Android Security: Taming the Complex Ecosystem

Stanford CS155 guest lecture
2019-05-23
René Mayrhofer, Director of Android Platform Security

Personal Twitter: @rene_mobile
Outline

1. The Ecosystem and State of the Union (*The Marketing Part*)


3. Taming Complexity (*The “What I learned” Part*)

4. Where do we go from here (*The Future Part*)?
State of the (Android) Union
The Android ecosystem in numbers

> 1.300 brands
> 24.000 devices
> 1 M apps
> 2 B users

https://www.blog.google/around-the-globe/google-europe/android-has-created-more-choice-not-less/
Measuring exploitation difficulty: 0-day pricing
Measuring exploitation difficulty: 0-day pricing

**Mobile Pwn2Own 2016**
- **iPhone**: $125,000.00
- **Nexus**: $75,000.00

**Mobile Pwn2Own 2017**
- **iPhone**: $150,000.00
- **Pixel**: $100,000.00

**Mobile Pwn2Own 2018**
- **iPhone X**: $75,000.00
- **Pixel 2**: $50,000.00

- **Sandbox**
- **Unauth App Install**
- **Browser**
- **Kernel Bonus**
- **Persistence Bonus**
- **Baseband**
- **Short distance wireless**
- **Kernel Bonus**
- **Messaging (SMS/MMS)**
critical security vulnerabilities affecting the Android platform in 2018 publicly disclosed without a security update or mitigation available
Android patching has improved

1B
Devices patched in 2018.

29%
more devices patched QoQ in Q4.

84%
more devices patched in Q4 than same time last year.
Malware is a universal risk

“This year, we celebrated the 30th anniversary of the World Wide Web. Fast forward thirty years and the threat landscape is exponentially more complex, and the available attack surface is growing faster than it has at any other point in the history of technology,” commented Ondrej Vlcek, President of Consumer at Avast.

New research reveals a dozen iPhone apps linked to Golduck malware

New research from Wandera find over a dozen iPhone apps linked to Golduck malware. The findings underline that fake apps is by no means an Android-only problem.
World’s most widely used Anti-Malware solution

Security protection for everyone (Play and off-Play).
Always updating to provide the latest protections from Google AI.

Scans apps daily - from both within Google Play and outside of it.
Remediates by removing potentially harmful apps (PHA).

50B Apps verified per day
2+B Devices protected
500K Apps analyzed per day
Identifies potential security enhancements when apps are uploaded to Play 300,000 developers have fixed 1,000,000+ Play apps.
In 2018, downloading a PHA from Google Play was 0.04%, and outside of Google Play was 0.92%.
The Android Platform Security Model

Security Goals

1. Protecting user data
   a. Usual: device encryption, user authentication, memory/process isolation
   b. Upcoming: personalized ML on device

2. Protecting device integrity
   a. Usual: malicious modification of devices
   b. Interesting question: against whom?

3. Protecting developer data
   a. Content
   b. IP
Threat Model

- Adversaries can get physical access to Android devices
  - Powered off
  - Screen locked
  - Screen unlocked by different user
  - Physical proximity

- Network communication and sensor data are untrusted
  - Passive eavesdropping
  - Active MITM

- Untrusted code is executed on the device
- Untrusted content is processed by the device
- **New**: Insiders can get access to signing keys

Principles

- Actors control access to the data they create
- Safe by design/default
- Consent is informed and meaningful
- Defense in depth
The Android Platform Security Model

(1) Multi-Party Consent
The Android Platform Security Model

(2) Open ecosystem access
(3) Security is a compatibility requirement
(4) Factory reset restores the device to a safe state
(5) Applications are security principals
Implementing the Security Model

Strategies

● Contain: isolating and de-privileging components, particularly ones that handle untrusted content.
  ○ Access control: adding permission checks, increasing the granularity of permission checks, or switching to safer defaults (for example, default deny).
  ○ Attack surface reduction: reducing the number of entry/exit points (i.e. principle of least privilege).
  ○ Architectural decomposition: breaking privileged processes into less privileged components and applying attack surface reduction.

● Mitigate: Assume vulnerabilities exist and actively defend against classes of vulnerabilities or common exploitation techniques.

- reduce reachability of code
- reduce impact of reachable bugs
It all starts with secure hardware

**TEE (Trust environment)** used for key generation, key import, signing and verification services are executed in hardware.

**Secure Lock Screen, PIN verification & Data encryption** (PIN+HW key) used to derive encryption keys.

**Version binding** ensures keys created with a newer OS cannot be used by older OS versions.

**Rollback prevention** (8.0+) prevents downgrading OS to an older less secure version or patch level.

**Verified Boot** provides cryptographic verification of OS to ensure devices have not been tampered with.

**Tamper-resistant hardware** (Android Pie) offers support to execute cryptographic functions in dedicated hardware.
Question:

Make bootloader/verified boot state available to all apps?
SELinux, process isolation and sandboxing

Android is built on SELinux where if an exploit is found, the attack vector is limited to the domain the exploit is able to execute in.

Application sandboxing ensures that application and system data is inaccessible from other apps. Each app runs in its own user ID (UID) - limiting exposure of apps to get data from one another.

Work profile apps are prevented from communicating with personal apps by default. Work profile apps run in a separate user space with separate encryption keys from personal apps, further limiting exposure, EMMs cannot manage the personal device when the device is managed only via the Work Profile.
Question:

Controlling device-wide parameters from profile owner?
Layers of containment on main AP

Personal app 1
Personal app 2
Work app 1
Work app 2
Personal app 3
Personal app 4
Work profile

Primary Android user
Additional Android user

Personal app 3
Personal app 4

Android Linux kernel
VM kernel

Hypervisor

HAL 1
HAL 2

System service 1
System service 2

VM apps

FP match
Key-master
Gate-keeper

TEE app 1
TEE app 2

TEE kernel

Hardware mode monitor
Question:
Dynamic SELinux policy update at run time?
Question:
Add another runtime permission for <X>?
Anti-exploitation

Bug = Exploit

- ASLR/KASLR
- Hardened ucopy
- ASAN/Fuzzing
- IOSan
- CFI/KCFI
- PAN
- LTS

Applications
Android Framework
Native Libraries
Android Runtime
HAL
Linux Kernel
The tiered authentication model

**Primary Auth**
- Knowledge-factor based
- Most secure

**Secondary Auth**
- Needs primary auth
- Less secure
- Somewhat constrained

**Tertiary auth**
- Needs primary auth
- Least secure
- Most constrained
Question: Expose authentication details to all apps?
Taming Complexity
Many variants and stakeholders: Enabling an active ecosystem

> 1.300 brands
> 24.000 devices
> 1 M apps

Can be written in any language
Question:
How many different platform signing keys?
Taming complexity in variants

**Compatibility Definition Document (Standards)**
- Defines requirements a device needs to fulfill to be considered "Android"
- Updated for every Android release
  - Many changes scoped to apps targeting this version
- Needs to strike a balance between strong standard base and openness for innovation
  - Some requirements scoped to hardware capabilities (e.g. form factors)
- Updating security requirements is one important means of driving ecosystem to improvement

**Compatibility/Vendor/Security/... Test Suite (Enforcement)**
- Tests need to be run by device manufacturer
- Guaranteed conformance to (testable parts of) CDD
  - *In Android Q, ca. 800 tests for SELinux policy*
- Usability of Android trademark and Google apps bound to passing tests
- Complexity in test execution:
  - Automation of test cases
  - Visibility on "user" firmware builds
Question:
How quickly to change the requirements?
Changing the ecosystem is hard - Various strategies

1. Introducing new requirements initially as optional, becoming mandatory only in future releases → time for development, testing, adaptation

   Important lesson: **Clear communication of plans way ahead of schedule**

2. Ratcheting requirements from release to release with a pace that lets hardware keep up (including low-end devices and verticals) or keeping carve-outs

   Important lesson: **Let the tail end of the ecosystem keep up**
100% of compatible devices launching with Q will encrypt user data.
<table>
<thead>
<tr>
<th>Strong</th>
<th>Weak</th>
<th>Convenience</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAR: 0-7%</td>
<td>SAR: 7-20%</td>
<td>SAR: &gt;20%</td>
</tr>
<tr>
<td>Pipeline: Secure</td>
<td>Pipeline: Secure</td>
<td>Pipeline: (In)secure</td>
</tr>
<tr>
<td>● 72-hours before fallback to primary auth</td>
<td>● 12 hours before fallback to primary auth</td>
<td>● 4 hours before fallback to primary auth</td>
</tr>
<tr>
<td>● Application integration via BiometricPrompt, FIDO2, or custom APIs</td>
<td>● No application integration of any kind.</td>
<td>● No application integration of any kind.</td>
</tr>
</tbody>
</table>
**Strong**

- SAR: 0-7%
- Pipeline: Secure
- 72-hours before fallback to primary auth
- Application integration via BiometricPrompt, FIDO2, or custom APIs

**Weak**

- SAR: 7-20%
- Pipeline: Secure
- 12 hours before fallback to primary auth
- No application integration of any kind.

**Convenience**

- SAR: >20%
- Pipeline: (in)secure
- 4 hours before fallback to primary auth
- No application integration of any kind.
Target API version requirements

Actively maintained apps (forefront) in Play

**August 2019**: New apps are required to target API level 28 (Android 9) or higher.

**November 2019**: Updates to existing apps are required to target API level 28 or higher.

Apps not getting updates (tail end) on device

**August 2019**: New apps will receive warnings during installation if they do not target API level 26 or higher.

**November 2019**: New versions of existing apps will receive warnings during installation if they do not target API level 26 or higher.

Apps targeting Pie, usage of NetworkSecurityConfig

- Not configured to block cleartext
- Block cleartext with manually-configured exceptions
- >80% block all cleartext with no exceptions (default)

Source: Google Internal Data, 2019-04-01
Taming complexity in stakeholders

Tooling

- Compiler/build toolchain ideally used by all stakeholders (e.g. drivers, TrustZone, etc. code)
- Can add new mitigations at this level, but *typically breaks old code*

Upstream first approach

- Importance to commit changes to common upstream code (e.g. Linux kernel, clang, etc.)
- Encouraging other stakeholders to upstream their changes (either to common upstream or to AOSP)

Open source and common issue trackers
Where do we go from here?
Identity Credentials
Identity Credential Application
  e.g. “CA Driving License”

Framework APIs
Credential Store
Transaction Viewer App

Identity Credential Impl
typically in tamper-resistant HW

Keymaster Attestation
typically in TEE

Android OS

Reader / Verifier
e.g. “VT State Police”

Issuing Authority
e.g. “CA DMV”

NFC / Bluetooth / Wifi Direct

Internet Protocol
Identity Credential impl in secure hardware
Security and Privacy for draft mDL standard

● **Security** properties:
  ○ **Anti-forgery**: Identity Credential data is signed by the Issuing Authority
  ○ **Anti-cloning**: Secure Hardware produces MAC during provisioning using a key derived from a private key specific to the credential and an ephemeral public key from the reader. Public key corresponding to credential private key is signed by the Issuing Authority
  ○ **Anti-eavesdropping**: Communications between Reader/Verifier and Secure Hardware are encrypted and authenticated

● **Privacy** properties:
  ○ **Data minimization**: Reader/Verifier only receives data consented to by the holder. Backend infrastructure does not receive information about use
  ○ **Unlinkability**: Application may provision single-use keys
  ○ **Auditability**: Every transaction and its data is logged and available only to the Holder (not the application performing the transaction)
Question:
Strictly require secure (certified) hardware?
Status

Android Q
- No changes to platform itself
- Software implementation as compatibility library
- SecurityType = SOFTWARE_ONLY
- CertificationLevel = NONE
- Can start developing identity apps, library will be compatible with vast majority of Android devices

Future versions
- HAL implementation based on secure hardware
- Optional **Direct Access** support
- Credential Store system daemon
- Framework APIs
Insiders
Simple and few trusted components
Hardware
Threat models / scenarios for hardware security

● **Basic** assumption for hardware security:
  ○ Adversary has possession of the hardware
  ○ Adversary has control over all network channels
  ○ Adversary can influence sensor readings/input

● **Intermediate** assumptions:
  ○ Side channel analysis: including power, RF, timing, and potentially others
  ○ Side channel injection: including power, clock, RF (up to laser), and potentially others
  ○ Reverse engineering of hardware
  ○ Modification of hardware on PCB level, but not chip level

● **Advanced** assumptions:
  (AKA nation state adversaries or *insider* threats)
  ○ Modification of hardware on chip level
  ○ Access to internal signing keys
Open research question:
Transparency and meaningful auditability for hardware components
Wipe on firmware update without user involvement

[C-SR] are strongly recommended to provide insider attack resistance (IAR), which means that an insider with access to firmware signing keys cannot produce firmware that causes the StrongBox to leak secrets, to bypass functional security requirements or otherwise enable access to sensitive user data. The recommended way to implement IAR is to allow firmware updates only when the primary user password is provided via the IAuthSecret HAL. IAR will likely become a requirement in a future release.

https://source.android.com/compatibility/9.0/android-9.0-cdd Section 9.11.2. StrongBox
Insider Attack Resistance for user PIN/password/pattern

**Google Pixel 2 (Weaver)**

- Javacard applets on NXP secure element hold secrets and compare user knowledge factor
- Explicitly *doesn’t implement data backup functionality*
- If app is updated, secrets are wiped
- NXP SE **OS upgrade itself requires app to be uninstalled**, wiping secrets.
- If a new app is needed, it’s installed alongside the old, and secrets are migrated when used.

**Google Pixel 3 (Weaver and Strongbox)**

- Custom firmware on Google Titan M
- Firmware update is atomic with A/B (active/inactive) slots
- **Any new firmware is put into untrusted “hold” state** during installation to inactive slot
- Only providing matching user knowledge factor transitions it into trusted active slot
- Resetting knowledge factor (e.g. for RMA) forces wiping secrets beforehand

[https://www.blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/](https://www.blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/)
System (OS)
Transparency for system updates
Android Verified Boot (AVB) / VBMeta

- AVB uses VBMeta structures to describe/verify elements of the boot chain.
- Bootloader stores hash measurement of VBMeta into KeyMaster v4
- VBMeta lives either in its own partition or on chained partitions
- The hash of VBMeta can be remotely attested with Key Attestation

https://android.googlesource.com/platform/external/avb/
https://developers.google.com/android/images
VBMeta digest verification

Getting reference VBMeta digest

1. Download Factory Image
2. Unzip Factory Image
3. avbtool verify image
4. avbtool calculate vbmeta digest
5. VBMeta Digest

Attestation and verification of VBMeta digest

Device side
1. Generate KeyPair
2. Get Key Attestation Cert Chain
3. Validate Key Attestation Cert Chain

Server side
1. VBMeta Digest from Cert Extension
2. Match?

https://android.googlesource.com/platform/external/avb/
https://developers.google.com/android/images
End-to-end backup encryption
Encrypted backup key protocol (simplified)

Backup

Restore

Google Cloud Key Vault

THM

https://developer.android.com/about/versions/pie/security/ckv-whitepaper
https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html
Encrypted backup key protocol (simplified)

Backup

1. \( K \) and \( \text{pin} \) sent to THM

2. \( E_{PK}(E_{\text{pin}}(K)) \) sent to Google Cloud Key Vault

Restore

THM retrieves \( E_{PK}(E_{\text{pin}}(K)) \) from Google Cloud Key Vault

\( K \) and \( \text{pin} \) are decrypted

Links:
- https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html
Encrypted backup key protocol (simplified)

**Backup**

1. \( PK \) -> THM

2. \( E_{PK}(E_{pin}(K)) \) -> Key Vault

**Restore**

3. \( PK \) -> THM

4. \( E_{PK}(E_{pin}(K)) \) -> Device

5. \( k' \) -> Device

---

https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html
Encrypted backup key protocol (simplified)

Google Cloud Key Vault

1. $E_{PK}(E_{pin}(K))$
2. $E_{PK}(E_{pin}(K))$
3. $E_{pin}(K)$
4. $pin \neq pin'$
5. $E_{PK}(pin' + k')$
6. (w. failure counter)

Backup

Restore

Google

https://developer.android.com/about/versions/pie/security/ckv-whitepaper
https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html
Encrypted backup key protocol (simplified)

Google Cloud Key Vault

1. Backup
   - $K$
   - $\text{pin}$
   - $PK$
   - $E_{PK}(E_{\text{pin}(K)})$

2. THM
   - $E_{\text{pin}(K)}$

3. Restore
   - $k'$
   - $\text{pin}'$
   - $PK$
   - $E_{PK}(\text{pin'} + k')$

4. $\text{THM}$
   - $E_{\text{pin}(K)}$
   - $\text{pin} = \text{pin'}$

5. $\text{THM}$
   - $E_{\text{pin}(K)}$
   - $\text{pin} = \text{pin'}$

6. THM
   - $k'$
   - $\text{pin} = \text{pin'}$

7. $E_{\text{pin}(K)}$
   - $k'$
   - $\text{pin} = \text{pin'}$

8. $E_{PK}(\text{pin'} + k')$
   - $K$

References:
- https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html
Dynamic code

Apps
Auditability is a key defense against insider attacks
Don’t take my word for it
(Some) Resources

- https://www.android.com/security-center/
- https://source.android.com/security
- https://developer.android.com/training/articles/security-tips
- https://android-developers.googleblog.com/search/label/Security
- https://www.google.com/about/appsecurity/research/presentations/
- https://www.mayrhofer.eu.org/post/android-tradeoffs-0-meta/
- https://www.mayrhofer.eu.org/post/android-tradeoffs-1-rooting/
Appendix
Calculating VBMeta Digest from Factory Image

- Build avbtool from AVB 2.0 AOSP.
- Download and unzip factory image for Pixel 3.
- Validate that VBMeta structures match up with referenced partitions.
  - `avbtool verify_image --image vbmeta.img --follow_chain_partitions`
- Calculate VBmeta Digest
  - `avbtool calculate_vbmeta_digest --image vbmeta.img`
Attesting VBMeta Digest

- `DevicePolicyManager.generateKeyPair()` to get AttestedKeyPair
- `AttestedKeyPair.getAttestationRecord()` to get Key Attestation Cert Chain
- Validate the chain up to the Google root certificate
- Extract extension OID 1.3.6.1.4.1.11129.2.1.17 from leaf certificate
- RootOfTrust sequence contains verifiedBootHash field with VBMeta Digest

```plaintext
RootOfTrust ::= SEQUENCE {
  verifiedBootKey OCTET_STRING,
  deviceLocked BOOLEAN,
  verifiedBootState VerifiedBootState,
  verifiedBootHash OCTET_STRING,
}
```
Encrypted backup key protocol (Details)

- Android Framework
- Recovery Agent
- Vault Server

List<cohort_pk>, list_signature

Verify list_signature over List<cohort_pk> against the installed root of trust

Randomly choose a cohort_pk from List<cohort_pk>, validate it against the root cert, and save it for future use

Android Framework
Recovery Agent
Vault Server

Cohort public keys: https://www.gstatic.com/cryptauthvault/v0/cert.xml

https://developer.android.com/about/versions/pie/security/ckv-whitepaper
https://security.googleblog.com/2018/10/google-and-android-have-your-back-by.html
User enters LSKF to unlock the phone, where local_lskf_hash is computed from the raw LSKF

Generate a random symmetric key recovery_key

locally_encrypted_recovery_key = SecureBox.encrypt(
    their_public_key = ",",
    shared_secret = local_lskf_hash,
    header = "V1 locally_encrypted_recovery_key",
    payload = recovery_key)

THM_KF_hash = SHA256("THM_KF_hash" || local_lskf_hash)

Construct vault_metadata and vault_params

THM_encrypted_recovery_key = SecureBox.encrypt(
    their_public_key = cohort_pk,
    shared_secret = THM_KF_hash,
    header = "V1 THM_encrypted_recovery_key" || vault_params,
    payload = locally_encrypted_recovery_key)

THM_encrypted_recovery_key, vault_params, vault_metadata

THM_encrypted_recovery_key, vault_params, vault_metadata
THM_KF_hash = SHA256("THM_KF_hash" || local_lskf_hash)

Generate a random symmetric key key_claimant

recovery_claim = SecureBox.encrypt(
    their_public_key = cohort_pk,
    shared_secret = "",
    header = "V1_KF_claim" || vault_params || challenge,
    payload = THM_KF_hash || key_claimant)

recovery_claim

recovery_claim, vault_handle

Android Framework

Recovery Agent

Vault Server
Vault Server

recovery_claim, vault_params, encrypted_recovery_key, challenge

Check if the failed attempt counter is under the limit

THM_KF_hash || key_claimant = SecureBox.decrypt(
  our_private_key = cohort_sk,
  shared_secret = "",
  header = "V1 KF_claim" || vault_params || challenge,
  encrypted_payload = recovery_claim),
where cohort_sk is the private key corresponding to cohort_pk

locally_encrypted_recovery_key = SecureBox.decrypt(
  our_private_key = cohort_sk,
  shared_secret = THM_KF_hash,
  header = "V1 THM_encrypted_recovery_key" || vault_params,
  encrypted_payload = THM_encrypted_recovery_key)

reencrypted_recovery_key = SecureBox.encrypt(
  their_public_key = "",
  shared_secret = key_claimant,
  header = "V1 reencrypted_recovery_key" || vault_params,
  payload = locally_encrypted_recovery_key)

 Vault Server

 reencrypted_recovery_key

 Trusted Hardware Module
locally_encrypted_recovery_key = SecureBox.decrypt(
    our_private_key = "",
    shared_secret = key_claimant,
    header = "V1 reencrypted_recovery_key" || vault_params,
    encrypted_payload = reencrypted_recovery_key)

recovery_key = SecureBox.decrypt(
    our_private_key = "",
    shared_secret = local_lskf_hash,
    header = "V1 locally_encrypted_recovery_key",
    encrypted_payload = locally_encrypted_recovery_key)