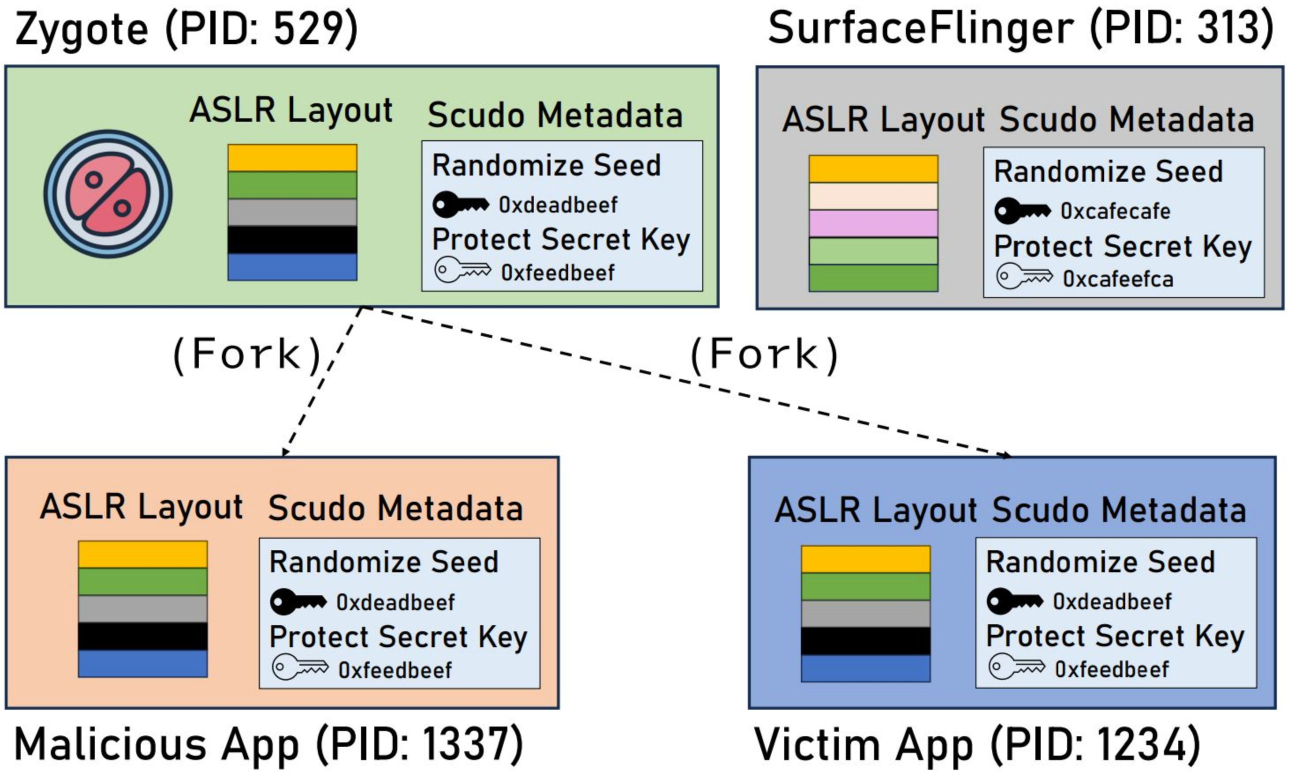
… **And** the chunk header is protected by a signature

Let’s get inspired by looking at the Android Userspace :)

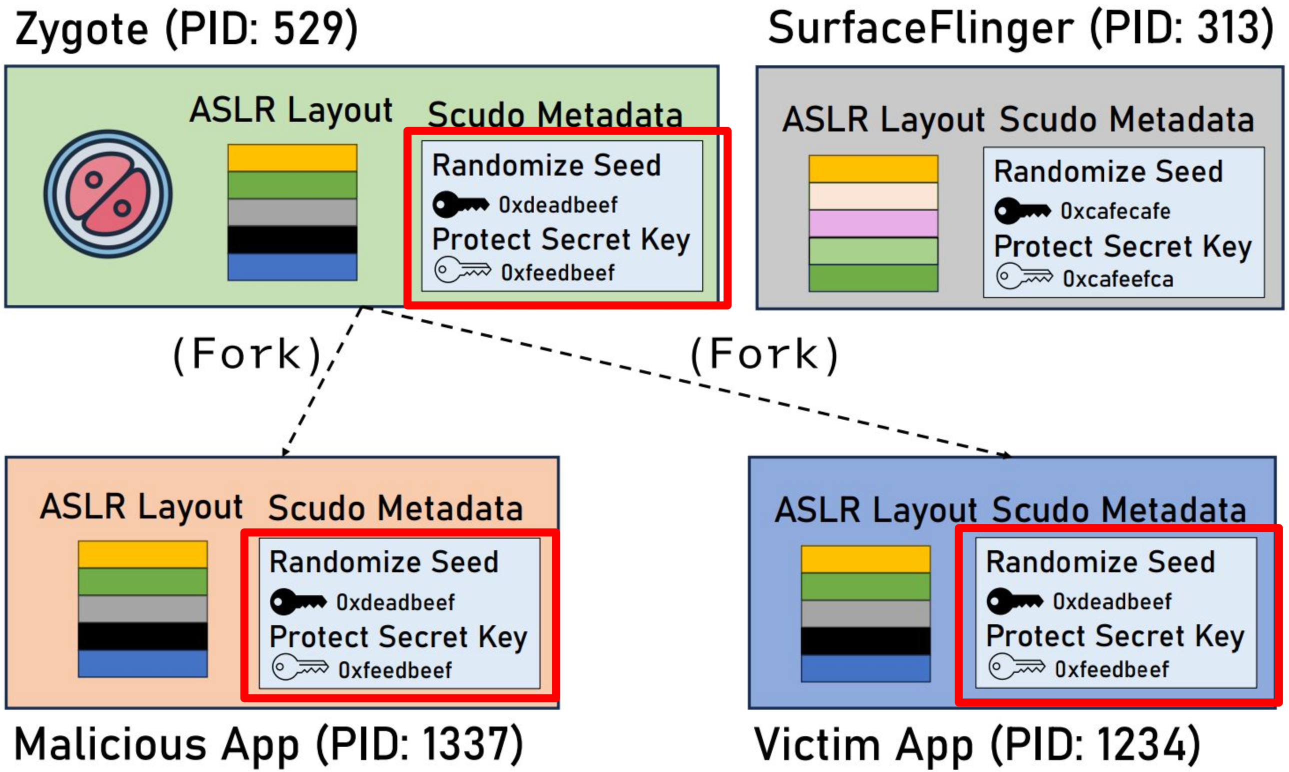
Android’s Performance Optimization Weakens Scudo



Android’s Performance Optimization Weakens Scudo



Android's Performance Optimization Weakens Scudo



ASLR and Scudo Secrets are shared between Apps

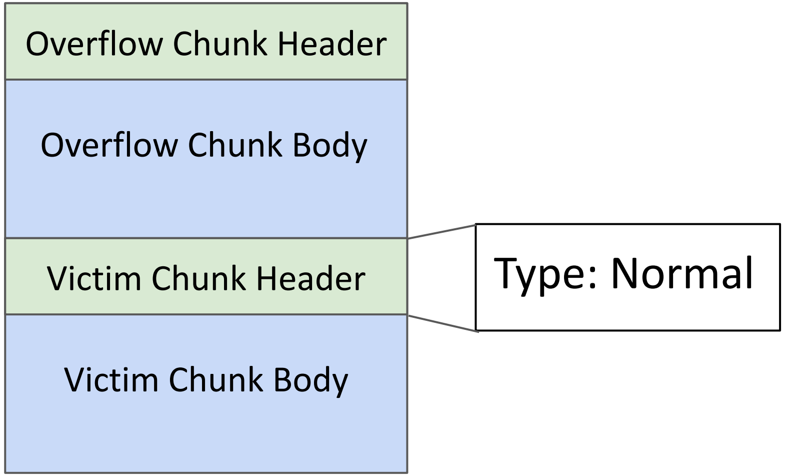
Consequences for Zygote-forked to Zygote-forked attack scenario:

Shared ASLR and Randomize Secret => Know addresses of all chunks

Shared Signature Secret => Manipulate chunk header (sign with own secret)

Exploiting Scudo with a Heap Buffer Overflow

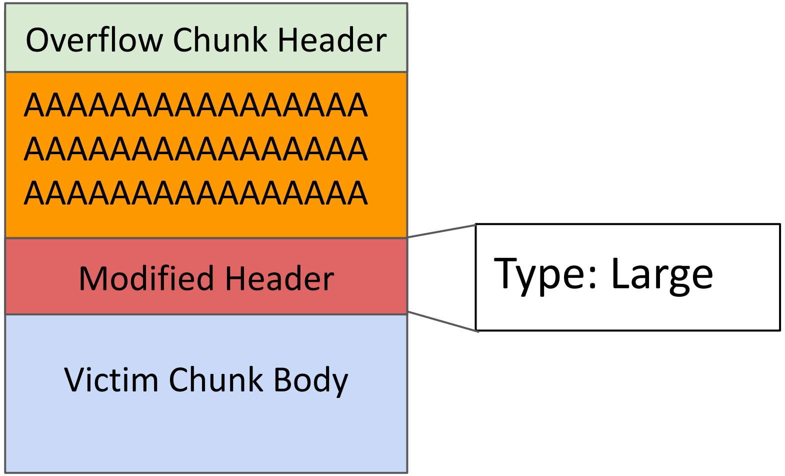
1. Arrange Chunk so layout becomes exploitable (possible because addresses are predictable)



Exploiting Scudo with a Heap Buffer Overflow

1. Arrange Chunk so layout becomes exploitable (possible because addresses are predictable)

2. Overflow and modify the victim chunk header (change type to large)

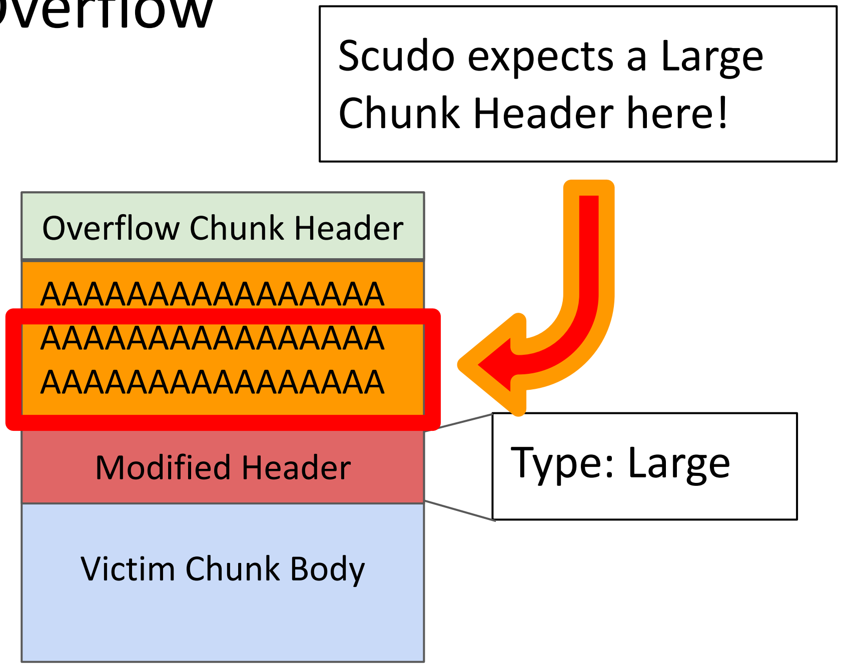


Exploiting Scudo with a Heap Buffer Overflow

1. Arrange Chunk so layout becomes exploitable (possible because addresses are predictable)

2. Overflow and modify the victim chunk header (change type to large)

3. When victim chunk is freed Scudo parses attacker-controlled pointers

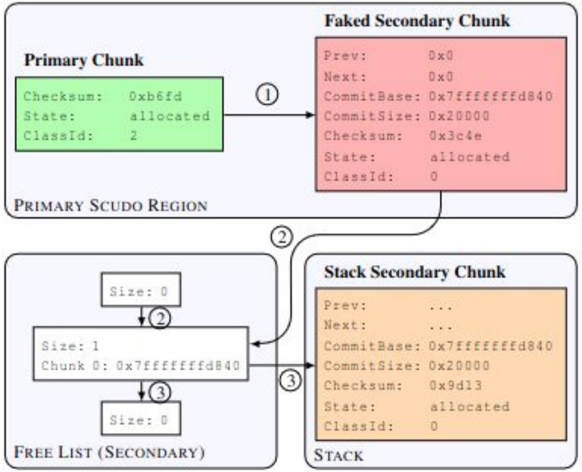


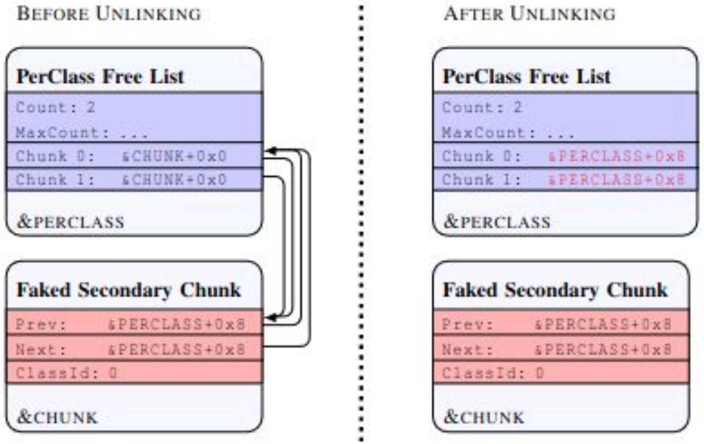
Two Techniques to Make Scudo allocate Chunk at Chosen Address

Forged Commitbase: directly insert fake large chunk into large chunk free list

Unsafe Unlink: Modify linked list pointers to corrupt a normal chunk free list

Refer to the Paper for the details!





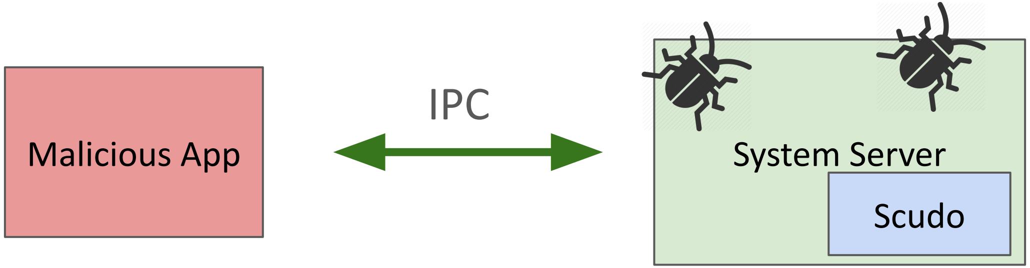
Feasible? Exploiting a Heap Underflow in the System Server

System Server is a highly privileged process, hosting multiple system services.

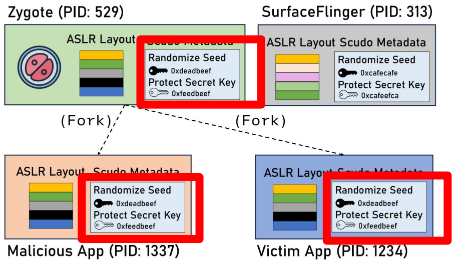
Apps interact with the system server over Binder IPC.

Backport CVE-2015-1528 to Android 14 (Heap overflow & underflow)

Use Forged Commitbase technique to allocate a chunk on the stack and hijack the PC (ROP)



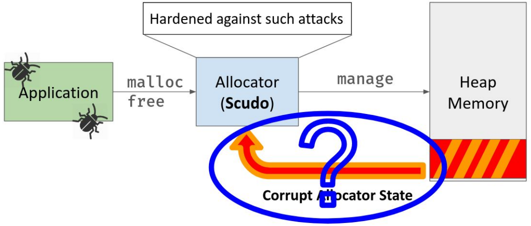
Conclusion



Discuss Scudo security mechanism in the context of android

Discover exploitation techniques against Scudo

Case study targeting the system server on Android 14



Tooling to analyze Scudo heap state and source of case study exploit:

https://github.com/HexHive/scudo-exploitation