Honey, I Shrunk the Attack Surface
Adventures in Android Security Hardening
Nick Kralevich - Blackhat
July 27th, 2017
Agenda

- Brief history of Android security
- Strategies for dealing with vulnerabilities
- Bugs and attack surface reduction efforts
- Recognition
- The future
- Questions?
Before we begin...
Android security is more than device security...
Layers of defense even before code gets to the device...

- Google Play
- Unknown Sources Warning
- Install Confirmation
- Verify Apps Consent
- Verify Apps Warning
- Runtime Security Checks
- Sandbox & permissions
Technology throughout Google working together

Google Play
- Install Apps
- App X
- App Y
- App Z
- Attest API

Application Analysis
- Static Dynamic Reputation Etc.
- Knowledge PHA or not Best practices

Other Google Services
- Search
- Drive
- Ads Etc.

Android
- App Sandbox
- Verified Boot
- Encryption Etc.

Chrome
- Smart Lock
- Device Manager
- Safe Browsing
- Google Play Protect
- Verify Apps
- Data App installs Install Source

SafetyNet Analysis
- Exploit Detection
- ACE
- SIC Etc.

Knowledge PHA or not

Risk Signal

Data Rare Apps

Protections
- Warnings
- Configuration changes Etc.

Device Data
- Events
- Measurements Configurations Etc.

Configuration changes Etc.

Device Data

Data Signal

Etc.
Key Principles
Key Android Security Principles

- Exploit Mitigation
- Attack Surface Reduction
  - Exploit Containment
  - Principle of Least Privilege
- Safe by design APIs and interfaces
- Architectural Decomposition
Stepping back in time...
10 years ago...

- Windows Vista was released
  - Replaced "administrator-by-default" philosophy of Windows XP
- All desktop OSes
  - No difference between application capabilities and user capabilities (remains mostly true today)
  - User has Administrator / root access (still mostly true today)
- Mobile devices
  - Primarily feature phones
  - Smart devices not widely available
Android enters the picture

- HTC Dream - October 22nd, 2008
  - First commercially available Android device
- Centralized application store
- Application sandboxing
- Memory safe programming language (Java)
- Designed with security in mind
- Strong desire to not repeat the security mistakes of legacy consumer OSes
Early Android Security

- Exploit mitigation technologies were the primary focus
  - fstack-protector
  - ASLR
  - NX
  - FORTIFY_SOURCE
  - mmap_min_addr
  - Format string vulnerabilities
  - etc...

https://source.android.com/security/enhancements/
Early Android Security

Applications sandboxed using Linux UID technologies. Sandboxing of other processes done on a limited basis.

Global “root” user which was unconstrained and targeted for attack.

IPC boundaries were not consistently defined and enforced.

Security “policy” not auditable.

http://powerofcommunity.net/poc2016/keen.pdf
Heavy early use of discretionary access control (DAC) tools.

- Address space separation/process isolation
- UID controls
- UNIX permissions
- DAC capabilities
- namespaces
- ...

Greater focus on compartmentalization, attack surface reduction

- Sept 2011
- Proven effectiveness at preventing or mitigating 7 rooting exploits
- Oct 2013: Android 4.4 partially enforcing
- Oct 2014: Android 5.0 fully enforcing
Immediate success in mitigating exploits!

- vold “asec create” exploit (Android 4.4)
- Constrained attack surface mitigated exploit
- Blocked several ways
  - /data/local/tmp directory and file access disallowed
  - No symlink following allowed
  - Mount restrictions

- [http://www.androidpolice.com/2014/06/04/android-4-4-3-patch-finally-closes-ancient-vulnerability-shuts-several-serious-security-exploits/](http://www.androidpolice.com/2014/06/04/android-4-4-3-patch-finally-closes-ancient-vulnerability-shuts-several-serious-security-exploits/)
- [https://plus.google.com/u/0/+JustinCaseAndroid/posts/7BxgPNc7ZJs?cfem=1](https://plus.google.com/u/0/+JustinCaseAndroid/posts/7BxgPNc7ZJs?cfem=1)
Modern Day Android Security

Every process compartmentalized (including UID=0 processes)

- “root” no longer exists on Android

Principle of least privilege widely deployed

Attack surface limited through tightly controlled IPC boundaries

Auditable security policy

Most executable code comes from signed source / cryptographically verified (dm-verity).

http://powerofcommunity.net/poc2016/keen.pdf
Android Today

User 1
- Google Play
  - Contacts
  - Game X
  - Email
  - Game Y

User 2
- Google Play
  - Contacts
  - Email
  - Game Y
  - Game Z

Google Security Services
- Verify Apps
- Safe Browsing
- Device Manager
- Smart Lock

Google Play Protect

System

Trust Zone
- UID=0
- UID=0
- UID=0
- UID=0
- UID=0

Kernel

Hardware
Attack Surface Reduction Examples
CVE-2017-6074: DCCP double-free vulnerability (local root)

Linux's decade-old flaw: Major distros move to patch serious kernel bug

Google fuzzer helps find 11-year-old memory-corruption flaw in the Linux kernel.
Networking Protocols

- Only a whitelist of socket families are allowed
  - Netlink Route Sockets
  - Ping Sockets
  - TCP / UDP Sockets
  - Unix stream and datagram sockets
- Whitelist allowed ioctls

```plaintext
# Restrict socket ioctls. Either
# 1. disallow privileged ioctls,
# 2. disallow the ioctl permission, or
# 3. disallow the socket class.
neverallowxperm untrusted_app domain:{ rawip_socket
tcp_socket udp_socket } ioctl priv_sock_ioctls;
neverallow untrusted_app *:{ netlink_route_socket
netlink_selinux_socket } ioctl;
neverallow untrusted_app *:{
  socket netlink_socket packet_socket key_socket
  appletalk_socket netlink_firewall_socket
  netlink_tcpdiag_socket netlink_nflog_socket
  netlink_xfrm_socket netlink_audit_socket
  netlink_ip6fw_socket
  netlink_dnrt_socket netlink_kobject_uevent_socket
  tun_socket netlink_iscsi_socket
  netlink_fib_lookup_socket netlink_connector_socket
  netlink_netfilter_socket netlink_generic_socket
  netlink_scsitransport_socket
  netlink_rdma_socket netlink_crypto_socket
} *;
```
CVE-2017-6074: DCCP double-free vulnerability (local root)

Layers of attack surface reduction

- Not compiled into Android common kernels
- Even if compiled in, not reachable due to SELinux restrictions.
- “dodged a bullet” -> “working as intended”
Whitelisted socket families - Other bugs mitigated

- Other bugs blocked
  - CVE-2016-2059 - Linux IPC router binding any port as a control port
  - CVE-2015-6642 - Security Vulnerability in AF_MSM_IPC socket: IPC_ROUTER_IOCTL_LOOKUP_SERVER ioctl leaks kernel heap memory to userspace
  - CVE-2016-2474 - Security Vulnerability - Nexus 5x wlan driver stack overflow
  - etc...
Ubuntu Linux Falls on Day 1 of Pwn2Own Hacking Competition

By: Sean Michael Kerner | March 16, 2017

The first day of the Trend Micro-sponsored Pwn2Own competition awards $233,000 in prize money to security researchers for exploiting software with previously unknown vulnerabilities.

The Pwn2Own hacking competition began on March 15, and security researchers have already successfully exploited Ubuntu Linux, Microsoft Edge, Apple Safari and Adobe Reader. In total,
CVE-2017-7184: xfrm kernel heap out-of-bounds access

- Compiled into Android kernels
- Requires CAP_NET_ADMIN
  - Available to lots of processes on Android.
- Requires netlink_xfrm_socket
  - Who has it?
CVE-2017-7184: xfrm kernel heap out-of-bounds access

- Reachability:
  - Only available to one process!
  - Effectively unreachable.

```
nnk@nick:/android$ adb pull /sys/fs/selinux/policy
/sys/fs/selinux/policy: 1 file pulled. 8.5 MB/s (451031 bytes in 0.051s)
nnk@nick:/android$ sesearch --allow -c netlink_xfrm_socket -p create ./policy
allow netmgrd netmgrd:netlink_xfrm_socket { nlmsg_write setopt setattr read lock create nlmsg_read write getattrs connect shutdown bind getopt append };```
Careful attack surface management kept these bugs from being reachable.
Attack Surface Management
Android O: Project Treble
Project Treble

- A modular base for Android
- Allows updating Android without additional work from silicon vendor
- Strong separation and APIs between vendor and Android code

Project Treble - Attack Surface Management

https://android-developers.googleblog.com/2017/07/shut-hal-up.html
Project Treble - Attack Surface Management

● Each HAL runs in its own sandbox
  ○ Limited to only capabilities needed

● Calling process no longer requires HAL permissions
  ○ Example: 20 HALs moved out of system_server

● Longer attack chain to the most vulnerable drivers

https://android-developers.googleblog.com/2017/07/shut-hal-up.html
Mediaserver hardening
Stagefright

- Series of bugs discovered mid 2015
- Integer overflow in parsing process
- Mediaserver architected for containment with minimal attack surface
- Mediaserver grew up. More features => more capabilities
- Android’s first “successful failure”
  - No evidence of widespread exploitation for 2 years now.
Media Stack Hardening in Nougat

mediaextractor: seccomp

Significant reduction in syscall attack surface

<table>
<thead>
<tr>
<th>Architecture</th>
<th>arm</th>
<th>arm64</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowed syscalls</td>
<td>42</td>
<td>34</td>
<td>42</td>
</tr>
<tr>
<td>Kernel syscalls</td>
<td>364</td>
<td>271</td>
<td>373</td>
</tr>
<tr>
<td>Percent reduction</td>
<td>89%</td>
<td>87%</td>
<td>88%</td>
</tr>
</tbody>
</table>

$ cat mediaextractor-arm64.policy
# Organized by frequency of system call
# - in descending order for best performance.
ioctl: 1
futex: 1
prctl: 1
write: 1
getpriority: 1
close: 1
dup: 1
mmap: 1
munmap: 1
openat: 1
mprotect: 1
madvise: 1
getuid: 1
...
mediaserver: additional changes

- Signed and unsigned integer overflow protections
- Remove “execmem”
  - No anonymous executable memory
- No loading executable code from outside /system (not new in Nougat)
- Executable content can only come from dm-verity protected partition
- ... and more

```c
open("/system/lib/libnetd_client.so", O_RDONLY) = 3
mmap2(NULL, 12904, PROT_READ|PROT_EXEC,
      MAP_PRIVATE, 3, 0) = 0xb6d9f000

open("/data/data/com.foo.bar/libnetd_client.
     so", O_RDONLY) = 4
mmap2(NULL, 12904, PROT_READ|PROT_EXEC,
      MAP_PRIVATE|MAP_FIXED, 4, 0) = -1 EACCES
                          (Permission denied)

mmap2(NULL, 20,
      PROT_READ|PROT_WRITE|PROT_EXEC,
      MAP_PRIVATE|MAP_ANONYMOUS, 4, 0) = -1 EACCES
                          (Permission denied)
```
mediaserver - Refactoring results

- Vastly improved architectural decomposition
- Vastly improved separation of privileges
- Riskiest code moved to strongly sandboxed process
- Containment model significantly more robust

https://android-developers.blogspot.com/2016/05/hardening-media-stack.html

"I started working on this exploit on a build of the upcoming Android N release, and anyone sitting near my desk will testify to the increased aggravation this caused me. A lot of general hardening work has gone into N, and the results are impressive."

Mark Brand
Google Project Zero

Mediaserver hardening effectiveness

Security bulletin bugs in the media stack for the first 4 months of 2017

- 23.8%: No longer security issue in N
- 3.2%: Downgraded severity from M to N
- 73.0%: No change between M and N
Google rebuilt a core part of Android to kill the Stagefright vulnerability for good

Android 7.0 has a few new security tricks up its sleeve

By Russell Brandom · @russellbrandom · Sep 6, 2016, 1:03p
Media Stack Hardening Improvements in O

Access to kernel drivers

MediaServer
ExtractorService
MediaCodecService
AudioServer
CameraServer
MediaDrmServer

with Project Treble

Access to kernel drivers

MediaServer
ExtractorService
MediaCodecService
AudioServer
CameraServer
MediaDrmServer

Audio HAL
Camera HAL
DRM HAL
Android O: Webview Security
Webview Security

- **KitKat**: Shipped with the operating system
- **Lollipop**: Separate APK updateable via the Play store
- **O Preview**: Renderer in isolated process, Safe Browsing
Linux Kernel
The kernel is the new target for vulnerability research

Security bugs reported to Android by year, broken down between userspace and kernel
Why the rise in kernel bugs?

- Lockdown of userspace makes UID 0 significantly less useful.
- 2016 is the first year > 50% of devices in ecosystem have selinux in global enforcing.
- Android Vulnerability Rewards: Critical bugs payout more $$.  
  - ... and kernel bugs tend to be high or critical severity
How are kernel bugs reached - syscall (before mitigations)

- 100% of perf vulns introduced in vendor customizations

Data: Jan 2014 → April 2016
selinux: extended permissions for ioctls

Add extended permissions logic to selinux. Extended permissions provides additional permissions in 256 bit increments. Extend the generic ioctl permission check to use the extended permissions for per-command filtering. Source/target/class sets including the ioctl permission may additionally include a set of commands. Example:

allowxperm <source> <target>::<class> ioctl unpriv_app_socket_cmds
auditallowxperm <source> <target>::<class> ioctl priv_gpu_cmds
Mitigations - attack surface reduction

Ioctl command whitelisting in SELinux

- **Wifi**
  - Originally hundreds of ioctl commands → 29 whitelisted safe network socket ioctls
  - Blocks access to all bugs without restricting legitimate access.
  - Unix sockets: wifi ioctls reachable by local unix sockets :( Hundreds → 8 whitelisted unix socket ioctls
  - No ioctls allowed on other socket types including generic and netlink sockets

- **GPU**
  - e.g. Shamu originally 36 -> 16 whitelisted commands
  - Ioctl commands needed varies by device but < 50% needed seems consistent across KGSL drivers
Mitigations - attack surface reduction

- Restrict access to perf
  - Access to `perf_event_open()` is disabled by default.
  - Developers may re-enable access via debug shell
- Remove access to `debugfs`
  - All app access to `debugfs` removed
- Remove default access to `/sys`
  - App access to files in `/sys` must be whitelisted
  - 38,000 files to 500 files (98% reduction)
Impact of mitigations

Because most bugs are driver specific, effectiveness of mitigations varies across devices. In general most previously reachable bugs were made unreachable.

- Case study of bugs reachable by apps on Nexus 6 (Shamu)
  - 100% of wifi bugs blocked
  - 50% of GPU bugs blocked
  - 100% of debugfs bugs blocked
  - 100% of perf bugs blocked (by default)
SELinux Effectiveness

SELinux reduced severity of almost half of kernel bugs
(Android security bulletin data for Jan-Apr, 2017)
Other Attack Surface Reductions

- Restricted /proc/PID visibility (hidepid=2, credit CopperheadOS)
  - Limit visibility between Android processes
  - Prevents popups, notification spam, and phishing
  - Addresses UI State Inference attacks
- DAC capabilities removal
  - Kernel module loading, writes to /system, most root capabilities
- Whitelist of /proc files (new in Android O)
  - 4400 files -> 2500 files (remainder mostly in /proc/sys/net)
- Hardlink removal
Recognition
Q: It might be good for everyone to know: Which Android devices do you find the most secure?

CunningLogic (aka jcase)

A: Android 5.x and up is particularly annoying for me to try and root, my go to tactics are often dead due to the strengthened SELinux policies.

https://www.reddit.com/r/Android/comments/3hhciw/ask_us_almost_anything_about_android_security/
Good reviews from attackers :-)
ZERODIUM Payout Ranges

LPE: Local Privilege Escalation
MTE: Mitigation Bypass
RCE: Remote Code Execution
RJB: Remote Jailbreak
SBX: Sandbox Escape
VME: Virtual Machine Escape

*All payout amounts are chosen at the discretion of ZERODIUM and are subject to change or cancellation without notice.

2016/09 © zerodium.com
Second highest exploit cost!
## Price List Changelog

Changes of Sep. 29, 2016

<table>
<thead>
<tr>
<th>Product / Exploit Type</th>
<th>New Price</th>
<th>Previous Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android 7 (Remote Jailbreak)</td>
<td>$200,000</td>
<td>$100,000</td>
</tr>
<tr>
<td>MS Edge + IE (RCE) + Sandbox Escape</td>
<td>$80,000</td>
<td>$50,000</td>
</tr>
<tr>
<td>Safari on Mac (RCE) + Sandbox Escape</td>
<td>$80,000</td>
<td>$50,000</td>
</tr>
<tr>
<td>OpenSSL or PHP (RCE)</td>
<td>$50,000</td>
<td>$40,000</td>
</tr>
<tr>
<td>MS Windows (RCE)</td>
<td>$40,000</td>
<td>$30,000</td>
</tr>
<tr>
<td>MS Office (RCE)</td>
<td>$30,000</td>
<td>$20,000</td>
</tr>
</tbody>
</table>

2x increase in exploit cost!
<table>
<thead>
<tr>
<th>Category</th>
<th>Phone</th>
<th>Price (USD)</th>
<th>“Master of Pwn” Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtaining Sensitive Information</td>
<td>Apple iPhone</td>
<td>$50,000</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Google Nexus</td>
<td>$50,000</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Other Android</td>
<td>$35,000</td>
<td>7</td>
</tr>
<tr>
<td>Install Rogue Application</td>
<td>Apple iPhone</td>
<td>$125,000</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Google Nexus</td>
<td>$100,000</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Other Android</td>
<td>$60,000</td>
<td>15</td>
</tr>
</tbody>
</table>

[http://blog.trendmicro.com/presenting-mobile-pwn2own-2016/]
pwn2own

- Price parity among the major mobile operating systems
- Smaller attack surface increases complexity and cost of finding an exploit

<table>
<thead>
<tr>
<th>Phone</th>
<th>Price (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple iPhone</td>
<td>$50,000</td>
</tr>
<tr>
<td>Google Nexus</td>
<td>$50,000</td>
</tr>
<tr>
<td>Other Android</td>
<td>$35,000</td>
</tr>
<tr>
<td>Apple iPhone</td>
<td>$125,000</td>
</tr>
<tr>
<td>Google Nexus</td>
<td>$100,000</td>
</tr>
<tr>
<td>Other Android</td>
<td>$60,000</td>
</tr>
</tbody>
</table>
## pwn2own successes

<table>
<thead>
<tr>
<th>Contest</th>
<th>Core Android Platform Bug</th>
<th>Additional Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009 pwn2own</td>
<td>NO</td>
<td>All mobile devices unexploited</td>
</tr>
<tr>
<td>2010 pwn2own</td>
<td>NO</td>
<td>iPhone 3GS compromised. No Android compromised</td>
</tr>
<tr>
<td>2011 pwn2own</td>
<td>NO</td>
<td>Google Stays Strong</td>
</tr>
<tr>
<td>2012 pwn2own</td>
<td>NO</td>
<td>Non-Android device specific parsing bug - NFC delivered</td>
</tr>
<tr>
<td>2013 pwn2own</td>
<td>NO</td>
<td>Non-Android device specific bug</td>
</tr>
<tr>
<td>2014 pwn2own</td>
<td>YES</td>
<td>1. NFC triggered remote leak of Bluetooth MAC address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. DHCP code execution (partial win)</td>
</tr>
<tr>
<td>2015 pwn2own</td>
<td>NO</td>
<td>Chrome exploit -&gt; Google Play Install - No OS compromise</td>
</tr>
<tr>
<td>2016 pwn2own</td>
<td>NO</td>
<td>Chrome exploit -&gt; Google Play Install - No OS compromise</td>
</tr>
</tbody>
</table>
No success from the Project Zero prize

Didn't we offer you enough? Google's $350,000 Project Zero prize attracts junk entries

Was Google's Project Zero prize too difficult or was the prize just too small?

By Liam Tung | March 31, 2017 -- 11:34 GMT (04:34 PDT) | Topic: Security

Accelerating bug discovery

Google Increases Maximum Android Bug Bounties to $200,000

By Ryan Whitwam on June 2, 2017 at 3:01 pm | 6 Comments

https://www.extremetech.com/mobile/250316-google-increases-android-bug-bounties-much-200000
“... no researcher has claimed the top reward for an exploit chains in 2 years ...”

<table>
<thead>
<tr>
<th></th>
<th>Old Amount</th>
<th>New Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote chain to TrustZone or Verified Boot compromise</td>
<td>$50,000</td>
<td>$200,000</td>
</tr>
<tr>
<td>Remote to Kernel</td>
<td>$30,000</td>
<td>$150,000</td>
</tr>
</tbody>
</table>

https://security.googleblog.com/2017/06/2017-android-security-rewards.html
“Furthermore, when SELinux became common on Android, this became more problematic since the radio SELinux context that rild started with was too restrictive for the implant to function.”

https://wikileaks.org/ciav7p1/cms/page_28049453.html
Future
## Future: Global Seccomp Whitelist

<table>
<thead>
<tr>
<th>Architecture</th>
<th>syscalls provided by kernel</th>
<th>syscalls in bionic</th>
<th>reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>arm</td>
<td>364</td>
<td>204</td>
<td>44</td>
</tr>
<tr>
<td>arm64</td>
<td>271</td>
<td>198</td>
<td>27</td>
</tr>
<tr>
<td>x86</td>
<td>373</td>
<td>203</td>
<td>46</td>
</tr>
<tr>
<td>x86_64</td>
<td>326</td>
<td>199</td>
<td>39</td>
</tr>
</tbody>
</table>
Future Attack Surface Reduction

- Take better advantage of Treble - system / vendor split
- Continued reduction in /proc files
- Removal of useless /dev files
  - Faster boot time, less kernel code, less attack surface
- Stronger IPC controls
- System Properties
- Finer grain attack surface reduction for applications
- Scale back shared data stores
Takeaways
Takeaways

● Attack surface management is critical to preventing or mitigating unknown bugs.
● Android has invested significantly in reducing attack surface and containing processes.
● Vulnerabilities will never go away, but they can be contained and managed.
“Perfection is achieved not when there is nothing more to add, but when there is nothing left to take away.”

- Antoine de Saint-Exupery - 1939
THANK YOU

security@android.com