Poster Abstract: Blender: Self-randomizing Address Space Layout for Android Apps

Mingshen Sun[1], John C.S. Lui[1], and Yajin Zhou[2]

[1] The Chinese University of Hong Kong
[2] Qihoo 360 Technology Co. Ltd.

Abstract. We first demonstrate that the newly introduced Android RunTime (ART) in latest Android versions (Android 5.0 or above) exposes a new attack surface, namely, "return-to-art" (ret2art) attack. Unlike traditional return-to-library attacks, the ret2art attack abuses Android framework APIs (like the API to send SMS) as payloads to conveniently perform malicious operations. For instance, attackers could easily construct the payload to send SMS, get GPS locations on behalf of the vulnerable app if the app has corresponding permissions, without the need to understand the tedious details of the binder IPC mechanism and bridge the semantic gaps between the high level framework APIs and low level system calls. This new attack surface, along with the weakened ASLR implementation in the Android system, makes the successful exploiting of vulnerable apps much easier.

To mitigate this threat, we then propose a user-level solution called Blender. Our system provides the capability of memory layout *self-randomization* to (sensitive) Android apps with high security requirement, without waiting for the changes of the Android framework nor underlying Linux kernel. Specifically, Blender first randomizes memory layout of loaded system libraries which are inherited from the `zygote` process. Then, to prevent the ret2art attack, Blender also randomizes the ART executable runtime dynamically at startup time. It ensures that the base addresses of libraries and the ART runtime are unpredictable.

We also implement a prototype of Blender and evaluate its effectiveness and overhead. Our evaluation shows that apps using our system have a much higher memory entropy than vanilla apps. This means attackers have to try many times to successfully bypass the Android ASLR protection, instead of a single attempt. Blender incurs an increase of 6 MB memory footprint for an app. Note that, this only affects apps using our system, and does not affect other ones running on the device, an extra advantage compared with the system-wide solution. Our system increases 0.3 seconds to the app starting time, has no obvious overhead to the CPU and battery resources.

Keywords: Android, ROP, ASLR, Blender

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.

Return-Oriented-Programming (ROP) Attacks on Android

What went wrong? Return-to-library attacks.

- Android apps are forked from the `zygote` process
- Memory structures of child apps are identical and duplicated by `zygote`
- Attackers can now easily predict memory layout information

The way that Android app is created *defeats the purpose of ASLR mechanism*.

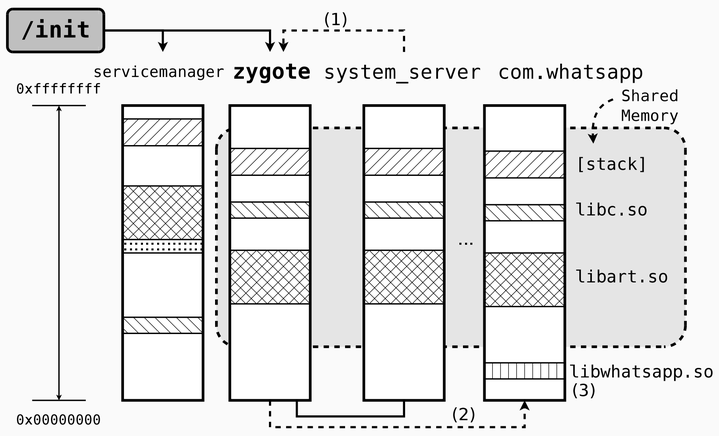

Fig. 1: Android booting and app creation process.

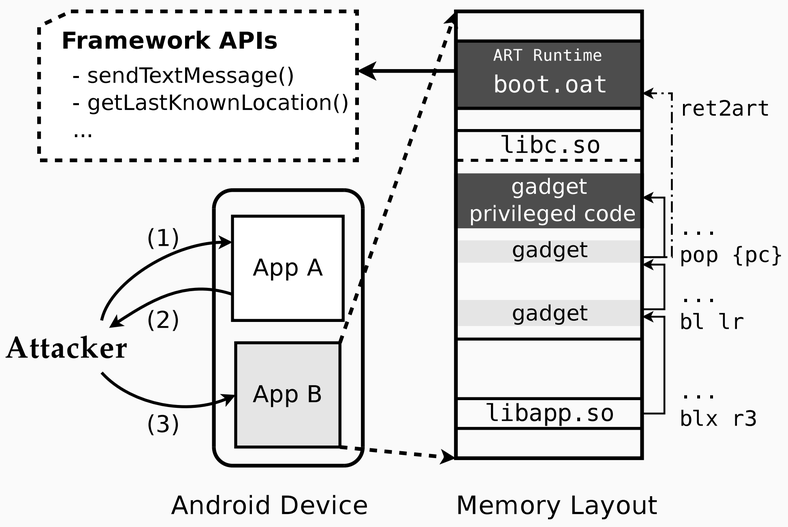

Fig. 2: ROP attack on Android (return-to-library attack and return-to-art attack).

What went wrong? Return-to-ART attacks.

- The address of the AOT-compiled framework sources (`boot.oat`) in the newly-desinged Android RunTime is predictable
- The loaded `boot.oat` increases the predictability of the memory layout of executable code regions which are the pool of ROP gadgets
- Return to `boot.oat` and leverage the well-defined Android framework APIs for attacks

Blender, a user-level mitigation solution to self-randomize address space layout, provides non-invasive and easy-to-deploy self-protection for Android apps.

Blender Library Randomization Module

Steps of randomizing libraries by BlenderLRM

1. Customized dynamic linker called `blinker`
2. Generate dependency graph for rearrangement
3. Rearrange loaded libraries and fix GOTs

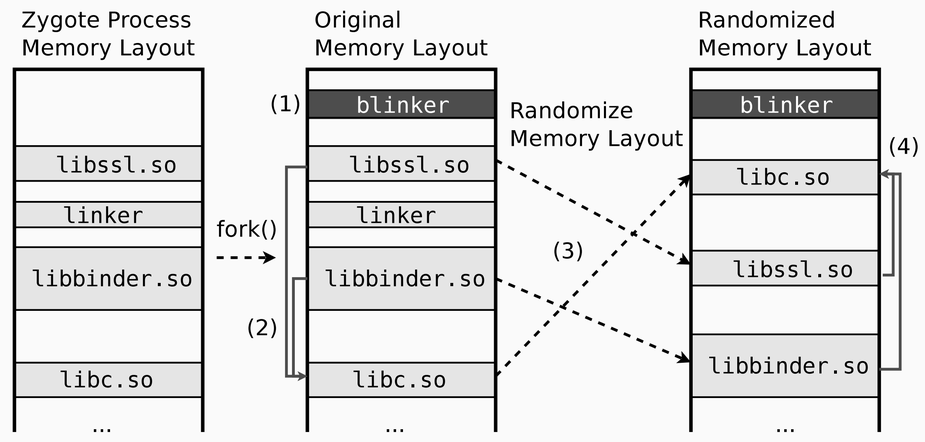

Fig. 3: Overview of BLENDER Library Randomization Module.

Blender ART Randomization Module

Steps of randomizing ART by BlenderART

1. Patch method addresses and load `boot.oat`
2. Fixup class linker data instance
3. Optimization for improving startup time

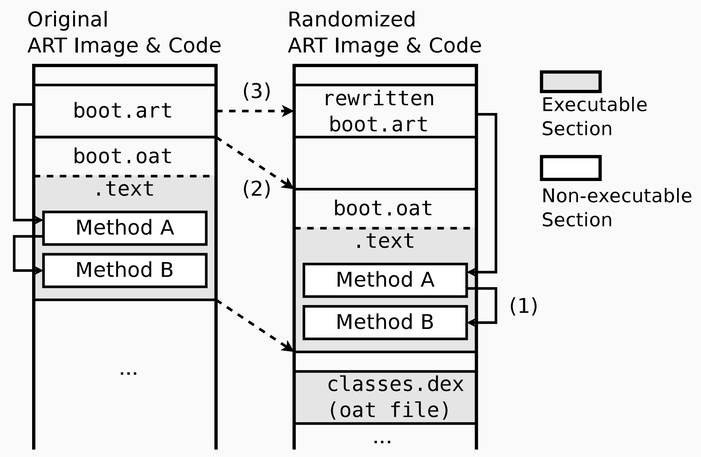

Fig. 4: Overview of BLENDER ART Randomization Module.

Implementation & Evaluation

- Implemented on Android 5.1 Lollipop on Nexus 5
- Customized Android dynamic linker, ART runtime
- Achieved high application entropy: 0.991
- Incurred 360 ms startup time (cold start) overhead
- With negligible overhead at runtime
- 11.5% memory overhead and 1% battery overhead