Blender: Self-randomizing Address Space Layout for Android Apps

Mingshen Sun *, John C.S. Lui *, and Yajin Zhou †

* The Chinese University of Hong Kong
† Qihoo 360 Technology Co. Ltd.

September 21, 2016

Introduction – Android Malware

- **Mobile devices become the biggest target among all threats**

Report

From 2004 to 2013 we detected nearly 200,000 samples of malicious mobile code. In 2014 there were 295,539 new programs, while the number was 884,774 in 2015. — Kaspersky [1]

[1] https://securelist.com/analysis/kaspersky-security-bulletin/73839/mobile-malware-evolution-2015/

Introduction – Android Malware

- **Android malware samples accounted for *98%* of all mobile threats**

- **Trojan**
- **Spyware**
- **Phishing apps**
- **Ransomware**
- **Rootkit**
- **…**

[2]

[2] http://www.phonearena.com/news/Malware-on-Android---a-myth-or-real-threat_id37322

Introduction – Android Vulnerabilities

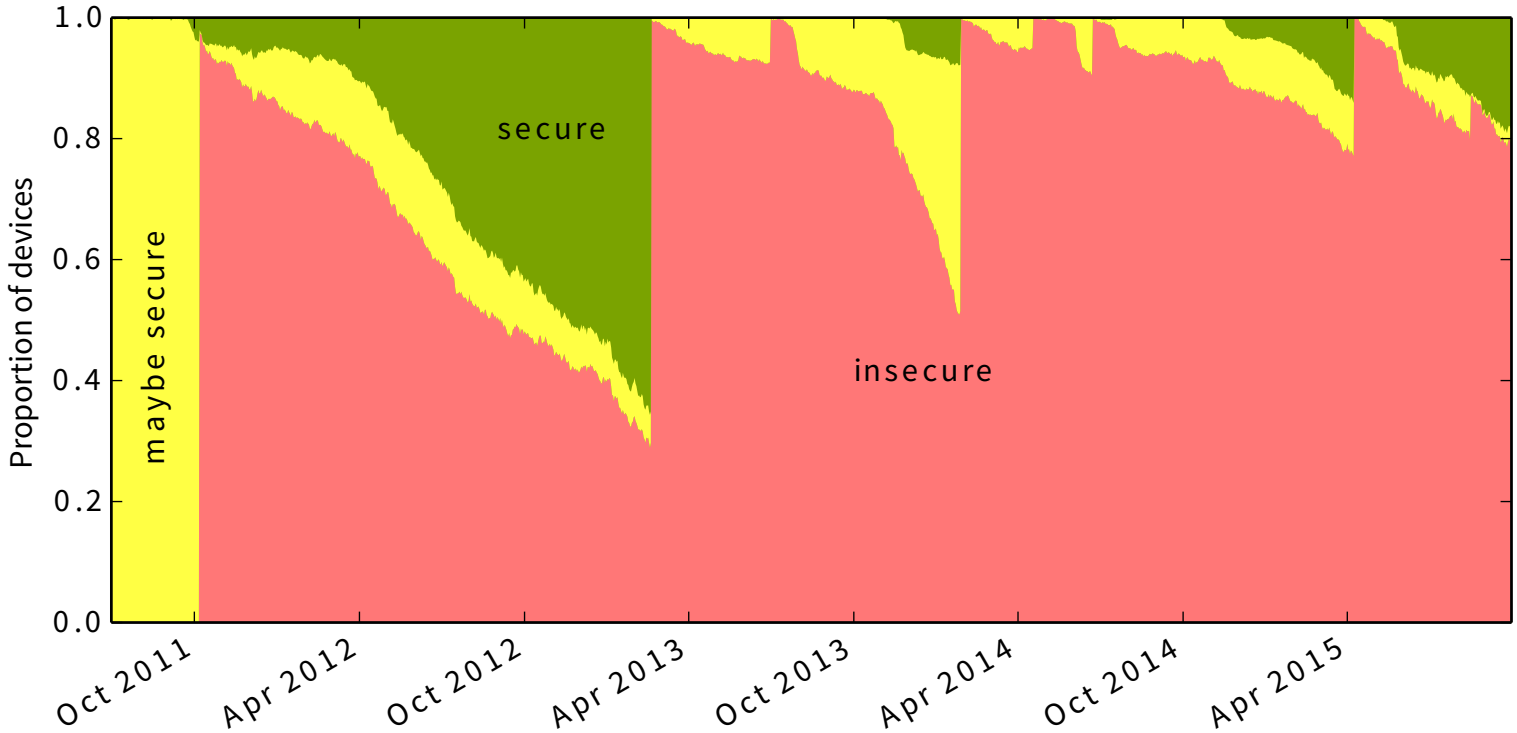


Figure: Proportion of devices running vulnerable versions of Android.[3]

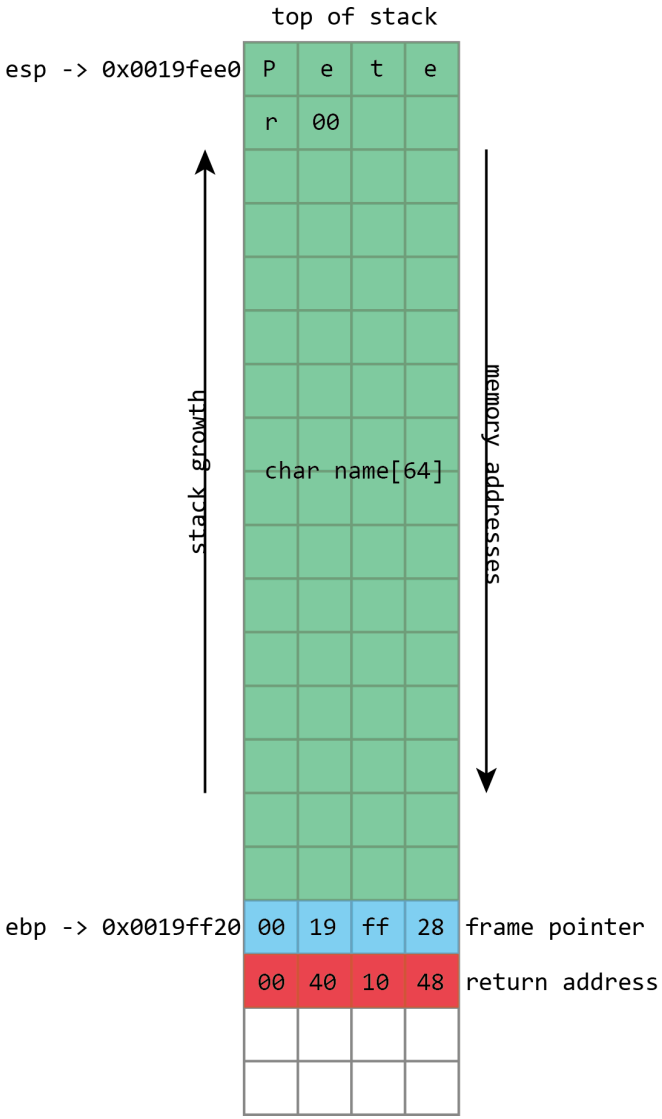
[3] http://androidvulnerabilities.org

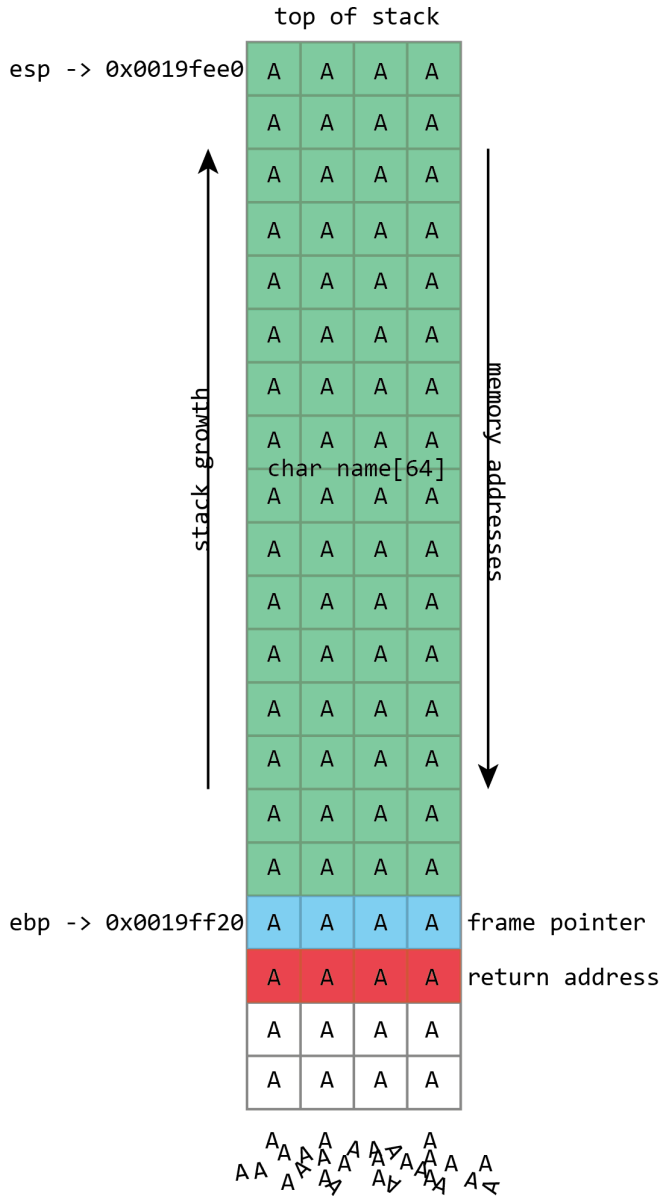
Introduction – Android Vulnerabilities

- **Google disclosed 108 CVE numbers (Common Vulnerabilities and Exposures) for Android in July 2016**
- **Nine of them are critical vulnerability (e.g., remote code execution and privilege elevation)**

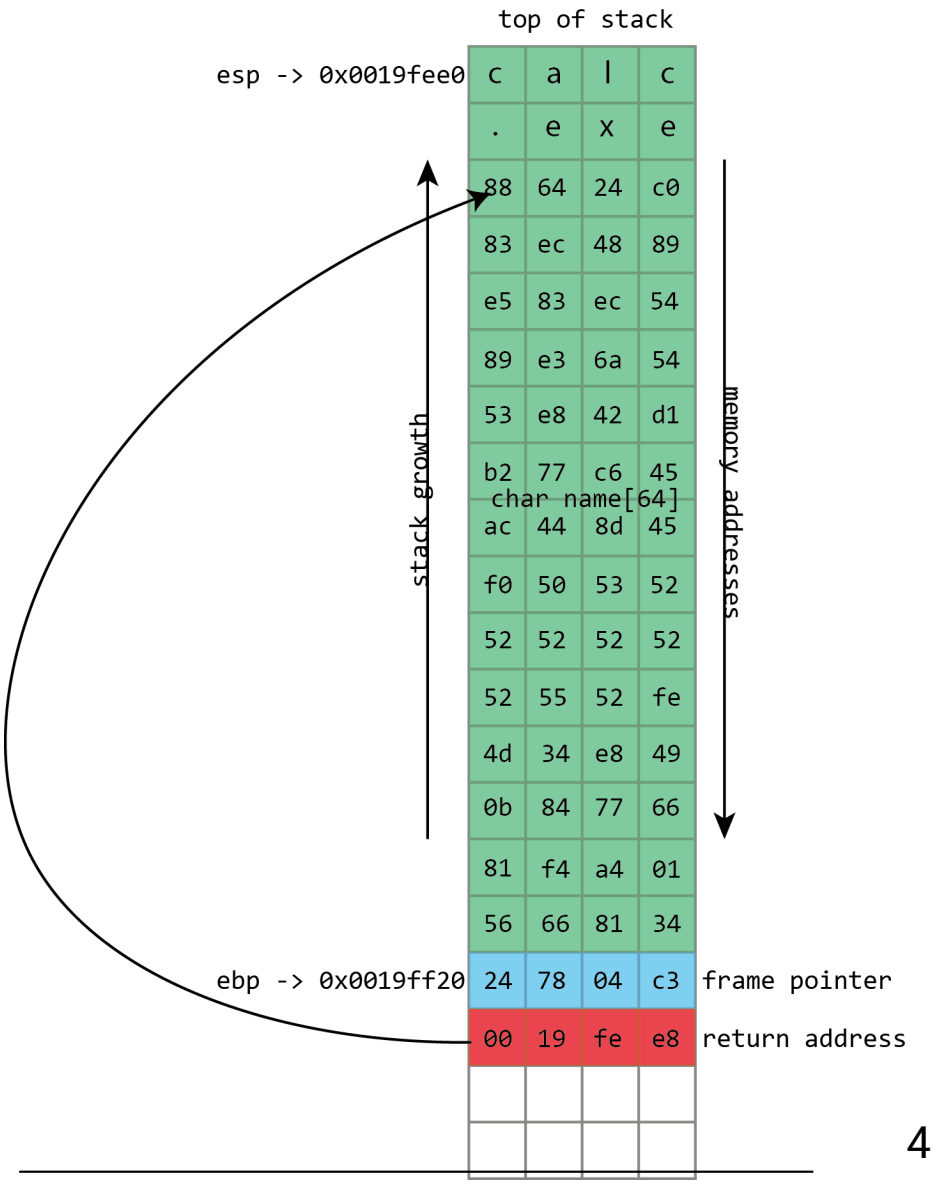
Issue	CVE	Severity	Affects Nexus?
Remote code execution vulnerability in Mediaserver	CVE-2016-2506, CVE-2016-2505, CVE-2016-2507, CVE-2016-2508, CVE-2016-3741, CVE-2016-3742, CVE-2016-3743	Critical	Yes
Remote code execution vulnerability in OpenSSL & BoringSSL	CVE-2016-2108	Critical	Yes
Remote code execution vulnerability in Bluetooth	CVE-2016-3744	High	Yes
Elevation of privilege vulnerability in libpng	CVE-2016-3751	High	Yes
Elevation of privilege vulnerability in Mediaserver	CVE-2016-3745, CVE-2016-3746, CVE-2016-3747	High	Yes
Elevation of privilege vulnerability in sockets	CVE-2016-3748	High	Yes
Elevation of privilege vulnerability in LockSettingsService	CVE-2016-3749	High	Yes
Elevation of privilege vulnerability in Framework APIs	CVE-2016-3750	High	Yes
Elevation of privilege vulnerability in ChooserTarget service	CVE-2016-3752	High	Yes

Introduction – Buffer Overflow





Introduction – Buffer Overflow

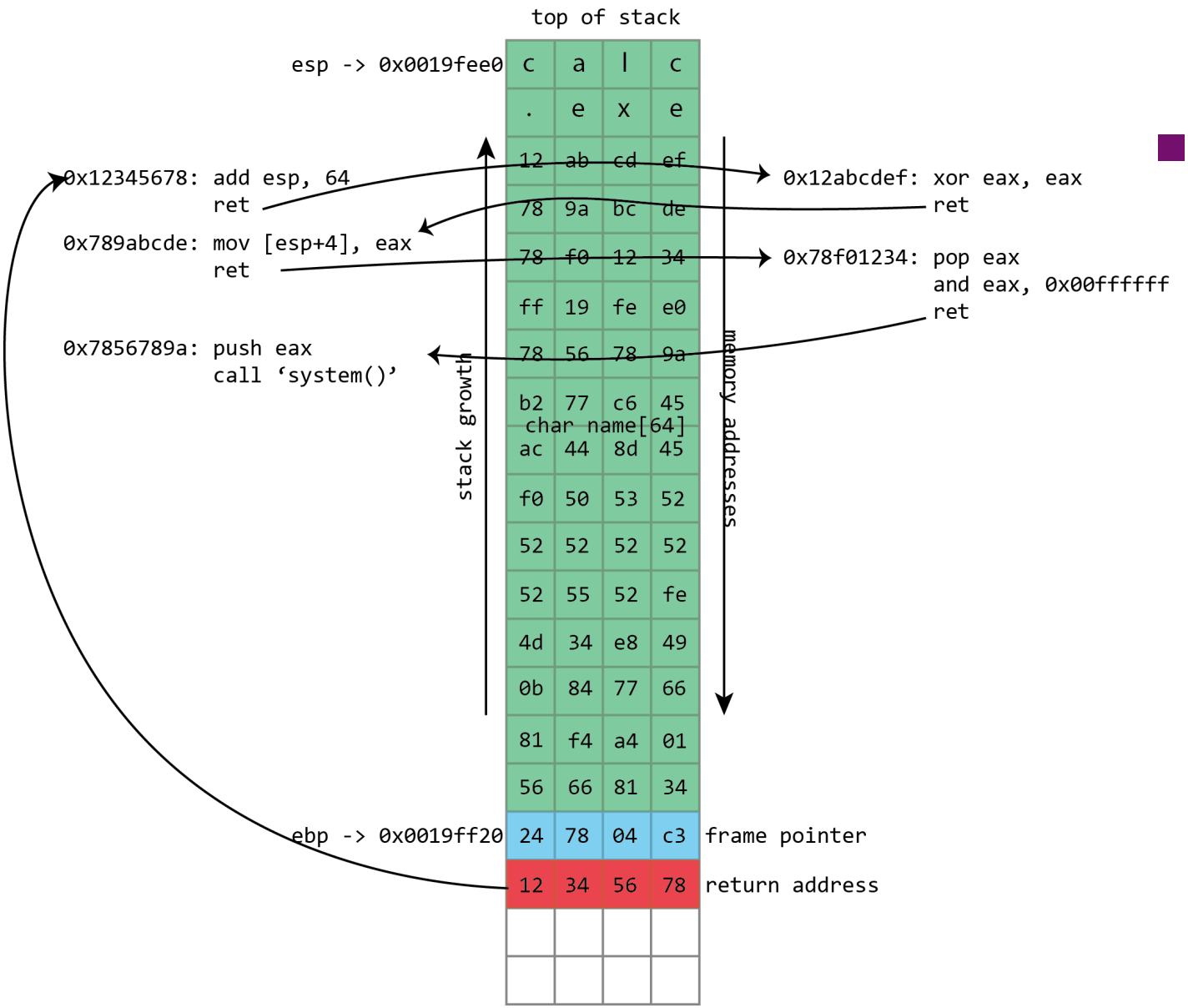


- **Solution**
  - Data Execution Prevention (DEP)
  - No-eXecute (NX) bit in CPU

4

[4] http://arstechnica.com/security/2015/08/how-security-flaws-work-the-buffer-overflow/

Introduction – ROP Attack



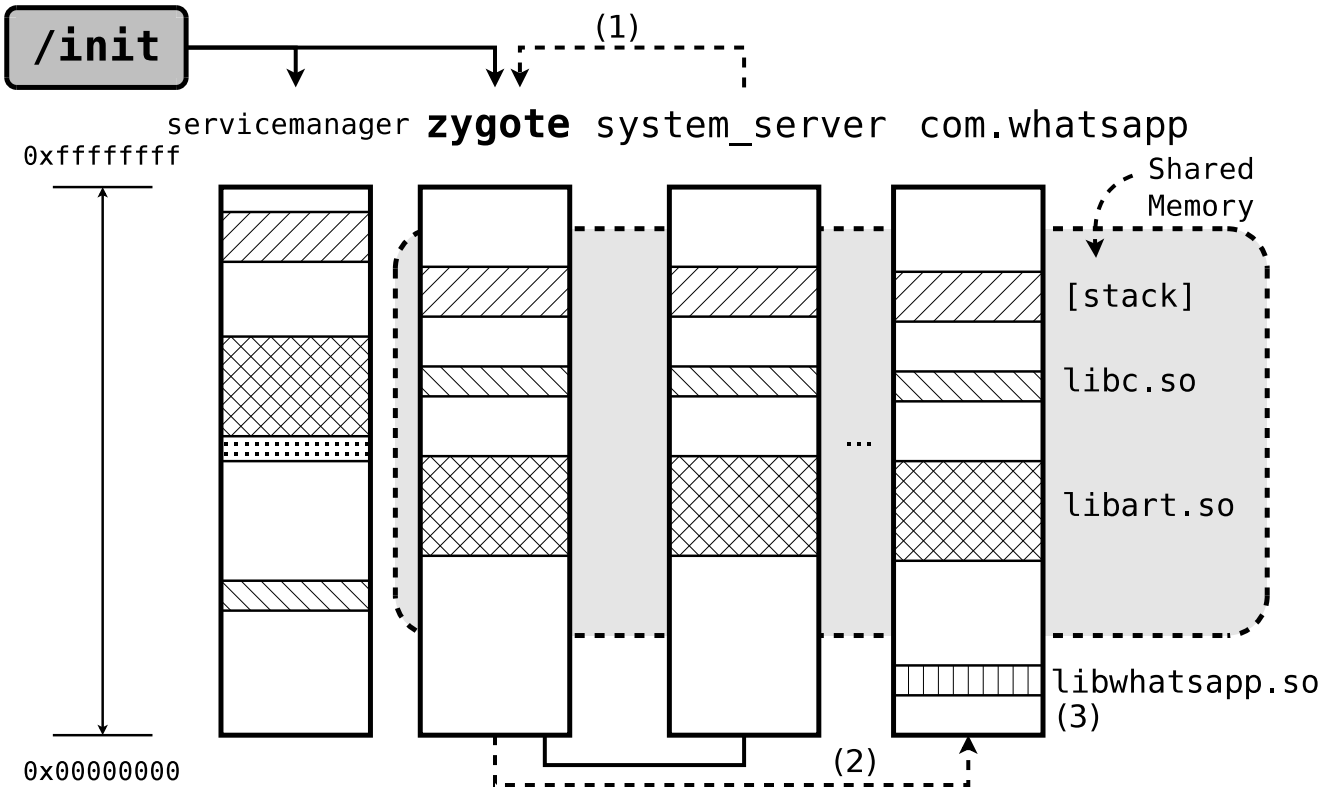
- **Solution: Address Space Layout Randomization (ASLR)**

ASLR in Android

- Android 2.x: stack
- Android 4.0: stack and shared libraries
- Android 4.0.3: stack, shared libraries, and heap
- Android 5.0: stack, shared libraries, heap, and linker

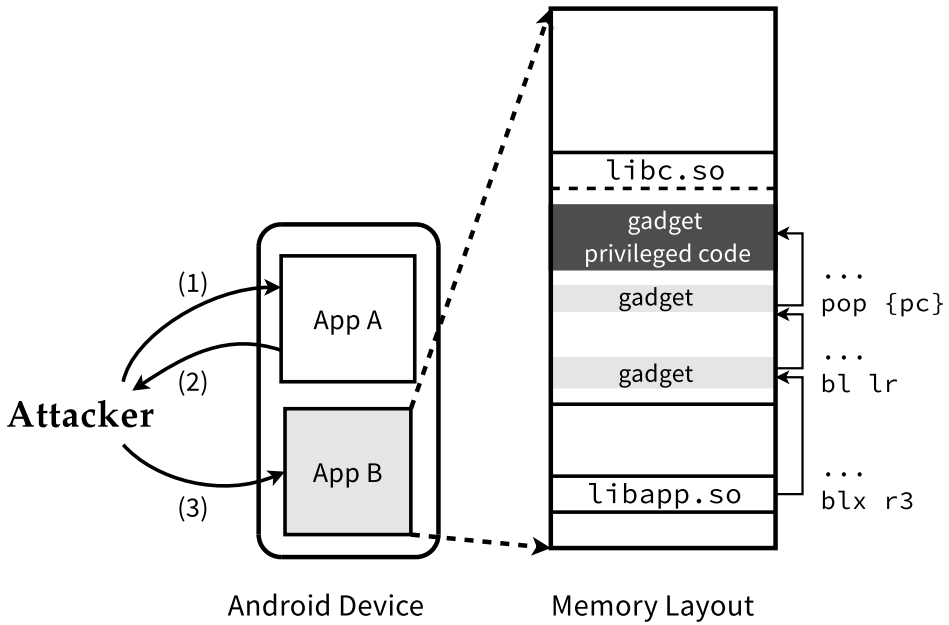
ASLR Circumvention in Android

- **What went wrong?**
  - Apps are forked from `zygote` process with identical memory layout
  - Android app creation model indirectly defeats the purpose of ASLR protection



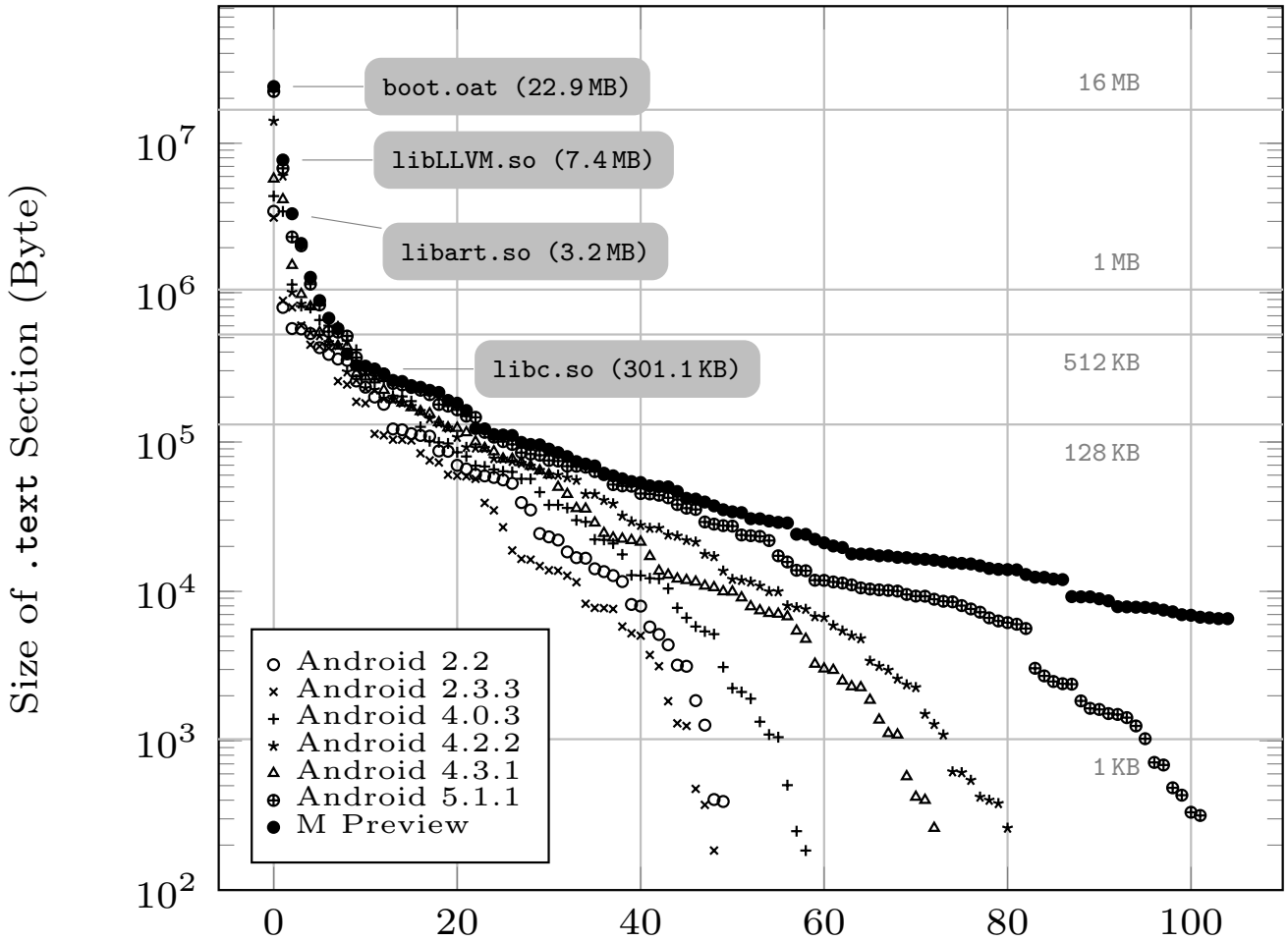
ROP attack

- **How to exploit? (ROP attack)**
  - obtain memory layout
  - calculate offsets of gadgets
  - overflow native library and hijack control flow
  - chain gadgets for attacks



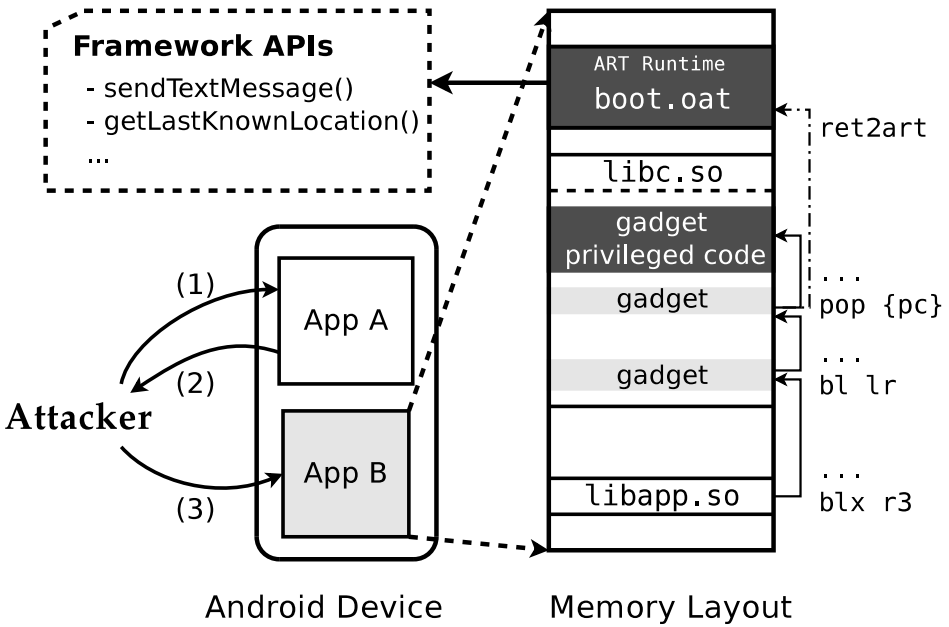
Executable Section in Android

- **Increasing .text section sizes of loaded shared libraries in zygote for different Android major versions.**



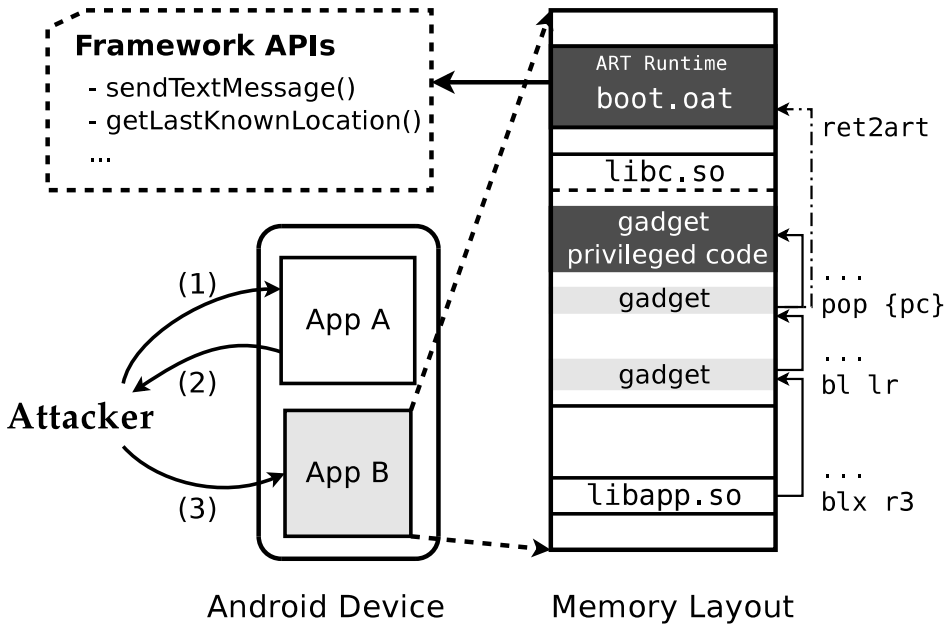
A New Attack: Ret2art

- **What went wrong?**
  - The newly-designed Android RunTime (ART) utilize ahead-of-time (AOT) compilation strategy to pre-compiled framework code into native code
  - The addresses of the pre-compiled native code of the system framework APIs are predictable



A New Attack: Ret2art

- **How to exploit?**
  - similar with traditional ROP attacks
  - prepare registers to call framework APIs
  - a more damaging and flexible attack method

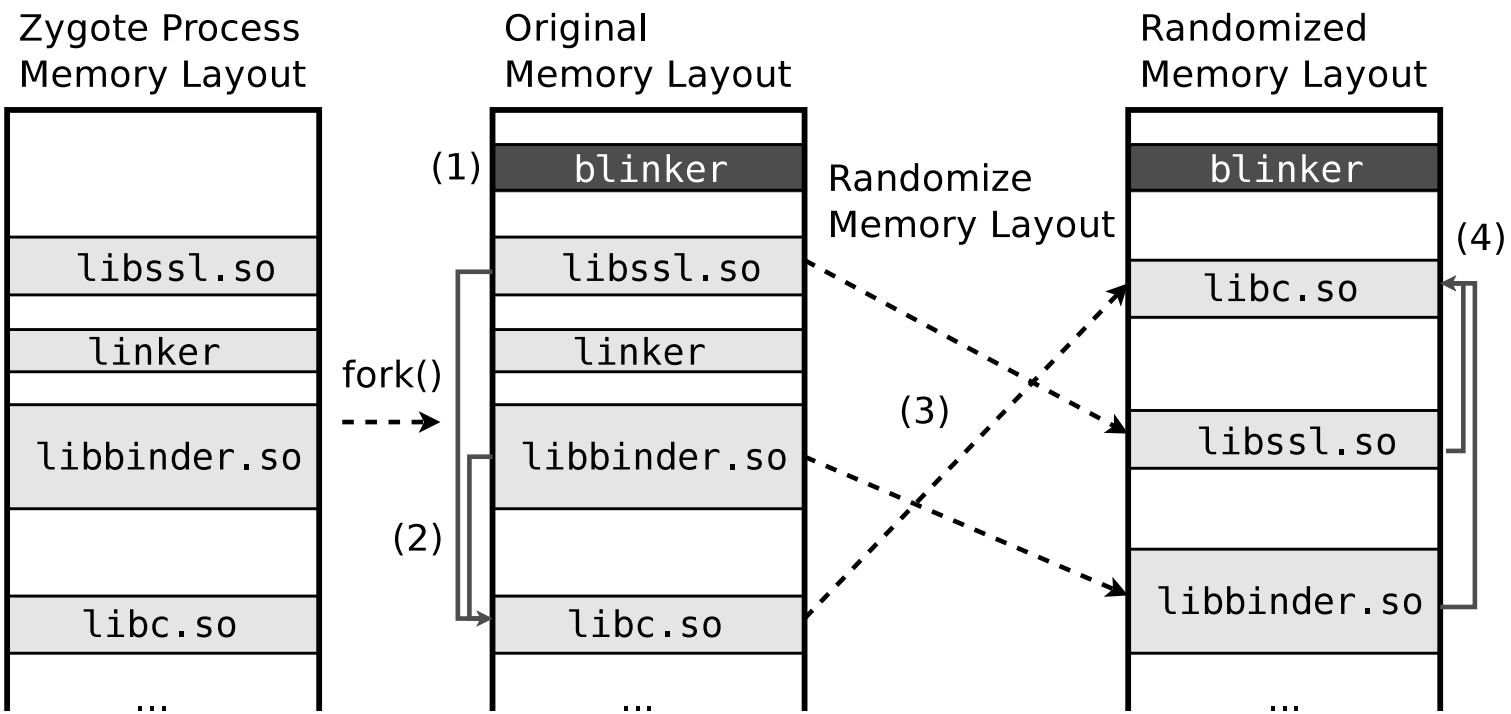


Blender – Overview

- **Blender, a user-level mitigation solution to self-randomize address space layout, provides non-invasive and easy-to-deploy self-protection for Android apps**
  - randomize the addresses of loaded system libraries for apps
  - randomize the executable code of the pre-compiled framework code in the ART runtime

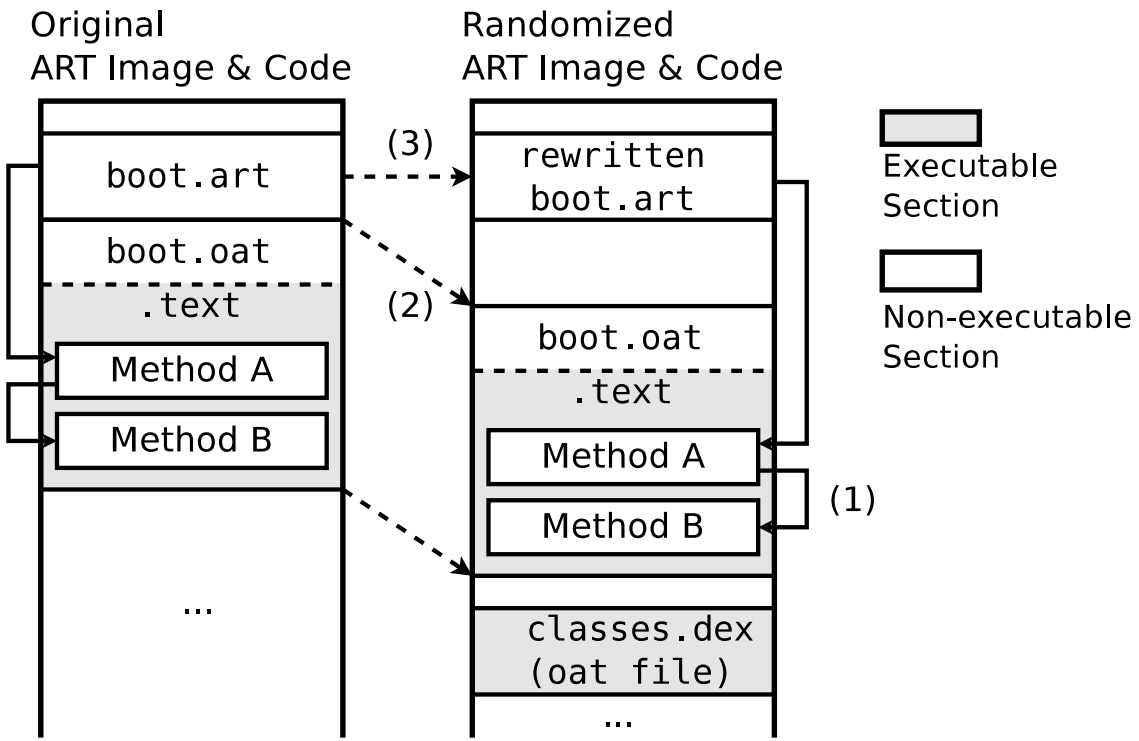
Blender – BlenderLRM

- **Blender Library Randomization Module**
  1. Customized dynamic linker called `blinker`
  2. Generate dependency graph between libraries for rearrangement
  3. Rearrange loaded libraries and fix GOTs (maintains pointers to external functions)



Blender – BlenderART

- **Blender ART Randomization Module**
  1. Patch method absolute addresses
  2. Rearrange `boot.oat`
  3. Fixup class linker data instance
  4. Optimization for improving startup time



Evaluation – Effectiveness

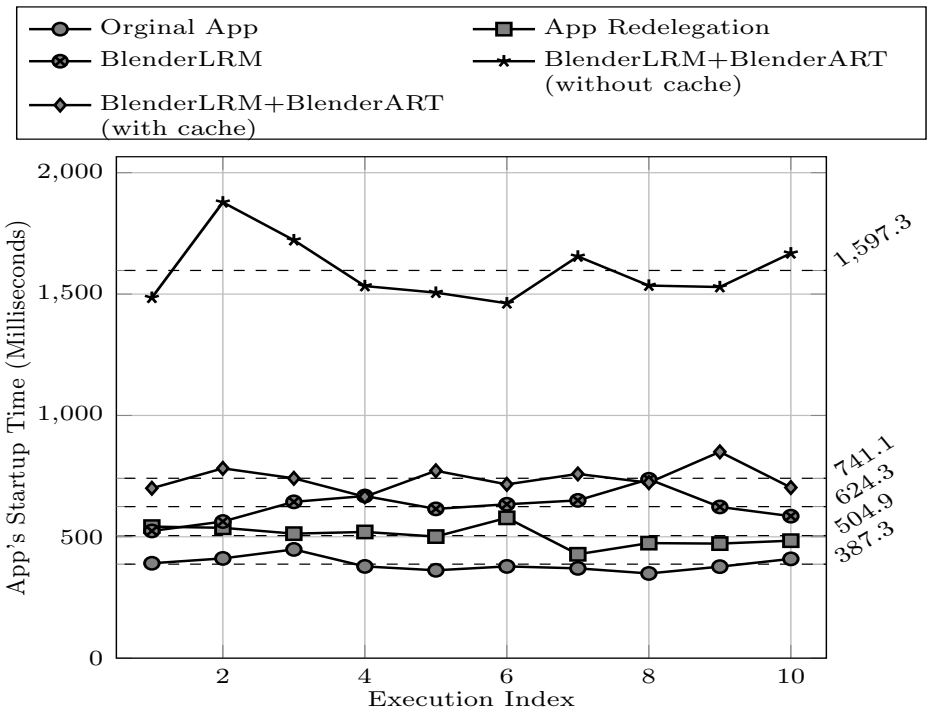
- **App randomness**
  - app entropy
  - We define $\{x_1, x_2, \ldots, x_n\}$ as base addresses of the library m, and n is the number of executions for one app.
  - $H(X_m) = -\sum_{i=1}^{n} p(x_i) \frac{\ln p(x_i)}{\ln n}$
  - average app entropy: $R(A) = \frac{\sum_{m \in M} H(X_m)}{|M|}$

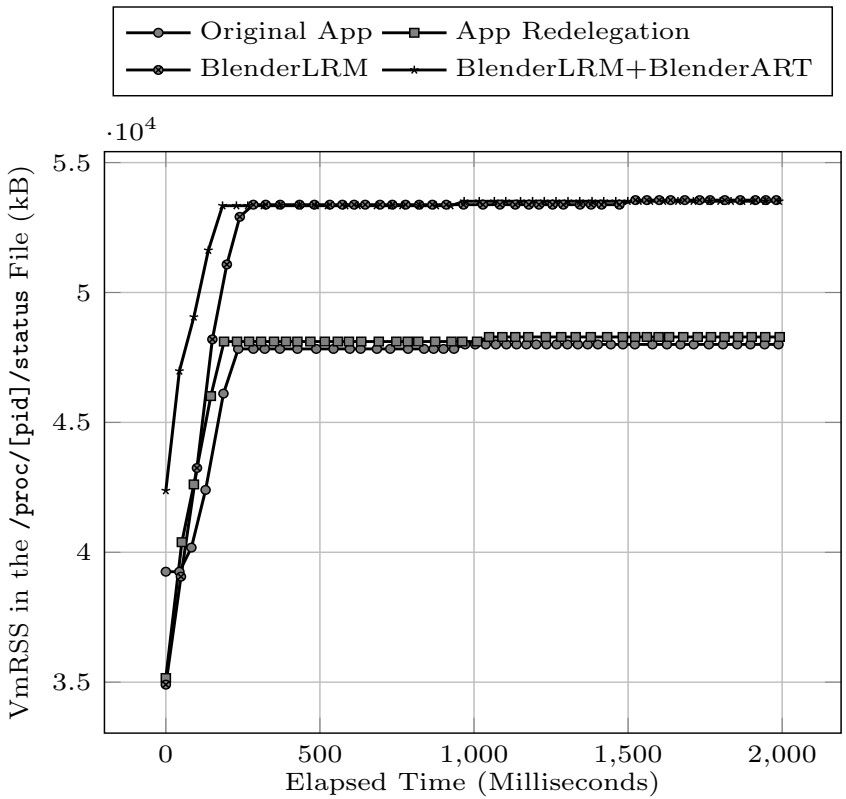
Mode	App Entropy R(A)
Original App	0.005
BLENDERLRM Only	0.981
BLENDERLRM and BLENDERARM	0.991

Table: Entropy Analysis Results

Evaluation – Performance

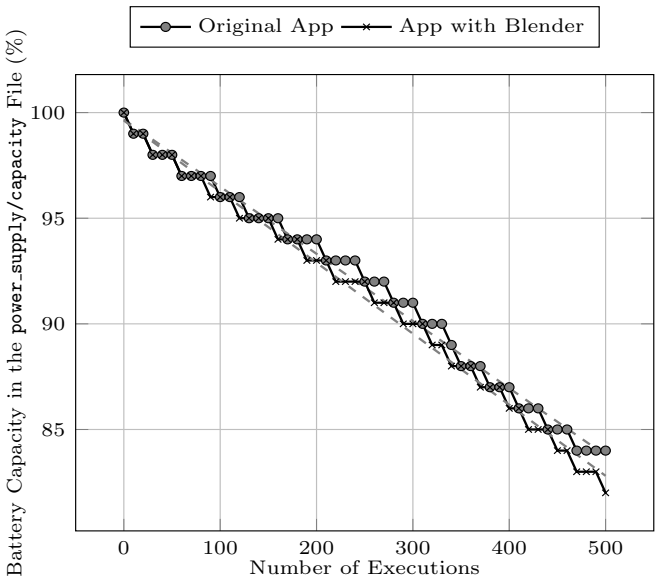
- **Startup time: launch time of apps**
  - 360 ms start time (code start) overhead
- **Memory overhead: memory usage during startup**
  - 11.5% memory overhead





Evaluation – Performance

- **Runtime overhead: benchmarks of CPU, Memory, I/O, etc.**
  - negligible overhead at runtime
- **Battery overhead: battery usage measurement**
  - 1% batter overhead



	Baseline	BlenderLRM	Full Blender
CPU	35915	36480	35969
Memory	13900	13846	14653
I/O	5874	5893	5900
2D	330	330	298
3D	1967	2019	1981
Total	57986	58568	58801

Table: Benchmark scores.

Summary

Contributions

1. discover a new attack surface in recent Android RunTime
2. propose a non-invasive solution to mitigate the threat of the weakened ASLR implementation in Android system
3. implement a system to self-randomize address space layout for both native libraries and the ART runtime

Thank you!

Thank you.

Question?