BIGMAC: Fine-Grained Policy Analysis of Android Firmware

Grant Hernandez †, Dave (Jing) Tian ‡, Anurag Swarnim Yadav †, Byron J. Williams †, and Kevin R. B. Butler †

† — University of Florida  ‡ — Purdue University
Android Attack-surface
High Impact Bugs

Android Is The Most Vulnerable Operating System
Number of vulnerabilities by operating system in 2016*

- Android: 523
- Debian Linux: 319
- Ubuntu Linux: 278
- Linux Kernel: 217
- Mac OS X: 215
- Windows 10: 172
- iPhone OS: 161
- Windows 8.1: 154

* Vulnerability defined as a mistake in software that can be directly used by a hacker to gain access to a system/network

Source: CVE Details
High Impact Bugs

• **CVE-2017-0737** - libstagefright (remote MMS triggerable)
High Impact Bugs

- **CVE-2017-0737** - libstagefright (remote MMS triggerable)

- **CVE-2018-9488** - Privilege escalation to full root compromise (USB)
High Impact Bugs

- **CVE-2017-0737** - libstagefright (remote MMS triggerable)
- **CVE-2018-9488** - Privilege escalation to full root compromise (USB)
- **CVE-2019-2215** - Binder Use After Free (app reachable)
Android Security Mechanisms

- SECCOMP
- DAC
- Capabilities
- SELinux
- Kernel Objects

Apps
Native Daemons & Privileged Apps
Android Security Mechanisms

- Primary Access Control
  - Linux DAC
  - Linux Capabilities
  - SELinux / SEAndroid (MAC)
Android Security Mechanisms

- **Primary Access Control**
  - Linux DAC
  - Linux Capabilities
  - SELinux / SEAndroid (MAC)

- **Other**
  - SECCOMP
  - Android Middleware
Android Security Mechanisms

- **Primary Access Control**
  - Linux DAC
  - Linux Capabilities
  - SELinux / SEAndroid (MAC)

- **Other**
  - SECCOMP
  - Android Middleware
Android Security Core Files

- SELinux for properties
- Assigns a SELinux label to files
- Boots system, manages props
  
  - property_contexts
  - file_contexts
  - seapp_contexts
  - init
  - /etc/services/*.rc
  - seapp_contexts
  - ueventd
  - /etc/services/*rc
  - service_contexts
  - servicemanager
  - ueventd.rc
  - system_server
  - mac_permissions.xml

Binary SELinux Policy

Android boot commands

SELinux for properties
Assigns a SELinux label to files
Boots system, manages props

Florida Institute for Cybersecurity (FICS) Research
Android Security Core Files

- **Binary SELinux Policy**
- **Android boot commands**
- **Assigns a SELinux label to files**
- **Assigns DAC context to /dev files**

**SELinux for properties**
- Assigns a SELinux label to files

**Boots system, manages props**
- property_contexts
- init
- file_contexts
- init.rc
- /etc/services/*.rc

**Creates device files and perms**
- seapp_contexts
- zygote
- ueventd
- service_contexts
- system_server
- mac_permissions.xml

**Android boot commands**
- Boots system, manages props
- Creates device files and perms

**Object Read-by**

**Process Relation**
Android Security Core Files

- **SELinux for properties**
- Assigns a SELinux label to files
- Assigns a SELinux label to apps
- All Java apps are forked from here

**Binary SELinux Policy**
- Boots system, manages props
- Creates device files and perms
- Assigns DAC context to /dev files
- Android boot commands

- SELinux for properties
- Assigns a SELinux label to files
- Assigns a SELinux label to apps
- All Java apps are forked from here

- Binary SELinux Policy
- Boots system, manages props
- Creates device files and perms
- Assigns DAC context to /dev files
- Android boot commands

- Kernel
- /etc/services/*.rc
- ueventd.rc
- mac_permissions.xml

- /etc/services/*.rc
- ueventd.rc
- mac_permissions.xml

- Septet
- Object Read-by
- Process Relation

- ueventd
- servicemanager
- system_server
- zygote

- property_contexts
- file_contexts
- init
- init.rc
- sepolicy
- seapp_contexts
- service_contexts
- uевентд
- ueventd.rc
- mac_permissions.xml
Android Security Core Files

- **Binary SELinux Policy**
- **Assigns a SELinux label to apps**
- **Assigns a SELinux label to files**
- **Assigns DAC context to /dev files**
- **Assigns labels to signed apps**
- **SELinux for properties**
- **Assigns labels to signed apps**
- **Boots system, manages props**
- **Creates device files and perms**
- **Mediates App IPC with Android perms**
- **All Java apps are forked from here**
- **Boots system, manages props**
- **creates device files and perms**
- **Kernel**
- **Sepolicy**
- **Property contexts**
- **File contexts**
- **Init**
- **Init.rc**
- **Etc/services/*.rc**
- **Service contexts**
- **Seapp contexts**
- **Zygote**
- **Ueventd**
- **Ueventd.rc**
- **Servicemanager**
- **System_server**
- **Mac_permissions.xml**
- **Object Read-by**
- **Process Relation**

**Florida Institute for Cybersecurity (FICS) Research**

5
Android Security Core Files

- **SELinux for properties**
- **Assigns a SELinux label to files**
- **Assigns a SELinux label to apps**
- **Mediates App IPC with Android perms**
- **Assigns labels to signed apps**

### System Components

- **Kernel**
- **Sepolicy**
- **Property contexts**
- **File contexts**
- **Seapp contexts**
- **Init**
- **Init.rc**
- **Etc/services/*.rc**
- **Ueventd**
- **Zygote**
- **System_server**
- **Ueventd.rc**
- **Mac_permissions.xml**
- **Service contexts**
- **Servicemanager**

### SELinux Policies

- **Binary SELinux Policy**
- **Android boot commands**
- **Assigns SELinux label to services**
- **Assigns DAC context to /dev files**

### SELinux Assignments

- Assigns a SELinux label to apps
- Assigns a SELinux label to files
- Assigns labels to signed apps
- Assigns DAC context to /dev files
- Assigns SELinux label to services
- Assigns SELinux policy to properties
- SELinux Policy boots system, manages props
- SELinux Policy creates device files and permissions
- SELinux Policy manages native services and their IPC

### Key Points

- All Java apps are forked from here
- Mediates App IPC with Android perms
- Boots system, manages props
- Creates device files and permissions
- Manages native services and their IPC

---

Florida Institute for Cybersecurity (FICS) Research
BigMAC at a High Level

- Maps MAC+DAC+CAP policies onto a fine-grained attack-graph
- Only considers running processes and present files
BigMAC at a High Level

- Maps MAC+DAC+CAP policies onto a fine-grained attack-graph
- Only considers running processes and present files
BigMAC at a High Level

- Maps MAC+DAC+CAP policies onto a fine-grained attack-graph
- Only considers running processes and present files
BigMAC at a High Level

- Maps MAC+DAC+CAP policies onto a fine-grained attack-graph
- Only considers running processes and present files
BigMAC at a High Level

- Maps MAC+DAC+CAP policies onto a fine-grained attack-graph
- Only considers running processes and present files
BigMAC at a High Level

- Maps MAC+DAC+CAP policies onto a fine-grained attack-graph
- Only considers running processes and present files
Building an Attack-Graph

Subject

Dataflow

Flat Dataflow

Objects

Process Tree

Instantiate

Fully Instantiated Attack Graph

SEPolicy

Parse

Subject

Gd

Instantiate

Gf

Overlay

Gs

Extract

Gp

Subjects

Processes

Ga

Backing Files

O

Tag/Link

Recover

Instantiate

F

S

P

G

D

G

S

G

P

O

g

F

S

P
Building an Attack-Graph

- **Subject**
  - **Gs** (SEPolicy)
  - **F (Backing Files)**
  - **Parse**

- **Dataflow**
  - **Gd** (Objects)
  - **Tag/Link**
  - **Recover**

- **Objects**
  - **Gf** (Fully Instantiated Attack Graph)
  - **Overlay**

- **Process Tree**
  - **Gp** (Subjects)
  - **Extract**
  - **Instantiate**

- **Processes**
  - **P**
  - **Processes**

- **Flat Dataflow**
  - **O** (Objects)
  - **Instantiated**

- **Overlay**
  - **Ga** (Objects)
Building an Attack-Graph

**Subject**
- Parse
- Extract
- Tag/Link
- Recover

**Processes**
- Instantiate
- Overlay
- Process Tree
- Subjects
- Processes

**Objects**
- Instantiate
- Fully Instantiated
- Attack Graph

**Backing Files**
- SEPolicy
- G_F
- G_S
- Flat Dataflow

**Processes**
- G_P
- G_D
- G_O
- G_a
Building an Attack-Graph

- SEPolicy
  - Parse
  - Tag/Link
  - Extract
- Subject
  - Recover
- Backing Files
- Objects
  - Instantiate
- Flat Dataflow
- Fully Instantiated Attack Graph
- Process Tree
  - Instantiate
- Subjects
- Processes
- Overlay
Building an Attack-Graph

Subject

Dataflow

Flat Dataflow

Process Tree

Instantiate

Recover

Instantiate

Overlay

Fully Instantiated Attack Graph

Backing Files

SEPolicy

Parse

Tag/Link

Extract

Objects

Ga

Ga

Process

Processes

Gf

Gp

S

F

Gs

Gd

Subject

Objects

Flat Dataflow

Instantiation

Recover

Instantiation

Overlay
Building an Attack-Graph

- **Subject**
  - **Gs**
    - Parse
    - Extract
    - Backing Files

- **Process Tree**
  - **Gp**
    - Instantiate
    - Recover
    - Subjects

- **Objects**
  - **Gd**
    - Instantiate
    - Tag/Link
    - Flat Dataflow

- **Overlay**
  - Fully Instantiated Attack Graph

- **Backing Files**
  - **F**

- **Processes**
  - **P**

- **SEPolicy**

- **Dataflow**

- **Overlay**
Building an Attack-Graph

- Extract
- Parse
- Recover
- Tag/Link
- Instantiate
- Overlay

Subject: $G_s$
Objects: $O$
Processes: $P$
Backing Files: $F$
SEPolicy: $SEPolicy$
Flat Dataflow
Fully Instantiated Attack Graph
Building an Attack-Graph

Subject

Dataflow

Flat Dataflow

Process Tree

Instantiate

Overlay

Fully Instantiated Attack Graph

Objects

SEPolicy

F

S

G

Gd

Gf

Ga

Gp

Processes

Subjects

Backing Files

Parse

Tag/Link

Extract

Recover

Instantiate

Overlay
Building an Attack-Graph

- **Subject (Gs)**
  - Parse
  - Extract
  - Backing Files

- **Processes (P)**
  - Subjects
  - Processes

- **Dataflow**
  - Instantiate
  - Recover
  - Tag/Link

- **Objects (O)**
  - Instantiate

- **Flat Dataflow**
  - Fully Instantiated Attack Graph

- **SEPolicy**
Building an Attack-Graph

Subject

Dataflow

Flat Dataflow

Process Tree

Instantiate

Instantiate

Overlay

Fully Instantiated Attack Graph

SEPolicy

Parse

Tag/Link

Recover

Extract

Backing Files

Gd

Gf

Gp

Ga

Objects

Processes

Subjects

Backing Files

Processes

Subjects
Processes Recovery
Processes Recovery

• We want to know what objects processes can access based upon the system policy
Processes Recovery

• We want to know what objects processes can access based upon the system policy
  • This is based upon their permissions (UID, GID, label, capabilities)
Processes Recovery

• We want to know what objects processes can access based upon the system policy
  • This is based upon their permissions (UID, GID, label, capabilities)

• **We have no processes in static firmware!**
Processes Recovery

• We want to know what objects processes can access based upon the system policy
  • This is based upon their permissions (UID, GID, label, capabilities)

• **We have no processes in static firmware!**
  • Can we recover processes and their credentials just from firmware?
Emulating Android’s Boot

- property_contexts
- file_contexts
- init
- init.rc
- /etc/services/*.rc

Boots system, manages props
Emulating Android’s Boot

• Android’s boot process is well-specified by the platform

![Diagram showing the boot process of Android with nodes for property_contexts, file_contexts, init, init.rc, and /etc/services/*.rc]
Emulating Android’s Boot

- Android’s boot process is well-specified by the platform
- `init.rc` files are loaded describing services, or native daemons
Emulating Android’s Boot

- Android’s boot process is well-specified by the platform
- **Init.rc** files are loaded describing services, or native daemons
- Explicit credential assignment for services
Emulating Android’s Boot

• Android’s boot process is well-specified by the platform
• `init.rc` files are loaded describing services, or native daemons
• Explicit credential assignment for services
• Allows the capture of boot-time changes to the filesystem
Emulating Android’s Boot

• Android’s boot process is well-specified by the platform
• `init.rc` files are loaded describing services, or native daemons
• Explicit credential assignment for services
• Allows the capture of boot-time changes to the filesystem
• Without incorporating this, cross-vendor analysis doesn’t scale and accuracy suffers
Evaluation of BigMAC

• **Ground Truth Evaluation**
  • How does BigMAC recovery compare to extracting security policies from a running device?

• **Attack Surface Case Studies**
  • Evaluation of our Prolog query engine to discover attack paths from and to critical Android components
# Ground-truth Evaluation (Files)

<table>
<thead>
<tr>
<th></th>
<th>Samsung S7 Edge (7.0.0)</th>
<th>Pixel 1 (7.1.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>Count</td>
<td>%Files</td>
</tr>
<tr>
<td>Correct Files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/system</td>
<td>5,233</td>
<td>93.1%</td>
</tr>
<tr>
<td>/data</td>
<td>115</td>
<td>2.0%</td>
</tr>
<tr>
<td>/dev</td>
<td>40</td>
<td>0.7%</td>
</tr>
<tr>
<td>Different DAC/MAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/dev</td>
<td>46</td>
<td>0.8%</td>
</tr>
<tr>
<td>/mnt</td>
<td>7</td>
<td>0.1%</td>
</tr>
<tr>
<td>/system</td>
<td>5</td>
<td>0.1%</td>
</tr>
<tr>
<td>Extra Files</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/dev</td>
<td>73</td>
<td>1.3%</td>
</tr>
<tr>
<td>/system</td>
<td>6</td>
<td>0.1%</td>
</tr>
<tr>
<td>/acct</td>
<td>1</td>
<td>0.0%</td>
</tr>
<tr>
<td>Total:</td>
<td>5,621</td>
<td>100%</td>
</tr>
<tr>
<td>DAC/MAC Correct:</td>
<td>98.7%</td>
<td></td>
</tr>
</tbody>
</table>
**Ground-truth Evaluation (Files)**

<table>
<thead>
<tr>
<th>Path</th>
<th>Count</th>
<th>%Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>/system</td>
<td>5,233</td>
<td>93.1%</td>
</tr>
<tr>
<td>/data</td>
<td>115</td>
<td>2.0%</td>
</tr>
<tr>
<td>/dev</td>
<td>40</td>
<td>0.7%</td>
</tr>
<tr>
<td>/mnt</td>
<td>7</td>
<td>0.1%</td>
</tr>
<tr>
<td>/system</td>
<td>5</td>
<td>0.1%</td>
</tr>
<tr>
<td>/acct</td>
<td>1</td>
<td>0.0%</td>
</tr>
<tr>
<td>/dev</td>
<td>73</td>
<td>1.3%</td>
</tr>
<tr>
<td>/data</td>
<td>115</td>
<td>3.4%</td>
</tr>
<tr>
<td>/dev</td>
<td>28</td>
<td>0.8%</td>
</tr>
<tr>
<td>/mnt</td>
<td>2</td>
<td>0.1%</td>
</tr>
<tr>
<td>/mnt</td>
<td>5</td>
<td>0.1%</td>
</tr>
<tr>
<td>/dev</td>
<td>167</td>
<td>4.9%</td>
</tr>
<tr>
<td>/cache</td>
<td>4</td>
<td>0.1%</td>
</tr>
<tr>
<td>/acct</td>
<td>1</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

**Samsung S7 Edge (7.0.0)**

**Pixel 1 (7.1.2)**

<table>
<thead>
<tr>
<th>Path</th>
<th>Count</th>
<th>%Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>/system</td>
<td>2,301</td>
<td>67.6%</td>
</tr>
<tr>
<td>/vendor</td>
<td>630</td>
<td>18.5%</td>
</tr>
<tr>
<td>/data</td>
<td>115</td>
<td>3.4%</td>
</tr>
<tr>
<td>/dev</td>
<td>28</td>
<td>0.8%</td>
</tr>
<tr>
<td>/mnt</td>
<td>5</td>
<td>0.1%</td>
</tr>
<tr>
<td>/mnt</td>
<td>2</td>
<td>0.1%</td>
</tr>
<tr>
<td>/dev</td>
<td>167</td>
<td>4.9%</td>
</tr>
<tr>
<td>/cache</td>
<td>4</td>
<td>0.1%</td>
</tr>
<tr>
<td>/acct</td>
<td>1</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

**Total:** 5,621 100%

**Total:** 3,405 100%

**DAC/MAC Correct:** 98.7%

**DAC/MAC Correct:** 98.6%

Our recovered file metadata is 98% accurate to an equivalent running device.
Ground-truth Evaluation (Proc.)

(a) Processes Recovered by BiGMAC

(b) Actual device processes
(a) Processes Recovered by BiGMAC

(b) Actual device processes

Of the paired processes, we achieve, at best, 74.7% accuracy of process credentials.
We developed a Prolog query engine to find attack-paths with MAC, DAC, CAP, and external attack surface filtering.

```prolog
query_mac(S, T, C, P).
query_mac_dac(S, T, C, P).
query_mac_dac_cap(S, T, C, B, P).
query_mac_dac_cap_ext(S, T, C, B, E, P).
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Starting node</td>
</tr>
<tr>
<td>T</td>
<td>Target node</td>
</tr>
<tr>
<td>C</td>
<td>Path cutoff</td>
</tr>
<tr>
<td>B</td>
<td>Linux capability</td>
</tr>
<tr>
<td>E</td>
<td>External interface</td>
</tr>
<tr>
<td>P</td>
<td>Returned paths</td>
</tr>
</tbody>
</table>
We developed a Prolog query engine to find attack-paths with MAC, DAC, CAP, and external attack surface filtering.

query_mac(S, T, C, P).
query_mac_dac(S, T, C, P).
query_mac_dac_cap(S, T, C, B, P).
query_mac_dac_cap_ext(S, T, C, B, E, P).

We ran queries against a 1.3 million edge Samsung S8+ and a ~2 million edge LG G7 image.

S – Starting node  B – Linux capability
T – Target node    E – External interface
C – Path cutoff    P – Returned paths
Layered Path Reduction

\texttt{query\_mac}(\texttt{untrusted\_app,mediaserver,4,P}).
\texttt{query\_mac\_dac}(\texttt{untrusted\_app,mediaserver,4,P}).

<table>
<thead>
<tr>
<th>#Paths</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>102,915</td>
<td>22.48</td>
</tr>
<tr>
<td>5,146</td>
<td>518.62</td>
</tr>
</tbody>
</table>
Layered Path Reduction

query_mac(untrusted_app,mediaserver,4,P).
query_mac_dac(untrusted_app,mediaserver,4,P).

<table>
<thead>
<tr>
<th>#Paths</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>102,915</td>
<td>22.48</td>
</tr>
<tr>
<td>5,146</td>
<td>518.62</td>
</tr>
</tbody>
</table>

Each additional layer reduces the number of possible paths.

MAC to MAC+DAC has a 20x reduction in the number of paths to be considered.
query_mac_dac(init,_,1,P).
query_mac_dac(system_server,_,1,P).
query_mac_dac(lpm,_,1,P).

query_mac_dac(init,_,1,P).
query_mac_dac(system_server,_,1,P).
query_mac_dac(hal_usb,_,1,P).

<table>
<thead>
<tr>
<th>Image</th>
<th>Process</th>
<th># Writable</th>
<th># IPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung S8+</td>
<td>init</td>
<td>2,066</td>
<td>296</td>
</tr>
<tr>
<td></td>
<td>system_server</td>
<td>1,398</td>
<td>458</td>
</tr>
<tr>
<td></td>
<td>lpm</td>
<td>634</td>
<td>8</td>
</tr>
<tr>
<td>LG G7</td>
<td>init</td>
<td>1,233</td>
<td>418</td>
</tr>
<tr>
<td></td>
<td>system_server</td>
<td>573</td>
<td>368</td>
</tr>
<tr>
<td></td>
<td>hal_usb_default</td>
<td>508</td>
<td>19</td>
</tr>
</tbody>
</table>
query_mac_dac(init,_,1,P).
query_mac_dac(system_server,_,1,P).
query_mac_dac(lpm,_,1,P).

query_mac_dac(init,_,1,P).
query_mac_dac(system_server,_,1,P).
query_mac_dac(hal_usb,_,1,P).

Some of the most powerful processes (system_server) on Android deal with some of the most untrusted data.
Some of the most powerful processes (system_server) on Android deal with some of the most untrusted data.

system_server should be refactored into smaller, less privileged processes, similar to mediaserver.
Privilege Escalation Analysis

#1 \texttt{query\_mac\_dac}(zygote, vold, 3, P).

\begin{tikzpicture}
  \node[draw, rectangle, red] (zygote) at (0,0) {process:zygote};
  \node[draw, rectangle, white, text=red] (var_files) at (3,0) {<various\_files>};
  \node[draw, rectangle, red] (crash_dump) at (0,-1) {process:crash\_dump};
  \node[draw, rectangle, red] (vold) at (3,-1) {process:vold};
  \node[draw, rectangle, white] (ptrace) at (3,-2) {vold:ptrace};
  \node[draw, rectangle, white] (transition) at (-1,-2) {crash\_dump\_exec:transition};
  \draw[-stealth] (zygote) -- node[above] {*:write} (var_files);
  \draw[-stealth] (crash_dump) -- (transition);
  \draw[-stealth] (transition) -- (vold);
  \draw[-stealth] (vold) -- (ptrace);
  \node[draw=green, rounded corners, text=white] at (2,-2) {CVE-2018-9488};
\end{tikzpicture}
#1 \texttt{query\_mac\_dac(zygote,vold,3,P)}.

\begin{itemize}
  \item \texttt{process:zygote}<CAP\_SYS\_ADMIN> \rightarrow *:write \rightarrow \texttt{<various\_files>}
  \item \texttt{crash\_dump\_exec:transition}
  \item \texttt{process:crash\_dump} \rightarrow \texttt{vold:ptrace} \rightarrow \texttt{process:vold}\texttt{<uid=0>}
\end{itemize}

\textbf{CVE-2018-9488}

#2 \texttt{query\_mac\_dac\_cap(\_\_\_,crash\_dump,1,CAP\_SYS\_ADMIN,P)}.

22 additional processes beyond zygote could escalate
Conclusion

- We create **BigMAC**, one of the most fine-grained policy analysis frameworks for Android devices, and recover a running system’s security state from static firmware.

- **BigMAC** surpasses previous MAC-only policy analysis approaches through its layered path reduction, improving analysis results and discarding impossible runtime paths.

- We highlight **BigMACs** ability to investigate escalation paths and examine the strength of processes.

https://github.com/fics/BigMAC

https://hernan.de/z

@Digital_Cold