Insider Attack Resistance in the Android Ecosystem

Enigma 2019
Burlingame, CA, USA - 2019-01-29, 16:30-17:00
René Mayrhofer, Director of Android Platform Security

Personal Twitter: @rene_mobile
Insiders
Simple and few trusted components
[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/)
DCL

Apps

System (OS)

Firmware

Hardware
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

- Generate KeyPair
- Get Key Attestation Cert Chain
- Validate Key Attestation Cert Chain
- VBMeta Digest from Cert Extension
- Match?

Device side

Server side

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

Backup

Google Cloud Key Vault

THM

Restore

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

Backup

1. THM

2. $E_{PK}(E_{pin}(K))$

Restore

Google Cloud Key Vault

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

K
pin
PK

E_{PK}(E_{pin}(K))

THM

1

Google Cloud Key Vault

2

E_{PK}(E_{pin}(K))

Restore

k'
PK

3

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

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

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

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

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

5. THM
   - $E_{PK}(E_{pin}(K))$

6. THM
   - $E_{pin}(K)$
   - $6$ (w. failure counter)

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

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

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

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

4. THM
   - \(E_{pin}(K)\)

5. Restore
   - \(pin'\)

6. (w. failure counter)
   - pin' \(\neq\) pin'

7. THM
   - \(K\)

8. Restore
   - \(K\)

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

Apps

System (OS)

Firmware

Hardware
Auditability is a key defense against insider attacks
Don’t take my word for it
Appendix
Calculating VBMeta Digest from Factory Image

- Build avbtool from [AVB 2.0](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

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

Cohort public keys: https://www.qstatic.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_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
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)

reencrypted_recovery_key

Vault Server

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)