TOMOYO Linux on Android

CE Linux Forum
Japan Technical Jamboree 28
Osaka, 2009/06/12

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

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

• “Vulcanus in Japan” (Sep. 2008 – Aug. 2009)
  programme of the EU-Japan Centre for Industrial Cooperation
  – Industrial placement-oriented student exchange
  – Scholarship offered by EU and Japanese METI
  – Japanese language intensive course, cultural activities
  – Internship in NTT Data Corporation (R&D)
    (January-August 2009)

Learning and experiencing Japanese culture, lifestyle, working way!
About my internship...

- Secure OS group
  - TOMOYO Linux project

- R&D:
  - Study
    - Mandatory Access Control (MAC)
    - TOMOYO Linux
  - Porting TOMOYO Linux to Android platform
  - Analyze TOMOYO and MAC potentials on embedded Linux
• **MAC** implementation for Linux Operating Systems  
  – pathname-based approach

• It consists of:  
  – a kernel patch (*ccspatch*)  
  – a set of utilities (*ccstools*), for management and security policy editing

• TOMOYO Linux 2.2.0 has just been merged in Linux kernel 2.6.30 (10 June 2009)!
Android overview

• Full software stack for mobile devices
Android kernel

• Linux Kernel 2.6 with some changes
  – reduced set of standard Linux utilities (→ toolbox)
  – no support for glibc (→ Bionic libraries)
  – 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 r2 (May 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**
- vold (mount)
- rild (radio)
- debuggerd
- installd

... Binder

- Daemons
- servicemanager
- mediaserver

Native Servers

init

Kernel

Runtime

zygote
Android Runtime: Dalvik and Zygote

• Runtime is made by **Java programs running in Dalvik**: Virtual Machine for mobile devices
  – slow CPU, small RAM, no swap space, battery
  – Not a JVM, no JIT: only interpreter of DEX (optimized bytecode obtained from Java .class)
  – Multiple VM instances can run efficiently.

• **Zygote process:**
  – first instance of Dalvik VM, partially initialized
  – load *preload* classes and resources
  – is kept always alive in idle state

When an application execution request occurs:
  – zygote `fork()`s to a new process...
    • ...which loads the requested package

(Biology concept of “zygote”: duplicate, specialize and differentiate)
Android from boot to user (2/2)

- adb
- vold (mount)
- rild (radio)
- debuggerd
- installd

...Binder

Daemons

Native Servers

servicemanager

mediaserver

init

Kernel

System Services

Dalvik VM

Applications

Dalvik VM

GUI

exec()

fork()

Dalvik specialization
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 Discretionary Access Control (DAC) for file access: all applications are assigned a unique UID (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
Porting TOMOYO Linux to Android

• Patching Android kernel with TOMOYO patch
• Adapting TOMOYO ccstools for embedded purposes
• Cross-compiling for Android
• Integrating TOMOYO Policy Loader in Android boot
• TOMOYO policy files location
There are 2 main development lines:

- **non-LSM (versions 1.6.x)**
  - provides full functionalities of pathname-based MAC (MAC for files, network, capabilities...)

- **mainline (version 2.2.0)**
  - uses Linux Security Modules (LSM)
    - necessary hooks available from Linux kernel 2.6.29
  - subset of MAC functionalities (only for files, so far)
    - currently developing the others to use LSM
Patching Android kernel

- TOMOYO Linux 1.6.8 (non-LSM version)

- Emulator (no real Android device available)
  - Linux kernel version: Goldfish v2.6.29
    - “Goldfish” is the given to the ARM architecture emulated by Android SDK Emulator

- ccspatch 1.6.8 (2009/05/28) for kernel vanilla v2.6.29
Adapting ccstools (1/2)

Since version 1.6.7, ccstools has been enhanced with **Network mode**, to support editing policy via TCP connection

$ ccs-editpolicy <IP>:<port>

In the case of embedded systems, this is more convenient for developing policies and debugging.

Using the policy editor from the embedded device is generally not required.
Only few utilities are actually useful on the device: loadpolicy, savepolicy, setprofile, ccstree, make_alias. 

Other tools would also need porting C libraries missing 

ccstools version for Android:

• Reduced the size and the complexity of these utilities removing the unnecessary code

• Introducing editpolicy-agent daemon to allow network mode communication

This ccstools version could actually be suitable for other embedded Linux systems as well.
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
Modifying Android boot

- Daemons
- Servicemanager
- Mediaserver
- Native Servers
- Binder
- Init
- Zygote
- Kernel
Modifying Android boot

It is required in order to:

- **start TOMOYO Policy Loader** `/sbin/ccs-init`, via `/sbin/ccs-start (*)`
- **launch agent for editing policy remotely**
  `/sbin/ccs-editpolicy-agent`

```
<table>
<thead>
<tr>
<th>Policy</th>
<th>Loader</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>/sbin/ccs-init</td>
<td></td>
</tr>
</tbody>
</table>

Daemons

```

TOMOYO Linux

Kernel

zygote
mediaserver
servicemanager

```

init

```

3

2

*
• `/data/ccs/` (at the moment)

• `/data` is...
  – ...a readwrite partition
  – ...allowed to be empty at Android boot time
  – ...supposed to be wiped off if Android device is reset to factory default state

Not a safe place for security policy files?
  – a reset would delete them!
A possible solution: **split policy files for**...

- **prebuilt firmware applications**
  - in `/system` readonly partition (i.e. `/system/ccs`)

- **other applications downloaded.installed later**
  - in `/data` readwrite partition (i.e. `/data/ccs`)
    - application-specific policies could be saved at application install-time, in `/data`
    - any eventual reset will wipe off those policies, but the related applications as well.
• Analyzing Android processes and domains

• Problem of Zygote “fork vs exec” approach

• Splitting domains in Android runtime

• Enforcing policy
Domain transition tree

```
<kernel> /system/bin/app_process

0: 1  <kernel>
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/install
17: 1  /system/bin/dexopt
18: 1  /system/bin/logcat
```
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 - installd (588) <kernel> /init /system/bin/installd
      1 - ccs-editpolicy- (593) <kernel> /sbin/ccs-editpolicy-agent
      1 - qemu (595) <kernel> /init /system/bin/qemu
      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
/ccs-ccstree
```

- servicemanager
- Daemons
- mediaserver
- zygote

Service:
```
service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-system-server
```
Problem of splitting domains

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

```
ps:
root  586 1  72420 20868 c0094184 afe0c584 S zygote
system 618 586 178840 26708 111111111 afe0c45c S system_server
radio  658 586 107880 20944 111111111 afe0d3e4 S com.android.phone
app_3  663 586 116120 24528 111111111 afe0d3e4 S android.process.acore
app_9  682 586 95496 17104 111111111 afe0d3e4 S com.android.mms
app_0  700 586 94284 16444 111111111 afe0d3e4 S com.android.alarmclock
app_2  709 586 95396 17876 111111111 afe0d3e4 S android.process.media
```

...they are all fork()ed from app_process!

ccs-ccstree:
```
1  +- app_process (586) <kernel> /system/bin/app_process
1   1  +- app_process (618) <kernel> /system/bin/app_process
1   1  +- app_process (658) <kernel> /system/bin/app_process
1   1  +- app_process (663) <kernel> /system/bin/app_process
1   1  +- app_process (682) <kernel> /system/bin/app_process
1   1  +- app_process (700) <kernel> /system/bin/app_process
1   1  +- app_process (709) <kernel> /system/bin/app_process
```
Problem of splitting domains

- New and unexpected situation for TOMOYO Linux

- In TOMOYO Linux, domain transitions occur after process invocation, that is **execve()**, not **fork()**

→ Splitting domain

`<kernel> /system/bin/app_process`

in different domains according to each single application is impossible...?
Problem of splitting domains

<kernel> /system/bin/app_process

0: 1 <kernel>
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/initpolicy.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/installid
17: 1 /system/bin/DEXOPT
18: 1 /system/bin/logcat
19: 1 /system/bin/midaserver
20: 1 /system/bin/qemud
21: 1 /system/bin/rild
22: 1 /system/bin/service_manager
23: 1 /system/bin/sh
24: 1 /system/bin/toolbox
25: 1 /system/bin/vold
26: 1 /system/bin/dofsck
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
An example

We want to allow the Browser to connect to Internet.

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

- TOMOYO Linux allows conditional ACL
- **Using task UID as a condition**, for access grant.

```
app_1    811    586     132256   31548   ffffffff afe03e4   S   com.android.browser
```

In this way **only** the task with UID=10001 (browser) will be able to connect
• Android security rule: data files of one application should be prevented from being accessed by other applications

• This is performed by using DAC permissions, as said before

• TOMOYO can provide with conditional ACL a further insurance that this rule is respected, especially in cases when:
  – DAC permissions are poorly configured
  – root process (zygote) would be hijacked

```
allow_read/write @APP_DATA_FILE if task.uid=path1.uid
allow_unlink @APP_DATA_FILE if task.uid=path1.uid
allow_mkdir @APP_DATA_DIR if task.uid=path1.parent.uid1
```
• DAC’s ability to restrict by UID has a low granularity: only “owner”, “group”, “others”.

• TOMOYO, on the other hand, allows minimal and customizable permissions to any group of specific UIDs.

• Example: users are app_1, app_2, app_3, app_4; some files owned by app_2 (uid=10002) need to be accessed by app_1 (uid=10001) also, but not by all the “others”.

allow_read/write @SOME_FILES if task.uid=10001-10002
• Remote management of Android + TOMOYO Linux
  – ccstools
• “<kernel> /system/bin/app_process” domain policy
• Enforcing MAC with conditional ACL
  – browser, application data, ...
• Support for symbolic links
  – toolbox
  – makealias
### Some numbers

<table>
<thead>
<tr>
<th></th>
<th>Android kernel</th>
<th>Android kernel + TOMOYO 1.6.8</th>
<th>impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel zImage size</td>
<td>1,397,200 Bytes</td>
<td>1,440,616 Bytes</td>
<td>about 43 KB</td>
</tr>
<tr>
<td>Kernel memory usage (total=96MB)</td>
<td>93856 KB available after boot</td>
<td>93772 KB available after boot</td>
<td>84 KB</td>
</tr>
<tr>
<td>Policy handling memory usage</td>
<td>-</td>
<td>32768 Bytes (with access logs 221952 Bytes)</td>
<td>about 33 KB (with access logs about 222 KB)</td>
</tr>
</tbody>
</table>

- **TO DO: Benchmark!**
  to measure all the overhead caused by TOMOYO Linux
Some possible scenarios

• MAC for Android devices
  – 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, ...
Conclusions

- 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
Conclusions

• Ideas and plans for future works
  – Benchmarking
  – Porting and testing TOMOYO Linux to other embedded Linux systems

• Call For Collaboration!

Discuss and confronting with you is essential to improve and develop new ideas. We need your help 😊

We would be glad to receive any form of feedback: questions, comments, critics, proposals!
Useful links

• TOMOYO Linux
  – compile & install manuals:
    http://tomoyo.sourceforge.jp/en/1.6.x/compile.html#2.6-vanilla
  – Mailing list:
    (japanese) http://lists.sourceforge.jp/mailman/listinfo/tomoyo-users
  – @ Sourceforge: http://tomoyo.sourceforge.jp/index.html.en

• Android
  – source: http://source.android.com/download
  – kernel: http://android.git.kernel.org/?p=kernel/common.git;a=summary
Thank you for listening
ご静聴どうもありがとうございました

TOMOYO is a registered trademark of NTT DATA CORPORATION in Japan. Linux is a trademark of Linus Torvalds. Other names and trademarks are the property of their respective owners.