A Component-based Environment for Android Apps

Alexander Senier
FOSDEM, Brussels, 2020-02-02
Smartphone Trust Challenges

Privilege Escalation

- Stagefright (July 2015)
  - Specially crafted media data
  - Remote code execution, privilege escalation
- Problem not solved since:

![Graph showing media-related Android vulnerabilities](https://source.android.com/security/bulletin/)
Media Frameworks are not getting simpler. How do we avoid such fatal errors?
Trustworthy Systems  
Component-based Architectures

- Can’t reimplement everything
- **Solution: software reuse**  
  - Untrusted software (gray)  
  - Policy object (green)  
  - Client software (orange)
- **Policy object**  
  - Establishes assumptions of client  
  - Sanitizes  
  - Enforces additional policies

![Diagram of Network Stack with Protocol validator and Web browser](Diagram.png)
Trustworthy Systems

Information Flow: Genode OS Framework

- Recursive system structure
  - Root: Microkernel
  - Parent: Responsibility + control
  - Isolation is default
  - Strict communication policy

- Everything is a user-process
  - Application
  - File systems
  - Drivers, Network stacks

- Stay here for the next 2 talks for details (13:00)

https://genode.org
Trustworthy Systems
Correctness: SPARK

- **Programming Language**
  - Based on Ada
  - Compilable with GCC and LLVM
  - Customizable runtimes
  - Contracts (preconditions, postconditions, invariants)

- **Verification Toolset**
  - Absence of runtime errors
  - Functional correctness

- **Applications**
  - Avionics
  - Defense
  - Air Traffic Control
  - Space
  - Automotive
  - Medical Devices
  - Security

https://www.adacore.com/about-spark
Applying this Approach to Android Apps
GART Project

Objectives

- Unmodified Android Apps
- On top of Genode OS Framework
- Formally-verified policy objects
GART Project

Elements

- Build system
- Android Runtime
- Trusted Proxies
- IPC
Build System Integration
Build System Integration

Android Build Files

- **Soong** replaced old make-based build system in Oreo (8.0)
- JSON-like blueprint files
- Purely declarative – no conditionals, no control flow
- Complex cases handled in Go application
- Manifests for the **Ninja** build system are generated

```
cc_binary {
    name: "gzip",
    srcs: ["src/test/minigzip.c"],
    shared_libs: ["libz"],
    stl: "none",
}
```
Build System Integration
Translating Android Build Files

```
$ gnoos -b android -i libnativehelper/Android.bp -o libnativehelper.mk -p LIBNATIVEHELPER_ \
'/cc_library[@name=libnativehelper]'
$ cat libnativehelper.mk
LIBNATIVEHELPER_CFLAGS = -Werror -fvisibility=protected
LIBNATIVEHELPER_EXPORT_HEADER_LIB_HEADERS = jni_headers jni_platform_headers libnativehelper_header_only
LIBNATIVEHELPER_EXPORT_INCLUDE_DIRS = include
LIBNATIVEHELPER_HEADER_LIBS = jni_headers jni_platform_headers libnativehelper_header_only
LIBNATIVEHELPER_HOST_SUPPORTED = True
LIBNATIVEHELPER_NAME = libnativehelper
LIBNATIVEHELPER_SHARED_LIBS = liblog
LIBNATIVEHELPER_SRCS = JNIHelp.cpp JniConstants.cpp JniInvocation.cpp toStringArray.cpp
```
Gnoos has been integrated into Genode build system
Consistent with regular Genode applications
  - Library build files are in lib/mk
  - Library import files are in lib/import
  - Applications have a target.mk
Build System Integration
Integrating Android Build Files (2)

- Porting native Android applications is easy
- With run script, they can be run as every other Genode application
- gart_gtest_main makes porting Android tests a one-liner

```bash
$ make -C build/arm_v8a run/test/libutils
...
[init → libutils_test] Note: Google Test filter = -VectorTest.SetCapacity_Overflow:VectorTest._grow_OverflowSize
[init → libutils_test] :VectorTest._grow_OverflowCapacstr16EmptyTarget_bug:SystemClock.SystemClock
[init → libutils_test] [========] Running 61 tests from 9 test cases.
[init → libutils_test] [--------] Global test environment set-up.
[init → libutils_test] [--------] 3 tests from VectorTest
[init → libutils_test] [ RUN    ] VectorTest.CopyOnWrite_CopyAndAddElements
[init → libutils_test] [       OK ] VectorTest.CopyOnWrite_CopyAndAddElements (2 ms)
...
[init → libutils_test] [ PASSED ] 61 tests.
[init] child "libutils_test" exited with exit value 0
Run script execution successful.
make: Leaving directory 'build/arm_v8a'
```
Android Runtime
Android Runtime
Components

- **dalvikvm**
  - Only ~200 LOC
  - Linked with only few libraries – *libsigchain*, *libnativehelper*, *libc*
    - *libsigchain*: interception layer for signals
    - *libnativehelper*: helpers for Java/native interface

- **libart**
  - Actual Java Virtual Machine
  - Loaded dynamically by dalvikvm
  - > 50 dependencies that needed to be ported
Android Runtime
Current State

- dalvikvm and libart ported to Genode
  - The ~1000 tests cases of dependencies succeed
  - Most of the ~500 ART test cases succeed
  - Runtime fully initializes and starts Java program on Genode/arm_v8a (and then crashes, see below ;)

- Open issues
  - Concurrency bugs due to missing futex implementation
  - Probably some more due to differences between Genodes libc and Linux
  - Runtime-compiler basically ported, but still has issues

https://github.com/Componolit/gart
Trusted Proxies
Trusted Proxies
Component Environment

- **Downsized SPARK/Ada runtime**
  - Optimized for critical low-complexity components
  - No allocators, no exception handlers, no implicit dynamic code, no tasking...
  - Support for Genode, Muen and Linux
  - Easy to customize and port to new (embedded) environments

- **Gneiss component library**
  - Fully asynchronous, event-driven and platform-independent
  - Support for Genode, Muen and Linux
  - Only constructs that are formally verifiable with SPARK proof tools
  - Generic interfaces: Log, timer, block device, message, shared memory

See recordings of previous talk by Johannes Kliemann for details
Trusted Proxies
Verified Binary Parsers

- **RecordFlux**
  - DSL and toolset for formal specification of binary messages
  - Model verification (absence of contradictions, reachability, ...)
  - Generation of verifiable binary parsers
  - Generation of message generators

- See recording of Tobias Reihers talk for details (Saturday, 11:30, security devroom)

```plaintext
package TLV is
type Tag is (Msg_Data \( \Rightarrow 1 \),
             Msg_Error \( \Rightarrow 3 \)) with Size \( \Rightarrow 2 \);
type Length is mod 2**14;

type Message is
  message
  Tag    : Tag
          then Length
          if Tag = Msg_Data,
             then null
             if Tag = Msg_Error;
      Length : Length
              then Value
              with Length \( \Rightarrow \) Length * 8;
      Value  : Payload;
  end message;
end TLV;
```
Future Work / Next Up:
Android IPC
Android IPC
Binder device

- Linux device node /dev/[vnd|hw]\binder
  - Interaction through ioctl() interface
  - Blocking or non-blocking
  - Send and/or receive phase
- Data is passed via linked data structure
  - Local/remote objects with reference counting
  - Special objects
  - File-descriptor passing
  - Linux kernel copies between processes
  - Name-service application (ServiceManager)
Android IPC

Idea: User-level message broker

- Verified broker component to handle binder transactions
  - Android apps are client of broker using message passing
  - Clients share memory region with broker
  - Broker implements name service and copies between clients

- Pros
  - No additional complexity in the kernel
  - Enables filter components / policies

- Cons
  - At least 3 copies per transaction (Android kernel needs 1)
Conclusions

- Rehosting Android Runtime to Genode is feasible
- Easy porting due to declarative nature of Androids build system
- Google's extensive test suite is extremely helpful
- Environment for trustworthy formally verified filters exists

Future Work

- User-level binder IPC on Genode
- Porting or emulation of required Android services
- Integration into Genodes Nitpicker UI subsystem
- Trusted filters (e.g. encrypted / tagged calendar entries)
- Test complex, unmodified Android applications on Genode