BYOD for Android — Just add Java

Jessica Buttigieg, Mark Vella, and Christian Colombo

PEST Research Lab @ University of Malta, Malta
[jessica.buttigieg.12][mark.vella][christian.colombo]@um.edu.mt

Bring-Your-Own-Device (BYOD) implies that the same mobile device is used for both work and personal purposes. This poses a security concern where untrusted user-installed applications might interfere maliciously with corporate ones. Android's existing fixed permissions mechanism is not a suitable countermeasure. Malware isolation through virtualization[1] and managed device scans[2] is possible, however a complete solution requires a context-specific (work/personal) policy mechanism. Our proposition, *BYOD-RV*, uses Dynamic Binary Instrumentation (DBI) and Runtime Verification (RV). DBI (in-memory code patching) avoids Android source code changes as typically required by similar approaches, e.g. [3]. RV (runtime monitoring of program correctness properties) enables expressing dynamic policy rules in Java, e.g. [2].

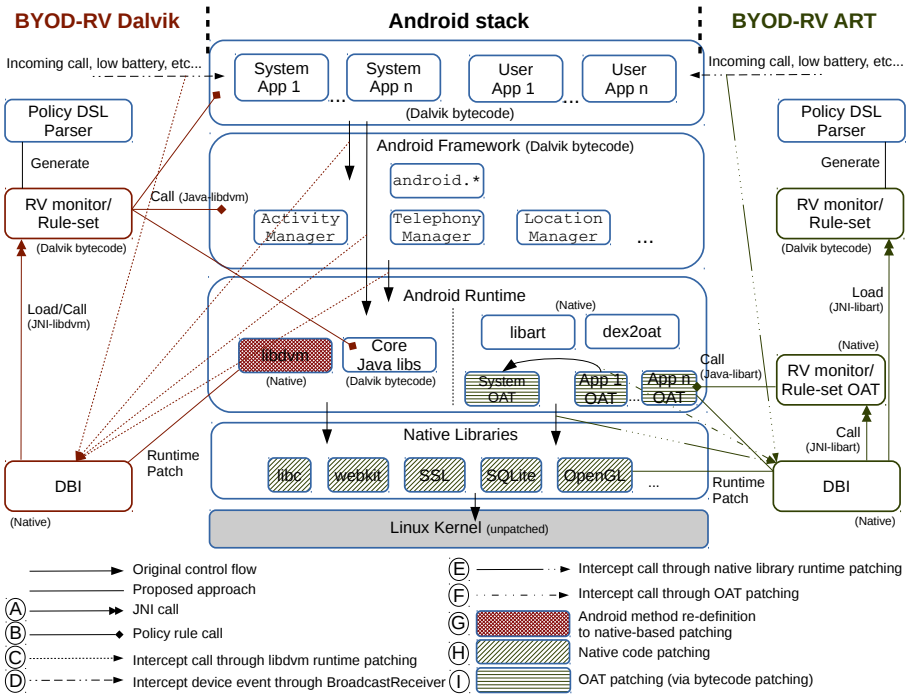

Fig. 1. BYOD-RV. Dalvik -left- and Android Runtime (ART) -right- versions.

[1] www.vmware.com/files/pdf/view/VMware-BYOD-Opportunity-Whitepaper.pdf
[2] nuvotera.com/solutions/mobile-device-management/

Method The architecture for the Dalvik runtime (`libdvm`) version is shown in Fig. 1 - left. The DBI component is loaded in process memory via `ptrace` and patches `libdvm` (G) to create in-line hooks that intercept (C) security-sensitive Android method calls by re-defining them as native. This is lightweight DBI that requires no code block copying. Device events e.g. low battery or incoming call events, are intercepted (D) with the inclusion of a `BroadcastReceiver` component. The DBI component is injected into every launched application by a system 'starter' application (requires a firmware update). It requires root privileges/SELinux re-configuration. Intercepted events are passed to the RV monitor, which is loaded through JNI (A) as Dalvik bytecode, rendering all application and framework classes available for calling from policy rules (B). Rules take an `event|condition→action` form (inspired by [1]), where conditions distinguish between work/personal modes and actions prescribe execution resumption. All is captured in familiar Java/Android API syntax as per following 'Photo Capture' rule snippet:

```
wifi.ruleset.add(new Rule("Photo_capture"){
  public boolean condition(){
    if(wifi.ruleset_work_WIFI || wifi.ruleset_work_location) return true; else return false; }
  public void action() {
    wifi_ruleset.continue_exec = false; ShowToast("Access Denied"); } });
```

Experimentation BYOD-RV was implemented on Android 4.4 using the DDI toolkit.[3] The following policy rules have been successfully experimented with. Conditions: identification of the workplace wifi; workplace geolocation; and executing corporate apps. Application access-control actions: blocking photo captures and video/voice recording at the workplace. Application modification actions: restrict Internet access in work mode to a URL white-list; append a corporate signature at the end of all outgoing messages in work mode. The device events experimented with so far are the low battery and incoming call events, resulting in the termination of non-work applications for the prior and terminating calls in case of an ongoing video conference for the latter. Due to ahead-of-time compilation by `dex2oat` of all Dalvik bytecode to OAT files, porting to ART (`libart`) requires hooking Android methods at alternate locations (Fig. 1 - right). Patching native libraries (H) to intercept native library calls (E) made by the system OAT (compiled Android framework and core Java libraries) is one option, which however introduces a semantic gap challenge. Patching OAT files through pre-compilation bytecode patching (I) avoids this issue by intercepting Android method calls made by application OATs (F), but requires disabling OAT integrity checks. BYOD configuration is to be further simplified with a Domain-Specific Language (DSL).

References

[1] Colombo, C., Francalanza, A., Mizzi, R., Pace, G.J.: polyLarva: Runtime verification with configurable resource-aware monitoring boundaries. In: SEFM. LNCS, vol. 7504, pp. 218–232. Springer (2012)

[2] Falcone, Y., Currea, S., Jaber, M.: Runtime verification and enforcement for android applications with RV-Droid. In: RV. pp. 88–95. Springer (2013)

[3] Russello, G., Conti, M., Crispo, B., Fernandes, E.: MOSES: supporting operation modes on smartphones. In: SACMAT. pp. 3–12. ACM (2012)

[3] https://github.com/crmulliner/ddi