TOMOYO Linux on Android

CE Linux Forum
Japan Technical Jamboree 27
2009/05/22

Giuseppe La Tona
giuseppelatona@gmail.com
About me...

• Master Degree in Computer Engineering
  University of Catania (Italy), 2008
  – Exchange student in Linkoping University (Sweden), 2007

• “Vulcanus in Japan” (Sep. 2008 – Aug. 2009)
  EU-Japan Centre for Industrial Cooperation
  – Industry-oriented student exchange programme
  – Scholarship offered by EU and Japanese METI
  – Japanese language intensive course, cultural activities
  – Internship in NTT Data Corporation (R&D)
    (January-August 2009)
    • Secure OS group, TOMOYO Linux project
  – Learning and experiencing Japanese culture, lifestyle, working way
Android and TOMOYO Linux

Android Robot is reproduced from work created and shared by Google and used according to terms described in the Creative Commons 2.5 Attribution License.
Android overview

- Full software stack for mobile devices
Linux Kernel 2.6 with some enhancements
- reduced set of standard Linux utilities (→ toolbox)
- no support for glibc (→ Bionic)
- no standard IPC (→ Binder, specific IPC driver)
- no native windowing system
- optimized Power Management
- Low memory killer, Alarm, Kernel Debugger, etc.

Android SDK 1.5 r1 (end of April 2009)
- released with Linux Kernel 2.6.27
- higher versions being developed (2.6.29 is ready)
Android from boot to user (1/2)

- `adb`d
- `vold` (mount)
- `ril`d (radio)
- `debuggerd`
- `installld`...

```
Daemons
  Binder
    Native Servers
      mediaserver
        service manager
          init
            Kernel
```

Runtime
Android Runtime: Zygote

• *Dalvik*: Virtual Machine for mobile devices
  – slow CPU, small RAM, no swap space, battery
  – Not a JVM, no JIT
  – Interprets DEX: optimized bytecode obtained from Java `.class`.
  – Multiple VM instances can run efficiently.

• Concept of *Zygote*:
  – first instance of Dalvik VM, partially initialized
  – preloads and pre-initializes classes
  – kept in an idle state by the system
  – when an *application execution* request occurs, zygote fork()s to a new process which loads the requested package ("zygote": duplicates, specializes and differentiates)
Android from boot to user (2/2)

Kernel

init

Daemons

Binder

Native Servers

adbd
vold (mount)
rild (radio)
debuggerd
installd
...

servicemanager

mediaserver

System Services

systemserver

Dalvik VM

Applications

Home

Dalvik VM

GUI

zygote

Runtime

fork()
Android security model

• Each application runs in its own process
  – Runtime in separate instances of Dalvik virtual machine

• Each process is a “secure sandbox”
  – Linux DAC for file access: all applications are assigned a unique UID which is maintained constant
    • UID for system services are hard-coded
    • UID for user packages are progressively assigned at install-time, starting from uid 10000 (and mapped to app_0, app_1, ...); they are saved in a file are maintained constant during the life of the package on the device.

• Application specific files are saved in /data/data in separate folders owned by specific UID users
Main points of porting TOMOYO Linux

- Patching Android kernel with TOMOYO Linux
- Adapting TOMOYO ccstools for embedded purposes
- Cross-compiling for Android
- Integrating TOMOYO Policy Loader in Android boot
- Analyzing Android processes using TOMOYO Linux
- Problem of Zygote “fork vs exec” approach
- Splitting security domains in Android runtime
• Patching Android kernel with TOMOYO Linux
• Adapting TOMOYO ccstools for embedded purposes
• Cross-compiling for Android
• Integrating TOMOYO Policy Loader in Android boot
• Analyzing Android processes using TOMOYO Linux
• Problem of Zygote “fork vs exec” approach
• Splitting security domains in Android runtime
Patching Android

• Device: emulator (no Android Dev Phone)

• Linux kernel version: Android Goldfish v2.6.29
  – “Goldfish” is the name of the ARM architecture emulated by Android SDK Emulator

• TOMOYO Linux 1.6.7 (non-LSM version)
  – ccs-patch 1.6.7 for kernel vanilla v2.6.29

• Minor rejects occurred
- Patching Android kernel with TOMOYO Linux
- **Adapting TOMOYO ccstools** for embedded purposes
- Cross-compiling for Android
- Integrating TOMOYO Policy Loader in Android boot
- Analyzing Android processes using TOMOYO Linux
- Problem of Zygote “fork vs exec” approach
- Splitting security domains in Android runtime
Adapting ccstools (1/2)

**ccstools**: set of utilities for TOMOYO Linux management

**Porting ccstools to TOMOYO Linux for embedded**

- Only few utilities are actually useful on the device: `loadpolicy`, `savepolicy`, `setprofile`, `ccstree`, `make_alias`.

*Other tools would also need porting C libraries missing in Android*

- Reducing the size and the complexity of these programs removing the unnecessary code (i.e. the code implementing network edit mode, offline edit mode)

- Introducing **editpolicy-agent** daemon to allow *network mode* (control of TOMOYO Linux via TCP)
Adapting ccstools (2/2)

Enhancing *ccstools* for normal TOMOYO Linux

- Using the policy editor tool (CUI) from the embedded device itself is not required.

- **Network mode:**
  
  ```
  $ ccs-editpolicy <IP>:<port>
  ```

- Edit policy from remote pc
  - Security admin *only* should be allowed to do it
  - User *may not* modify policies from device terminal
• Patching Android kernel with TOMOYO Linux
• Adapting TOMOYO ccstools for embedded purposes
• Cross-compiling for Android
• Integrating TOMOYO Policy Loader in Android boot
• Analyzing Android processes using TOMOYO Linux
• Problem of Zygote “fork vs exec” approach
• Splitting security domains in Android runtime
Cross-compiling for Android

• **C libraries used by Android: Bionic**
  – no glibc

• **Toolchain**
  – suite of cross-compilers for different architectures

• **agcc** (Perl script by Andrew Ross [http://plausible.org/andy/agcc](http://plausible.org/andy/agcc))
  – simple gcc-like front-end to compile C programs for Android
  – links Android libraries (needs Android source)
  – uses the appropriate cross-compiler from the Android Toolchain
• Patching Android kernel with TOMOYO Linux
• Adapting TOMOYO ccstools for embedded purposes
• Cross-compiling for Android
• Integrating TOMOYO Policy Loader in Android boot
• Analyzing Android processes using TOMOYO Linux
• Problem of Zygote “fork vs exec” approach
• Splitting security domains in Android runtime
Modifying Android boot

- Editing init.rc config file to execute
  - `/sbin/ccs-start`
  - `/sbin/ccs-editpolicy-agent` (with daemons)
- `ccs-start` triggers TOMOYO Linux
  → Policy Loader `/sbin/ccs-init` starts
• Patching Android kernel with TOMOYO Linux
• Adapting TOMOYO ccstools for embedded purposes
• Cross-compiling for Android
• Integrating TOMOYO Policy Loader in Android boot

• Analyzing Android processes using TOMOYO Linux
• Problem of Zygote “fork vs exec” approach
• Splitting security domains in Android runtime
Domain transition tree

```
<kernal> /system/bin/app_process
  0: 1  <kernal>
  1: 1  *     /init
  2: 1  /sbin/adbd
  3: 1    /system/bin/sh
  4: 1    /sbin/ccs-loadpolicy
  5: 1    /sbin/ccs-savepolicy
  6: 1    /sbin/ccs-setprofile
  7: 1    /sbin/init_policy.sh
  8: 1    /sbin/ccs-savepolicy
  9: 1    /sbin/make_exception
 10: 1    /system/bin/toolbox
 11: 1    /system/bin/ls
 12: 1    /system/bin/toolbox
 13: 1    /sbin/ccs-start
 14: 1    /system/bin/debuggerd
 15: 1    /system/bin/flash_image
 16: 1    /system/bin/installd
 17: 1    /system/bin/dexopt
 18: 1    /system/bin/logcat
 19: 1    /system/bin/mediasever
 20: 1    /system/bin/gemud
 21: 1    /system/bin/rild
 22: 1    /system/bin/servicemanager
 23: 1    /system/bin/sh
 24: 1    /system/bin/toolbox
 25: 1    /system/bin/vold
 26: 1    /system/bin/dofstck
 27: 1    /system/etc/init.goldfish.sh
 28: 1    /system/bin/getprop
 29: 1    /system/bin/setprop
 30: 1    /system/bin/toolbox
 31: 1  *    /sbin/ccs-editpolicy-agent
 32: 1  *    /system/bin/app_process
 33: 1    /system/bin/dexopt
```
Process tree

```
# ccs-ccstree
ccs-ccstree
1 init (1) <kernel> /init
  1 + sh (581) <kernel> /init /system/bin/sh
  1 + servicemanager (582) <kernel> /init /system/bin/servicemanager
  1 + vold (583) <kernel> /init /system/bin/vold
  1 + debuggerd (584) <kernel> /init /system/bin/debuggerd
  1 + rild (585) <kernel> /init /system/bin/rild
  1 + app_process (586) <kernel> /system/bin/app_process
     1 ++ app_process (618) <kernel> /system/bin/app_process
     1 ++ app_process (658) <kernel> /system/bin/app_process
     1 ++ app_process (663) <kernel> /system/bin/app_process
     1 ++ app_process (682) <kernel> /system/bin/app_process
     1 ++ app_process (700) <kernel> /system/bin/app_process
     1 ++ app_process (709) <kernel> /system/bin/app_process
  1 + mediaserver (587) <kernel> /init /system/bin/mediaserver
  1 + installd (588) <kernel> /init /system/bin/installd
  1 + ccs-editpolicy- (593) <kernel> /sbin/ccs-editpolicy-agent
  1 + qemu (595) <kernel> /init /system/bin/qemud
  1 + adb (597) <kernel> /init /sbin/adb
     1 ++ sh (739) <kernel> /init /sbin/adb /system/bin/sh
  1 + ccs-ccstree (741) <kernel> /init /sbin/adb /system/bin/sh /sbin/ccs-ccstree
```

ccs-ccstree: command showing process tree with relative security domains and profiles
**Process tree**

ccs-ccstree:

```
# ccs-ccstree
ccs-ccstree
  1 init (1) <kernel> /init
  1 ++ sh (581) <kernel> /init /system/bin/sh
  1 ++ servicemanager (582) <kernel> /init /system/bin/servicemanager
  1 ++ vold (583) <kernel> /init /system/bin/vold
  1 ++ debuggerd (584) <kernel> /init /system/bin/debuggerd
  1 ++ rild (585) <kernel> /init /system/bin/rild
  1 ++ app_process (586) <kernel> /system/bin/app_process
     1 ++ app_process (618) <kernel> /system/bin/app_process
     1 ++ app_process (658) <kernel> /system/bin/app_process
     1 ++ app_process (663) <kernel> /system/bin/app_process
     1 ++ app_process (682) <kernel> /system/bin/app_process
     1 ++ app_process (700) <kernel> /system/bin/app_process
     1 ++ app_process (709) <kernel> /system/bin/app_process
     1 ++ mediaserver (587) <kernel> /init /system/bin/mediaserver
        1 ++ instald (588) <kernel> /init /system/bin/instald
        1 ++ ccs-editpolicy- (593) <kernel> /sbin/ccs-editpolicy-agent
        1 ++ gemud (595) <kernel> /init /system/bin/gemud
        1 ++ adbd (597) <kernel> /init /sbin/adbd
           1 ++ sh (739) <kernel> /init /sbin/adbd /system/bin/sh
           1 ++ ccs-ccstree (741) <kernel> /init /sbin/adbd /system/bin/sh /sbin
```

- servicemanager
- Daemons
- mediaserver

---

service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-system-server

zygote
• Patching Android kernel with TOMOYO Linux
• Adapting TOMOYO ccstools for embedded purposes
• Cross-compiling for Android
• Integrating TOMOYO Policy Loader in Android boot
• Analyzing Android processes using TOMOYO Linux
• Problem of Zygote “fork vs exec” approach
• Splitting security domains in Android runtime
Problem of splitting domains

- The applications are executed with different UID (i.e.: root, system, app_#, ...) and different process name, but...

```
root  586  1  72420 20868 c009c184 afe0c584 S zygote
system  618  586  170384 26708 fffffffe afe0c584 S system_server
radio  658  586  107880 20944 fffffffe afe0d3e4 S com.android.phone
app_3  663  586  116120 24528 fffffffe afe0d3e4 S android.process.acore
app_9  682  586  95496 17104 fffffffe afe0d3e4 S com.android.mms
app_0  700  586  94284 16444 fffffffe afe0d3e4 S com.android.alarmclock
app_2  709  586  95396 17376 fffffffe afe0d3e4 S android.process.media
```

- ...they are all fork()ed instances of app_process!

```
ccs-ccstree:

1  +- app_process (586) <kernel> /system/bin/app_process
1  |   +- app_process (618) <kernel> /system/bin/app_process
1  |   |   +- app_process (658) <kernel> /system/bin/app_process
1  |   |   |   +- app_process (663) <kernel> /system/bin/app_process
1  |   |   |   |   +- app_process (682) <kernel> /system/bin/app_process
1  |   |   |   |   |   +- app_process (700) <kernel> /system/bin/app_process
1  |   |   |   |   |   |   +- app_process (709) <kernel> /system/bin/app_process
```
• Patching Android kernel with TOMOYO Linux
• Adapting TOMOYO ccstools for embedded purposes
• Cross-compiling for Android
• Integrating TOMOYO Policy Loader in Android boot
• Analyzing Android processes using TOMOYO Linux
• Problem of Zygote “fork vs exec” approach
• Splitting security domains in Android runtime
Problem of splitting domains

- In TOMOYO Linux, domain transitions occur after process invocation, that is after any execve() on a specific path name, not after fork().

- Impossible to split domain
  
  ```
  <kernel> /system/bin/app_process
  ```

  in different domains according to each single application.
Problem ofsplitting domains

Example: we want to allow the Browser to connect to Internet.

```
<kernel> /system/bin/app_process
134: allow_read /system/lib/hw/sensors.goldfish.so
135: allow_read /system/usr/keychars/qwerty.kcm.bin
136: allow_read /system/usr/keylayout/qwerty.kl
137: allow_read /system/usr/share/zoneinfo/zoneinfo.dat
138: allow_read /system/usr/share/zoneinfo/zoneinfo.idx
139: allow_read /system/usr/share/zoneinfo/zoneinfo.version
140: allow_network TCP bind 0.0.0.0 0
141: allow_network TCP connect 0.0.0.0-255.255.255.255 80
142: allow_network TCP connect 74.125.153.113 443
143: allow_network UDP bind 0.0.0.0 1024-65535
144: allow_network UDP connect 10.0.2.3 53
```

In this way any process running under “<kernel> /system/app/process” domain would be allowed to open TCP connection on any IP, port 80.

→ least-privilege principle violated
Solution: conditions on task UID

- TOMOYO Linux allows conditional ACL
- Using task UID as a condition

In this way **only** the task with UID=10001 (browser) will be able to connect
• Remote management of Android + TOMOYO Linux
  – ccstools
• app_process security policy
• Enforcing MAC with conditional ACL
  – browser, application data, ...
• Support for symbolic links
  – toolbox
  – makealias
Summary

- TOMOYO Linux successfully working on Android
- MAC enforced for system services and user applications
- Solution proposed with the minimum modifications in both Android and TOMOYO Linux
  - no changes in Android kernel source code, beside the TOMOYO patch
  - no changes in Android userland applications, except adding TOMOYO management tools
  - no changes in TOMOYO Linux patch
  - enhancements to TOMOYO Linux tools for embedded purposes
Scenarios

• MAC for any Android device
  – Protection for vulnerabilities
  – Effective containment of malicious attacks

• Function restriction
  – Limited network access
    (specific sites, applications)
  – Package Installing/Uninstalling ability

• Enterprise employee portable devices

• User smartphones, netbooks, ...
Future works

- code optimization
- performances measuring (benchmark)
  - evaluation
  - plan for optimization
- try to port TOMOYO Linux to other embedded Linux systems
Useful links

• Android source
  http://source.android.com/download

• Android kernel source
  http://android.git.kernel.org/?p=kernel/common.git;a=summary

• TOMOYO Linux patch and tools
  http://tomoyo.sourceforge.jp/en/1.6.x/compile.html#2.6-vanilla
  wget http://osdn.dl.sourceforge.jp/tomoyo/30298/ccs-tools-1.6.7-20090401.tar.gz
ご静聴どうもありがとうございました

Q&A