Inside Android's SafetyNet Attestation

Collin Mulliner

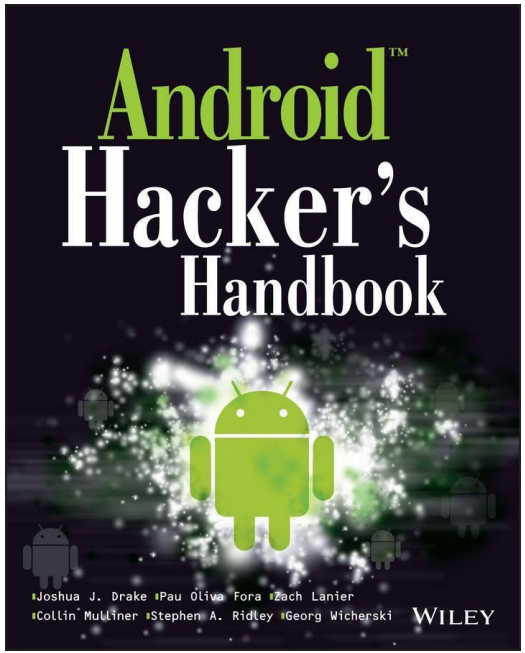
collin@mulliner.org
www.mulliner.org/collin
@collinrm

mobile security monthly blog: www.mulliner.org/blog

About Me

- Mobile App development since 1997 (for PalmOS)
- Mobile Security since 1998 2001 (hard to tell)
- Touched: PalmOS, J2ME, S40, Windows Mobile, Symbian, Android, iOS
- Worked a lot on MMS (2005-2006) and SMS (2009-2011) security
- Involved in several books on smartphone security
- Worked at mobile device manufacturers
- Worked on mobile app security team
- I find and report issues to device/OS/app manufacturers

Books and Papers



https://mulliner.org/collin/#techreports

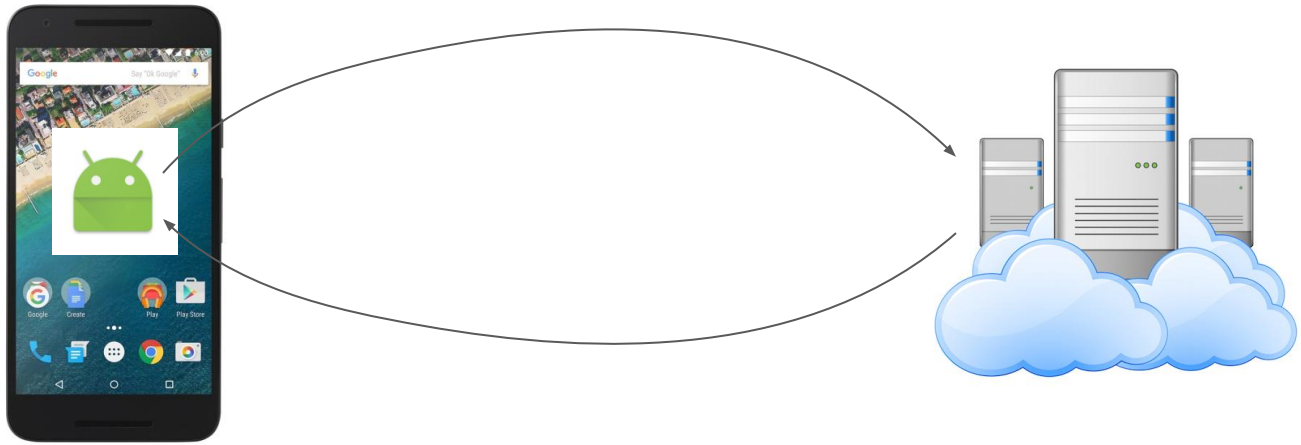
Goals for this Talk

- Understand Android's SafetyNet Attestation
  - How to implement and deploy
- SafetyNet Attestation
  - What can and can't it do
- Attacks & Bypasses
  - Past, current, and future!
- Document SafetyNet Attestation, Google's documentation not good enough.
  - This was the main idea for the talk!

Mobile Apps

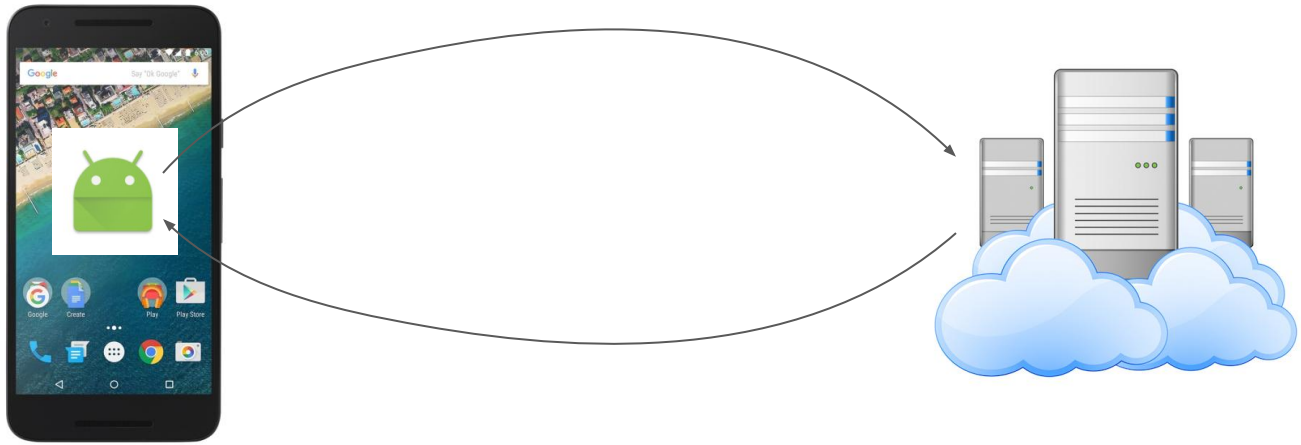
Mobile Apps

- Apps communicate ...



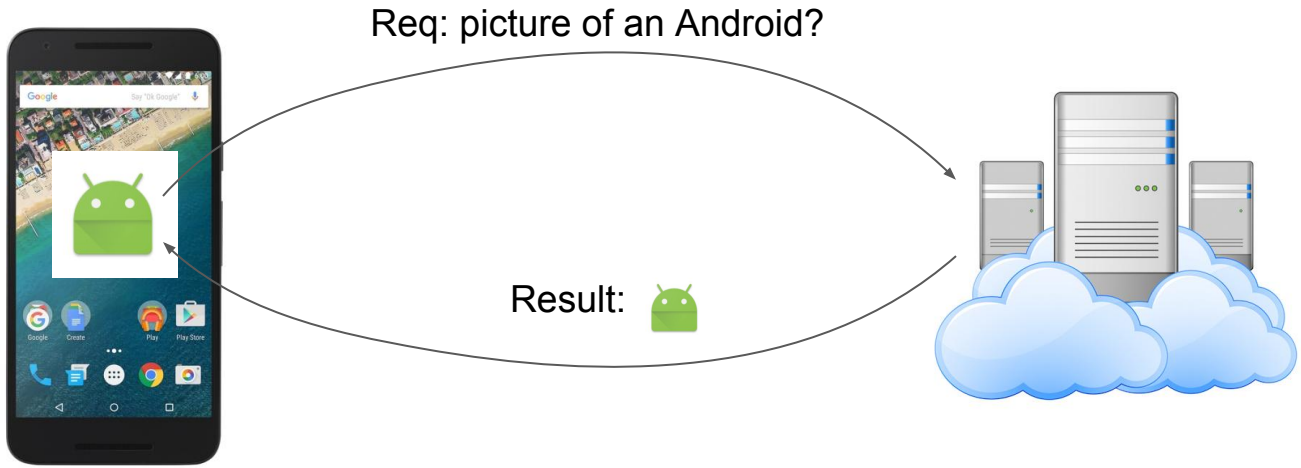
Mobile Apps

- App communicates with **its backend**



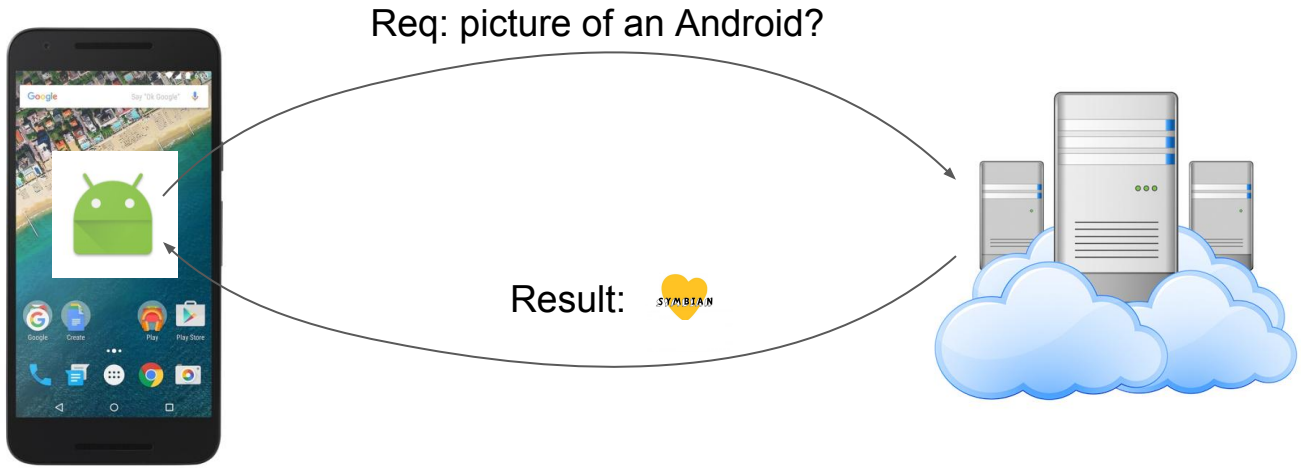
Mobile Apps

- App communicates with its backend
- App works as expected → User & App-Maker happy (=$$$$$)



Mobile Apps

- App communicates with its backend
- Attack! → User and/or App-Maker unhappy (~~$$$$$~~)



Mobile App Security

- App is the gateway to the service
  - More so if mobile first or mobile only (and no public APIs)

- Data displayed & managed by app
  - User is allowed to see content in the app but isn't allowed to copy it

Mobile App Security protects: Service, Revenue, Brand, User / Customer

Attacks

- OS Modification
  - Root device → break assumptions
    (read private data, take screenshot, instrument app, ...)
- App Modification
  - Make app do “something” else
- Network Traffic
  - Modify request / response (mostly solved with TLS and cert-pinning)

Rooting

- Re-Gain full control over device
  - Just one step towards attacking apps

- Access any resource
  - Take screenshot, debug any app, instrument process
- Read / Write any file
  - Read private app data
- Modify OS and software framework
  - API returns different result

Highly dependent on Android version due to SELinux (longer discussion…)

App Security the old Days

- Rooting checks
  - access(“/system/xbin/su”, F_OK)
  - com.chainfire.supersu installed?
- Check for instrumentation tools
  - Xposed installed ?
- Emulator detection
  - if (getDeviceId() == 0) ....

Hard Coded Checks

- Developer, easy to:
  - Understand
  - Implement
  - Deploy (app doesn’t start or tells backend to deny access)

- Attacker, easy to:
  - Understand
  - Circumvent (remove check from app, rename file, ...)
  - (Ab)use app

Modern Mobile App Security

- **Run checks by collecting data on the client but enforce on the backend**

- Attacker can’t just patch out checks but has to fake data
  - Much more work and uncertainty about what is used for check

- **This is what SafetyNet Attestation does!**

Android becomes more Secure

- Secure Boot
  - Trust Anchor for the rest of the platform
- Way way more SELinux restrictions
  - Stronger sandbox

- Google builds platform security service: SafetyNet
  - Provide basic security features needed by many apps

Android’s SafetyNet

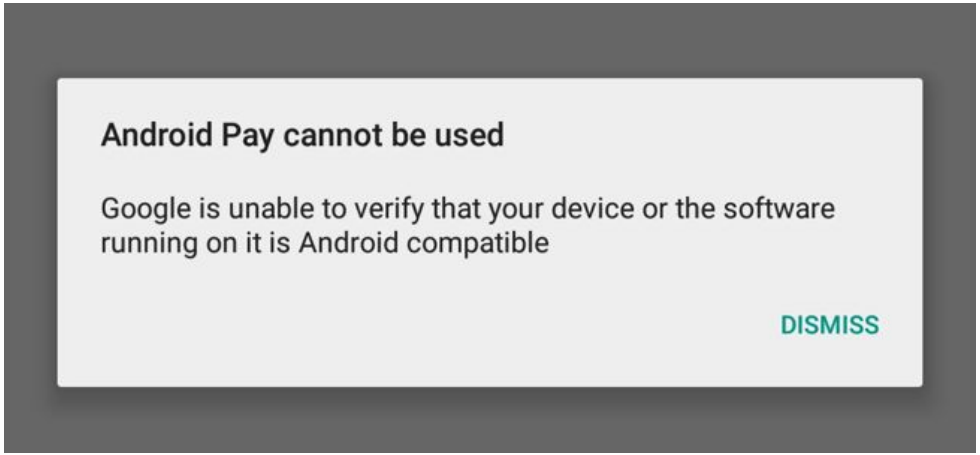
- SafetyNet is the brandname for security services on Android
  - Verified Apps, check for installed PHAs, **Attestation**, captcha service

- Designed to be run on any Android device (with Google Play)
  - Part of Google Play Services… **independant from device manufacturer**

- SafetyNet Attestation → remote **device** and **app** attestation

Google Apps & SafetyNet Attestation

- Google makes heavy use of Attestation for their own services
  - They can't control other device manufacturers
  - Needed a way determine security of a device
  - Method for fast reaction without waiting for software update (OS patches take even longer)



SafetyNet Attestation

- Remote Attestation of the **device** and the **app**
  - What every app used to implement themselves
- Part of Google Play Services
  - Independent of manufacturer, present on every Google Play enabled device
  - Dynamic code downloaded for attestation (without update of PlayServices)
    - Reference: https://koz.io/inside-safetynet
- **Just call an API to validate app and device are benign**

SafetyNet Attestation: just call an API, right???

- Docs: https://developer.android.com/training/safetynet/attestation.html
  - Not very detailed, even less so when I started working on SafetyNet stuff 1,5 years ago

SafetyNet Attestation API

The SafetyNet Attestation API helps you assess the security and compatibility of the Android environments in which your apps run. You can use this API to analyze devices that have installed your app.

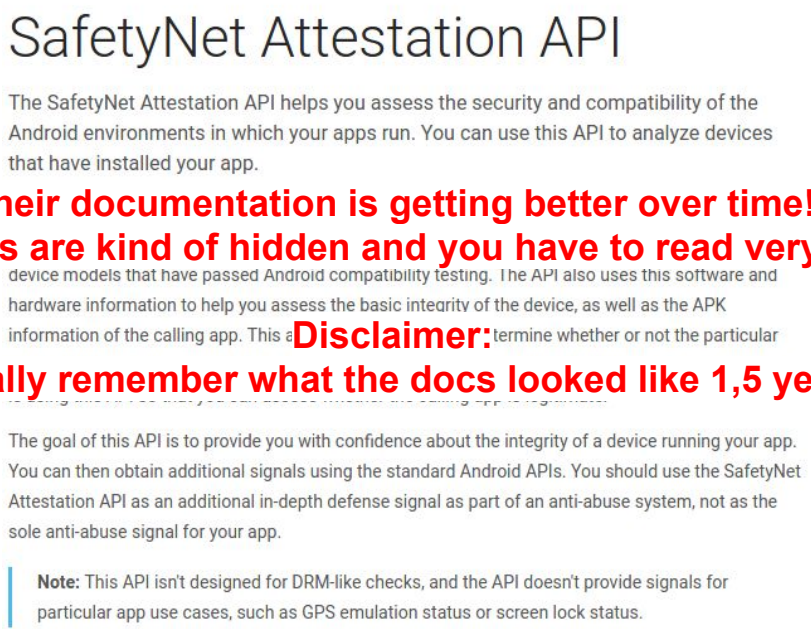
SafetyNet examines software and hardware information on the device where your app is installed to create a profile of that device. The service then attempts to find this same profile within a list of device models that have passed Android compatibility testing. The API also uses this software and hardware information to help you assess the basic integrity of the device, as well as the APK information of the calling app. This attestation helps you to determine whether or not the particular device has been tampered with or otherwise modified. It also provides information about the app that is using this API so that you can assess whether the calling app is legitimate.

The goal of this API is to provide you with confidence about the integrity of a device running your app. You can then obtain additional signals using the standard Android APIs. You should use the SafetyNet Attestation API as an additional in-depth defense signal as part of an anti-abuse system, not as the sole anti-abuse signal for your app.

> **Note:** This API isn't designed for DRM-like checks, and the API doesn't provide signals for particular app use cases, such as GPS emulation status or screen lock status.

SafetyNet Attestation: just call an API, right???

- Docs: https://developer.android.com/training/safetynet/attestation.html
  - Not very detailed, even less so when I started working on SafetyNet stuff 1,5 years ago



SafetyNet Attestation API

The SafetyNet Attestation API helps you assess the security and compatibility of the Android environments in which your apps run. You can use this API to analyze devices that have installed your app.

Their documentation is getting better over time!
Some details are kind of hidden and you have to read very carefully.

device models that have passed Android compatibility testing. The API also uses this software and hardware information to help you assess the basic integrity of the device, as well as the APK information of the calling app. This a**Disclaimer:**termine whether or not the particular

I don't really remember what the docs looked like 1,5 years ago.

The goal of this API is to provide you with confidence about the integrity of a device running your app. You can then obtain additional signals using the standard Android APIs. You should use the SafetyNet Attestation API as an additional in-depth defense signal as part of an anti-abuse system, not as the sole anti-abuse signal for your app.

> **Note:** This API isn't designed for DRM-like checks, and the API doesn't provide signals for particular app use cases, such as GPS emulation status or screen lock status.

Running the Attestation

Official example uses a callback...

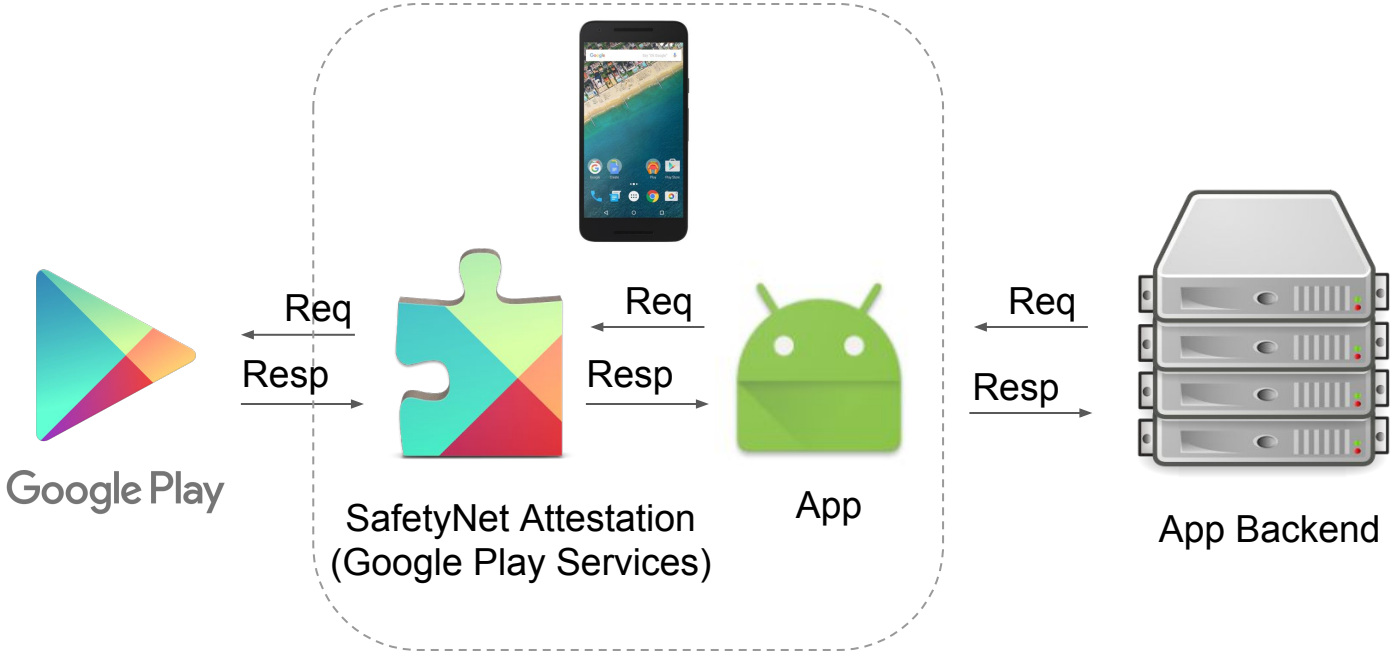
This example makes a synchronous call:

```
protected ConnectionResult connectGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(mContext).addApi(SafetyNet.API).build();

    return mGoogleApiClient.blockingConnect(10, TimeUnit.SECONDS);
}

public SafetyNetApi.AttestationResult attestation(byte[] nonce) {
    PendingResult<SafetyNetApi.AttestationResult> res = SafetyNet.SafetyNetApi.attest(mGoogleApiClient, nonce);

    return res.await(60, TimeUnit.SECONDS);
}
```

Note: this is example does NOT use an API Key, the API Key prevents rate limiting.

SafetyNet Attestation: Overview

SafetyNet Attestation: Call Chain



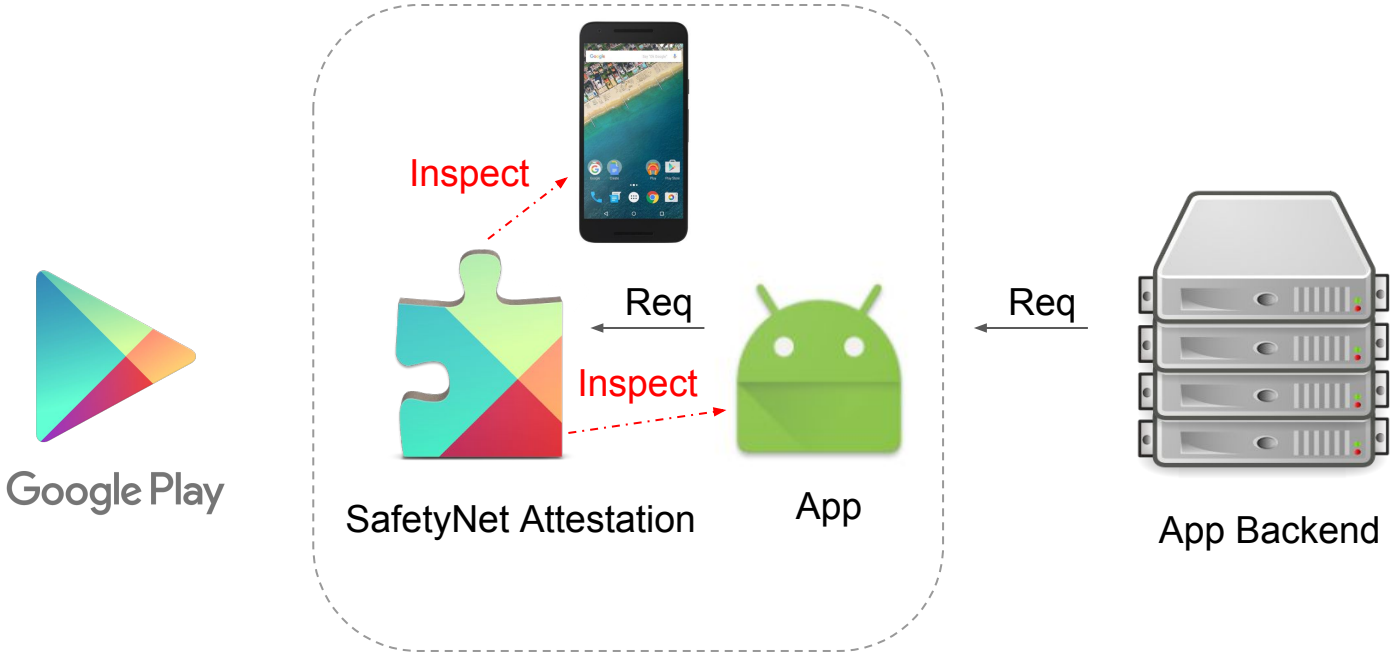
SafetyNet Attestation: Request Attestation

SafetyNet and the Nonce

Nonce → number used once

- **Prevent replay and reuse of attestation result**
  - **Also sharing between users/devices...**
- Nonce needs to be unique (used once!)
- Derive from account information or transaction information

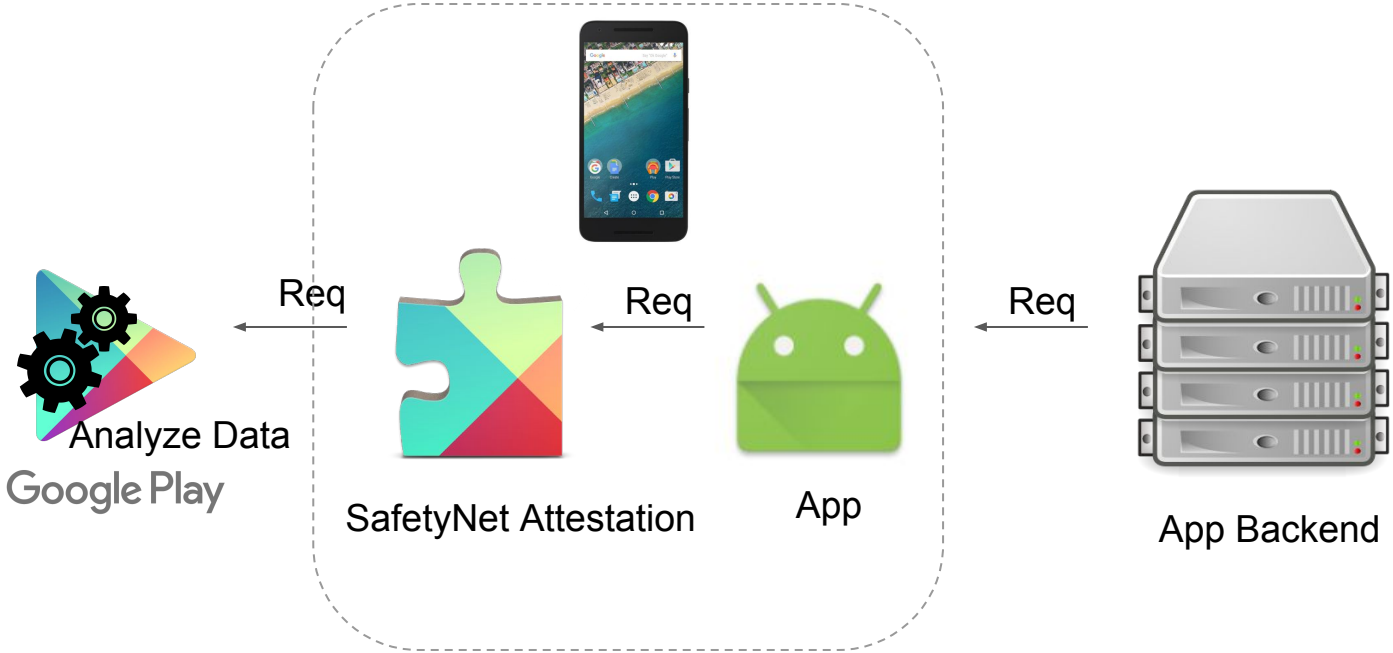
SafetyNet Attestation Overview: Request Attestation



This is what every app used to implement for themselves

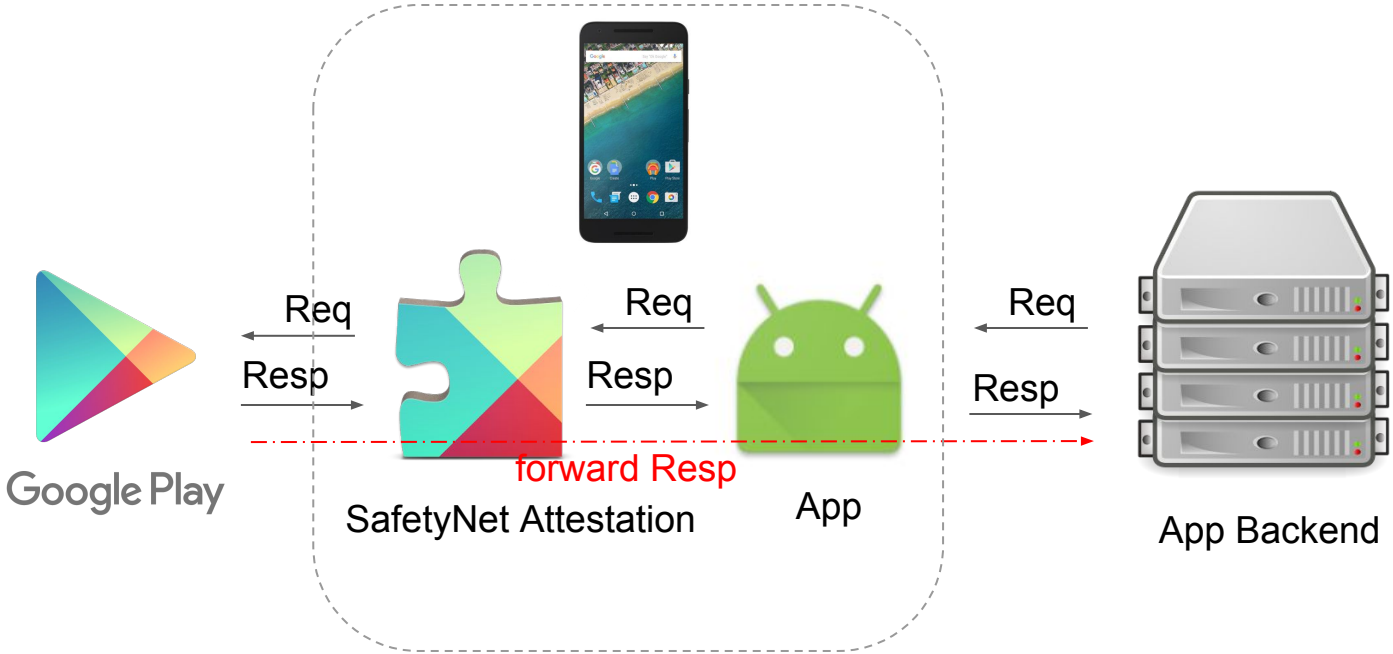
SafetyNet Attestation: Forward Data

SafetyNet Attestation: Attest Device & App



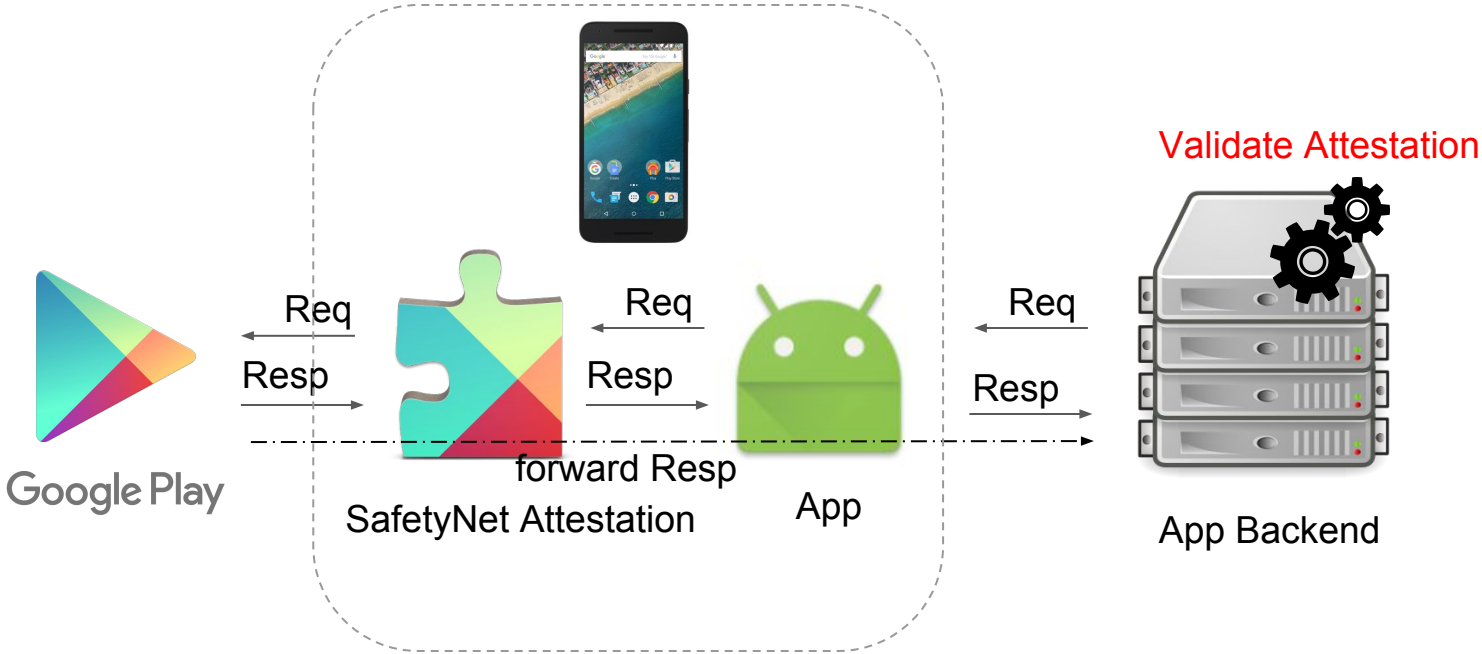
SafetyNet Attestation: Deliver Result

SafetyNet Attestation: Deliver Result



Response is protected with a cryptographic signature

SafetyNet Attestation: Validate Attestation



Attestation Result

Format: JSON Web Signature (JWS)

eyJhbGciOiJSUzI1NiIsIng1YyI6WyJNSUlFZmpDQ0EyYWdBd0lCQWdJSVZaeDlNZDVhb3JVd0RRWUpLb1pJaHZjTkFRUxCUUF3U1RFTE1Ba0dBMVVFQmhNQ1ZWTXhFekFSQmdOVkJBb1RDa2R2YjJkc1pTQkpibU14SlRBakJnTlZCQU1USE
VkdmIyZHNaU0JKYm5SbGNtNWxkQ0JCZFhSb2IzSnBkSGtnUnpJd0hoY05NVFV3T0RNeE1qQXpOalE0V2hjTk1UWXdPRE13TURBd01EQXdXakJzTVFzd0NRWURWUVFHRXdKVlV6RVRNQkVHQTFVRUNBd0tRMkZzYVdadmNtNXBZVEVXTUJRR0Ex
VUVCd3dOVFc5MWJuUmhhVzRnVm1sbGR6RVRNQkVHQTFVRUNnd0tSMjl2WjJ4bElFbHVZekViTUJrR0ExVUVBd3dTWVhSMFpYTjBMbUZ1WkhKdmFXUXVZMjl0TUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEFNSUlCQ2dLQ0FRRUEzaW
pVemNKOHl3NmhlYnpiQTRYbDJsOTM0dG96SFYyNWdJZ2VMNnU0eWVNNE4yMTh4WitPMWhkelBLbmR6bjArc1VuUHNTekl6SWZiMzV3Nk9xRDlxLysydlk5OUN3T2c0RXF2QXU2OTV1ZjVibzFjNk4rcHpNOWRWMDZIR3dSdUUxUE1OY2Y4Y01C
UEJDZy9jWmo2bUlsbFdGVXFERlFmVE5tL25vU0lucmg2WUpUOWhvdUJ6U2d5ZE1Kb2NsYnZEdjlEcThFQ1lWUVhFanA4Z00yYWNnOTNTZXhjb2xmZCtLVUFrNXdkaVBTeXhINFVRaDFvV25iMFR1bzJzeUpQZHh1cWQ3MVRFd1NweE5wcDZxZE
Ficy9XNE8vZ2swMVVxWEVqbFZvaFhmSE1sbHZsZEd5dWhEM0Z0dFIzOEFEb0dRaWVUVnlzK2VaZWY3ZXYzem9uNFFJREFRQUJvNElCUlRDQ0FVRXdIUVlEVlIwbEJCWXdGQVlJS3dZQkJRVUhBd0VHQ0NzR0FRVUZCd01DTUIwR0ExVWRFUVFX
TUJTQ0VtRjBkR1Z6ZEM1aGJtUnliMmxrTG1OdmJUQm9CZ2dyQmdFRkJRY0JBUVJjTUZvd0t3WUlLd1lCQlFVSE1BS0dIMmgwZEhBNkx5OXdhMmt1WjI5dloyeGxMbU52YlM5SFNVRkhNaTVqY25Rd0t3WUlLd1lCQlFVSE1BR0dIMmgwZEhBNk
x5OWpiR2xsYm5Sek1TNW5iMjluYkdVdVkyOXRMMjlqYzNBd0hRWURWUjBPQkJZRUZIVGh6cHVGbTNYcGs5c2xScDlRLzNSTGVNK2NNQXdHQTFVZEV3RUIvd1FDTUFBd0h3WURWUjBqQkJnd0ZvQVVTdDBHRmh1ODltaTFkdldCdHJ0aUdycGFn
Uzh3RndZRFZSMGdCQkF3RGpBTUJnb3JCZ0VFQWRaNUFnVUJNREFHQTFVZEh3UXBNQ2N3SmFBam9DR0dIMmgwZEhBNkx5OXdhMmt1WjI5dloyeGxMbU52YlM5SFNVRkhNaTVqY213d0RRWUpLb1pJaHZjTkFRUxCUUFEZ2dFQkFENkxLN25UZl
haUzZEMTg1ZlQvencxVGp0SUxOditrYlE3bVJZT2Z6dzY5bW1xWGNaeFppZllsNXRsdWVNZ0xzWFNFOWJQRXNKZk9hZzJLSnFiTVhXUUpGR1F5cmJ1OGszeDZXNDEvNWkzdUl6ZWsvTm5hZ00yV2hmK2lYcWcrdkxmakgyVlJoRmtQQ2k4Z21D
TDZneEZidm5ldUd5UlpyMEErS3NOUUxMMW1SQ3RjLzZRYWF0ZWV5Uy9TMmVGcVJaT2NJN2hpak95QTdvRUo4ZDNJMnlOZXdJSmlWd2dMZDNmYWRyekpwVmFyN1ZRR21jRnJUK0doVnpHSld4U1E0VEQzdUhZY0hZTAwR2VYUVoxMms3SEtEWD
RpRUNrek9jMEtXbG1WVXNXMXRrMTJnTitXQXlkM0QrVkdhV1lwQjNYeWd4VytTd3JrSkZoalpOaURBRkE9IiwiTUlJRDhEQ0NBdGlnQXdJQkFnSURBanFETUEwR0NTcUdTSWIzRFFFQkN3VUFNRU4Q3pBSkJnTlZCQVlUQWxWVE1SWXdGQVlE
VlFRS0V3MUhaVzlVY25WemRDQkpibU11TVJzd0dRWURWUVFERXhKSFpXOVVjblZ6ZENCSGJHOW1ZV3dnUTBFd0hoY05NVE13TkRBMU1UVXhOVFUyV2hjTk1UWXhNak14TWpNMU9UVTVXakJKTVFzd0NRWURWUVFHRXdKVlV6RVRNQkVHQTFVRU
NoTUtSMjl2WjJ4bElFbHVZekVsTUNNR0ExVUVBeE1jUjI5dloyeGxJRWx1ZEdWeWJtVjBJRUYxZEdodmNtbDBlU0JITWpDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBSndxQkhkYzJGQ1JPZ2FqZ3VEWVVFaThp
VC94R1hBYWlFWis0SS9GOFluT0llNWEvbUVOdHpKRWlhQjBDMU5QVmFUT2dtS1Y3dXRaWDhiaEJZQVN4RjZVUDd4YlNEajBVL2NrNXZ1UjZSWEV6L1JURGZSSy9KOVUzbjIrb0d0dmg4RFFVQjhvTUFOQTJnaHpVV3gvL3pvOHB6Y0dqcjFMRV
FUcmZTVGU1dm44TVhIN2xOVmc4eTVLcjBMU3krckVhaHF5ekZQZEZVdUxIOGdaWVIvTm5hZytZeXVFTldsbGhNZlp4VVlpK0ZPVnZ1T0FTaERHS3V5Nmx5QVJ4em1aRUFTZzhRRjZsU1dNVGxKMTRyYnRDTW9VL000aWFyTk96MFlEbDVjRGZz
Q3gzbnV2UlRQUHVqNXh0OTcwSlNYQ0RUV0puWjM3RGhGNWlSNDN4YStPY21rQ0F3RUFBYU9CNXpDQjVEQWZCZ05WSFNNRUdEQVdnQlRBZXBob2pZbjdxd1ZrREJGOXFuMWx1TXJNVGpBZEJnTlZIUTRFRmdRVVN0MEdGaHU4OW1pMWR2V0J0cn
RpR3JwYWdTOHdEZ1lEVlIwUEFRSC9CQVFEQWdFR01DNEdDQ3NHQVFVRkJ3RUJCQ013SURBZUJnZ3JCZ0VGQlFjd0FZWVNhSFIwY0RvdkwyY3VjM2x0WTJRdVkyOXRNQklHQTFVZEV3RUIvd1FJTUFZQkFmOENBUUF3T1FZRFZSMGZCQzR3TERB
cW9DaWdKb11rYUhSMGNEb3ZMMmN1YzNsdFkySXVZMjl0TDJOeWJITXZaM1JuYkc5aVlXd3VZM0pzTUJjR0ExVWRJQVFRTUE0d0RBWUtLd1lCQkFIV2VRSUZBVEFOQmdrcWhraUc5dzBCQVFzRkFBT0NBUUVBcXZxcElNMXFaNFB0WHRSKzNoM0
VmK0FsQmdERkpQdXB5QzF0ZnQ2ZGdtVXNnV00wWmo3cFVzSUl0TXN2OTErWk9tcWNVSHFGQll4OTBTcEloTk1KYkh6Q3pUV2Y4NEx1VXQ1b1grUUFpaGNnbHZjcGpacE55NmplaHNnTmIxYUhBMzBEUDl6NmVYMGhHZm5JT2k5UmRvekhRWkp4
anlYT04vaEtUQUFqNzhRMUVLN2dJNEJ6ZkUwMExzaHVrTllRSHBtRWN4cHc4dTFWRHU0WEJ1cG43akxyTE4xbkJ6LzJpOEp3M2xzQTVyc2Iwe1lhSW14c3NEVkNiSkFKUFpQcFpBa2lEb1VHbjhKekIkUG1YNERrallVaU9uTURzV0NPcm1qaT
lENlg1MkFTQ1dnMjNqclc0a09WV3plQmtvRWZ1NDNYclZKa0ZsZVcyVjQwZnNnMTJBPT0iXX0.eyJub25jZSI6IjFYSlNLUDJqWXAxRk1abkVaWUk5RlE9PSIsInRpbWVzdGFtcE1zIjoxNDQ2NzYwMzgyMjQ3LCJhcGtQYWNrYWdlTmFtZSI6
ImNvbS5leGFtcGxlLnNhZmV0eW5ldHRlc3Quc2FmZXR5bmV0dGVzdCIsImFwa0RpZ2VzdFNoYTI1NiI6Imh6TGJPSWlYYURSLzVRM014MVljNTQyV29OT2lnc3V1MEhwWFJFYTRqU0k9IiwiY3RzUHJvZmlsZU1hdGNoIjp0cnVlLCJleHRlbn
Npb24iOiJDUjM3cjh1QVoya0ciLCJhcGtDZXJ0aWZpY2F0ZURpZ2VzdFNoYTI1NiI6WyJmM2ZrbHp5Q1BPMXo5LzB6bytoR29haE8rcE9nWGR6UW5adnk5blFDQ2FvPSJdfQ.bZj8fZeWuByLg4u34S4Kr0wMsCQqJuLpvGjnGhzFKmSPzT2H
VUUPjCZ8IAtTg-XCP2eAcRr_FhEMaHthkUsw3OmCgw-V-dMb6IiJIcPiEvDfkeSqbLGkoXEWW8uqSxy0iXxLTNrNX20oIviCEznFvVgoBwZVLS7vtsKlAk8Fzb1Kmr2NiTcd1VqdvcoQ-cvqc-benqdJpYNcTE2Qp534B_nuimiC_ZJoKWpSAT
Ie5-Ge4CkOeHC1ilw76aWRyb7rh4GAchqs-_QDQucFTbZFpfK4q7-pDLgCtYiqgsiv8959lllooP8sHxRMd-d99rckkekUnGCdqbM8xyNmkgc8A

Attestation Result

Format: JSON Web Signature (JWS)

Cert Chain .

Attestation Data .

Signature

(base64 encoded)

eyJhbGciOiJSUzI1NiIsIng1YyI6WyJNSUlFZmpDQ0EyYWdBd0lCQWdJSVZaeDlNZDVhb3JVd0RRWUpLb1pJaHZjTkFRUxCUUF3U1RFTE1Ba0dBMVVFQmhNQ1ZWTXhFekFSQmdOVkJBb1RDa2R2YjJkc1pTQkpibU14SlRBakJnTlZCQU1USE
VkdmIyZHNaU0JKYm5SbGNtNWxkQ0JCZFhSb2IzSnBkSGtnUnpJd0hoY05NVFV3T0RNeE1qQXpOalE0V2hjTk1UWXdPRE13TURBd01EQXdXakJzTVFzd0NRWURWUVFHRXdKVlV6RVRNQkVHQTFVRUNBd0tRMkZzYVdadmNtNXBZVEVXTUJRR0Ex
VUVCd3dOVFc5MWJuUmhhVzRnVm1sbGR6RVRNQkVHQTFVRUNnd0tSMjl2WjJ4bElFbHVZekViTUJrR0ExVUVBd3dTWVhSMFpYTjBMbUZ1WkhKdmFXUXVZMjl0TUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEFNSUlCQ2dLQ0FRRUEzaW
pVemNKOHl3NmhlYnpiQTRYbDJsOTM0dG96SFYyNWdJZ2VMNnU0eWVNNE4yMTh4WitPMWhkelBLbmR6bjArc1VuUHNTekl6SWZiMzV3Nk9xRDlxLysydlk5OUN3T2c0RXF2QXU2OTV1ZjVibzFjNk4rcHpNOWRWMDZIR3dSdUUxUE1OY2Y4Y01C
UEJDZy9jWmo2bUlsbFdGVXFERlFmVE5tL25vU0lucmg2WUpUOWhvdUJ6U2d5ZE1Kb2NsYnZEdjlEcThFQ1lWUVhFanA4Z00yYWNnOTNTZXhjb2xmZCtLVUFrNXdkaVBTeXhINFVRaDFvV25iMFR1bzJzeUpQZHh1cWQ3MVRFd1NweE5wcDZxZE
Ficy9XNE8vZ2swMVVxWEVqbFZvaFhmSE1sbHZsZEd5dWhEM0Z0dFIzOEFEb0dRaWVUVnlzK2VaZWY3ZXYzem9uNFFJREFRQUJvNElCUlRDQ0FVRXdIUVlEVlIwbEJCWXdGQVlJS3dZQkJRVUhBd0VHQ0NzR0FRVUZCd01DTUIwR0ExVWRFUVFX
TUJTQ0VtRjBkR1Z6ZEM1aGJtUnliMmxrTG1OdmJUQm9CZ2dyQmdFRkJRY0JBUVJjTUZvd0t3WUlLd1lCQlFVSE1BS0dIMmgwZEhBNkx5OXdhMmt1WjI5dloyeGxMbU52YlM5SFNVRkhNaTVqY25Rd0t3WUlLd1lCQlFVSE1BR0dIMmgwZEhBNk
x5OWpiR2xsYm5Sek1TNW5iMjluYkdVdVkyOXRMMjlqYzNBd0hRWURWUjBPQkJZRUZIVGh6cHVGbTNYcGs5c2xScDlRLzNSTGVNK2NNQXdHQTFVZEV3RUIvd1FDTUFBd0h3WURWUjBqQkJnd0ZvQVVTdDBHRmh1ODltaTFkdldCdHJ0aUdycGFn
Uzh3RndZRFZSMGdCQkF3RGpBTUJnb3JCZ0VFQWRaNUFnVUJNREFHQTFVZEh3UXBNQ2N3SmFBam9DR0dIMmgwZEhBNkx5OXdhMmt1WjI5dloyeGxMbU52YlM5SFNVRkhNaTVqY213d0RRWUpLb1pJaHZjTkFRUxCUUFEZ2dFQkFENkxLN25UZl
haUzZEMTg1ZlQvencxVGp0SUxOditrYlE3bVJZT2Z6dzY5bW1xWGNaeFppZllsNXRsdWVNZ0xzWFNFOWJQRXNKZk9hZzJLSnFiTVhXUUpGR1F5cmJ1OGszeDZXNDEvNWkzdUl6ZWswTm5hZ00yV2hmK2lYcWcrdkxmakgyVlJoRmtQQ2k4Z21D
TDZneEZidm5ldUd5UlpyMEErS3NOUUxMMW1SQ3RjLzZRYWF0ZWV5Uy9TMmVGcVJaT2NJN2hpak95QTdvRUo4ZDNJMnlOZXdJSmlWd2dMZDNmYWRyekpwVmFyN1ZRR21jRnJUK0doVnpHSld4U1E0VEQzdUhZY0hZTAwR2VYUVoxMms3SEtEWD
RpRUNrek9jMEtXbG1WVXNXMXRrMTJnTitXQXlkM0QrVkdhV1lwQjNYeWd4VytTd3JrSkZoalpOaURBRkE9IiwiTUlJRDhEQ0NBdGlnQXdJQkFnSURBanFETUEwR0NTcUdTSWIzRFFFQkN3VUFNRU14Q3pBSkJnTlZCQVlUQWxWVE1SWXdGQVlE
VlFRS0V3MUhaVzlVY25WemRDQkpibU11TVJzd0dWURWUVFERXhKSFpXOVVjblZ6ZENCSGJHOW1ZV3dnUTBFd0hoY05NVE13TkRBMU1UVXhOVFUyV2hjTk1UWXhNak14TWpNMU9UVTVXakJKTVFzd0NRWURWUVFHRXdKVlV6RVRNQkVHQTFVRU
NoTUtSMjl2WjJ4bElFbHVZekVsTUNNR0ExVUVBeE1jUjI5dloyeGxJRWx1ZEdWeWJtVjBJRUYxZEdodmNtbDBlU0JITWpDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBSndxQkhkYzJGQ1JPZ2FqZ3VEWVVFaThp
VC94R1hBYWlFWis0SS9GOFluT0llNWEvbUVOdHpKRWlhQjBDMU5QVmFUT2dtS1Y3dXRaWDhiaEJZQVN4RjZVUDd4YlNEajBVL2NrNXZ1UjZSWEV6L1JURGZSSy9KOVUzbjIrb0d0dmg4RFFVQjhvTUFOQTJnaHpVV3gvL3pvOHB6Y0dqcjFMRV
FUcmZTVGU1dm44TVhIN2xOVmc4eTVLcjBMU3krckVhaHF5ekZQZEZVdUxIOGdaWVIvTm5hZytZeXVFTldsbGhNZlp4VVlpK0ZPVnZ1T0FTaERHS3V5Nmx5QVJ4em1aRUFTZzhRRjZsU1dNVGxKMTRyYnRDTW9VL000aWFyTk96MFlEbDVjRGZz
Q3gzbnV2UlRQUHVqNXh0OTcwSlNYQ0RUV0puWjM3RGhGNWlSNDN4YStPY21rQ0F3RUFBYU9CNXpDQjVEQWZCZ05WSFNNRUdEQVdnQlRBZXBob2pZbjdxd1ZrREJGOXFuMWx1TXJNVGpBZEJnTlZIUTRFRmdRVVN0MEdGaHU4OW1pMWR2V0J0cn
RpR3JwYWdTOHdEZ1lEVlIwUEFRSC9CQVFEQWdFR01DNEdDQ3NHQVFVRkJ3RUJCQ0l3SURBZUJnZ3JCZ0VGQlFjd0FZWVNhSFIwY0RvdkwyY3VjM2x0WTJRdVkyOXRNQklHQTFVZEV3RUIvd1FJTUFZQkFmOENBUUF3T1FZRFZSMGZCQzR3TERB
cW9DaWdKb11rYUhSMGNEb3ZMMmN1YzNsdFkySXVZMjl0TDJOeWJITXZaM1JuYkc5aVlXd3VZM0pzTUJjR0ExVWRJQVFRTUE0d0RBWUtLd1lCQkFIV2VRSUZBVEFOQmdrcWhraUc5dzBCQVFzRkFBT0NBUUVBcXZxcElNMXFaNFB0WHRSKzNoM0
VmK0FsQmdERkpQdXB5QzF0ZnQ2ZGdtVXNnV00wWmo3cFVzSUl0TXN2OTErWk9tcWNVSHFGQll4OTBTcEloTk1KYkh6Q3pUV2Y4NEx1VXQ1b1grUUFpaGNnbHZjcGpacE55NmplaHNnTmIxYUhBMzBEUDl6NmVYMGhHZm5JT2k5UmRvekhRWkp4
anlYT04vaEtUQUFqNzhRMUVLN2dJNEJ6ZkUwMExzaHVrTllRSHBtRWN4cHc4dTFWRHU0WEJ1cG43akxyTE4xbkJ6LzJpOEp3M2xzQTVyc2Iwe1lhSW14c3NEVkNiSkFKUFpQcFpBa2lEb1VHbjhKeklkUG1YNERrallVaU9uTURzV0NPcm1qaT
lENlg1MkFTQ1dnMjNqclc0a09WV3plQmtvRWZ1NDNYclZKa0ZsZVcyVjQwZnNnMTJBPT0iXX0.eyJub25jZSI6IjFYSlNLUDJqWXAxRk1abkVaWUk5RlE9PSIsInRpbWVzdGFtcE1zIjoxNDQ2NzYwMzgyMjQ3LCJhcGtQYWNrYWdlTmFtZSI6
ImNvbS5leGFtcGxlLnNhZmV0eW5ldHRlc3Quc2FmZXR5bmV0dGVzdCIsImFwa0RpZ2VzdFNoYTI1NiI6Imh6TGJPSWlYYURSLzVRM014MVljNTQyV29OT2lnc3V1MEhwWFJFYTRqU0k9IiwiY3RzUHJvZmlsZU1hdGNoIjp0cnVlLCJleHRlbn
Npb24iOiJDUjM3cjh1QVoya0ciLCJhcGtDZXJ0aWZpY2F0ZURpZ2VzdFNoYTI1NiI6WyJmM2ZrbHp5Q1BPMXo5LzB6bytoR29haE8rcE9nWGR6UW5adnk5blFDQ2FvPSJdfQ.bZJj8fZeWuByLg4u34S4Kr0wMsCQqJuLpvGjnGhzFKmSPzT2H
VUUPjCZ8IAtTg-XCP2eAcRr_FhEMaHthkUsw3OmCgw-V-dMb6IiJIcPiEvDfkeSqbLGkoXEWW8uqSxy0iXxLTNrNX20oIviCEznFvVgoBwZVLS7vtsKlAk8Fzb1Kmr2NiTcd1VqdvcoQ-cvqc-benqdJpYNcTE2Qp534B_nuimiC_ZJoKWpSAT
Ie5-Ge4CkOeHC1ilw76aWRyb7rh4GAchqs-_QDQucFTbZFpfK4q7-pDLgCtYiqgsiv8959lllooP8sHxRMd-d99rckkekUnGCdqbM8xyNmkgc8A

Attestation Signature Validation

Google has a sig check API - this is only for dev purposes only!

Verify the compatibility check response

You should take steps to make sure that the compatibility check response actually came from the SafetyNet service and includes data that matches your request data.

> **Caution:** You should send the entire JWS response to your own server, using a secure connection, for verification. We don't recommend that you perform the verification directly in your app because, in that case, there is no guarantee that the verification logic itself hasn't been modified.

Follow these steps to verify the origin of the JWS message:

1. Extract the SSL certificate chain from the JWS message.
2. Validate the SSL certificate chain and use SSL hostname matching to verify that the leaf certificate was issued to the hostname `attest.android.com`.
3. Use the certificate to verify the signature of the JWS message.
4. Check the data of the JWS message to make sure it matches the data within your original request. In particular, make sure that the nonce, timestamp, package name, and the SHA-256 hashes match.

Attestation Data


```
{"extension":"CQzq39pKPKAo",
"ctsProfileMatch":true,
"apkPackageName":"org.mulliner.labs.selfaware",
"apkDigestSha256":"p3wl5Y07Ecxvb0CzVCLXW2L3nj\/TPzK0mIX3CA1QTM8=",
"nonce":"4yAtXmJk1NFtfPk2NPnW6Q==",
"apkCertificateDigestSha256":["IadGdZbBaGUP17YxtlQi61Y+HSGv8i3ec4m6DV1zh0g="],
"timestampMs":1471972187275,
"basicIntegrity":true}
```

Possible attestation results

The JWS message contains two fields that indicate the attestation check result: `"ctsProfileMatch"` and `"basicIntegrity"`. The status of the device running your app determines the value for each field, as shown in Table 1:

Table 1. How device status affects the values of `"basicIntegrity"` and `"ctsProfileMatch"`

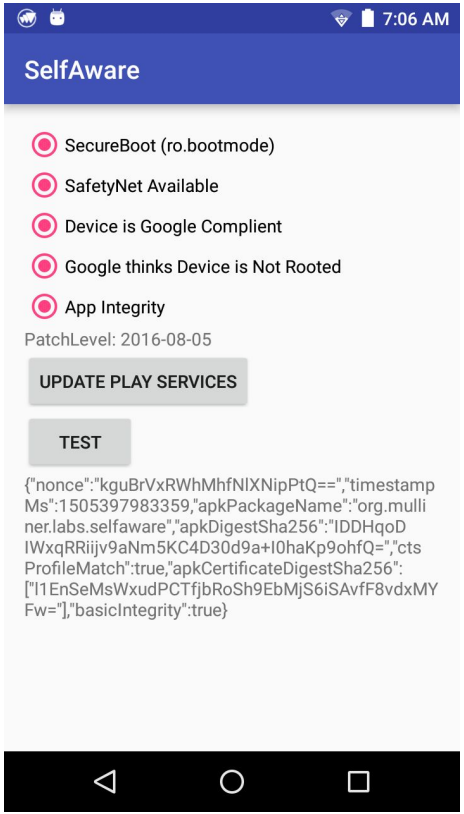
Device Status	Value of "ctsProfileMatch"	Value of "basicIntegrity"
Certified, genuine device that passes CTS	`true`	`true`
Certified device with unlocked bootloader	`false`	`true`
Genuine but uncertified device, such as when the manufacturer doesn't apply for certification	`false`	`true`
Device with custom ROM (not rooted)	`false`	`true`
Emulator	`false`	`false`
No device (protocol emulator script)	`false`	`false`
Signs of system integrity compromise, such as rooting	`false`	`false`
Signs of other active attacks, such as API hooking	`false`	`false`

Demo App

Device is Google compliant (CTS = true)

Not Rooted (basicIntegrity = true)

AppIntegrity (APK/ApkCert digest match)



Errors...

Error cases

The JWS message can also show several types of error conditions:

- A `null` result indicates that the call to the service didn't complete successfully.
- An `"error"` field indicates that an issue occurred, such as a network error or an error that an attacker feigned. Most errors are transient and should be absent if you retry the call to the service. You may want to retry a few more times with increasing delays between each retry. Keep in mind, however, that if you trigger more than 5 calls per minute, you could exceed the rate limit, which causes the remaining requests during that minute to return an error automatically.

> **Note:** If an error occurs, the result cannot represent a passed test, as an attacker might intentionally trigger such an error.

Error (no example for this in their docs)

```
{"extension":"CYOUMWN1YUXN",

"Error":"internal_error",

"apkCertificateDigestSha256":[]}"
```

This means the API works but the attestation failed to run!

Just a side note in their documentation!

Untrusted Device / Can’t determine calling APK?


```
{"extension":"CaOav6U9qRO1",

"ctsProfileMatch":false,

"nonce":"Ehq+1HB3KyRWAT8zv\/vDmw==",

"apkCertificateDigestSha256":[],

"timestampMs":1471950172731,

"basicIntegrity":false}
```


No actual example in their docs!

The package name and APK digests are missing!

Again this is a side note in their documentation.

That’s how SafetyNet Attestation works!

- YOUR App’s backend requests attestation
- YOUR App initiates attestation
- PlayServices attests
- YOUR App’s backend validates the attestation → grant/deny access

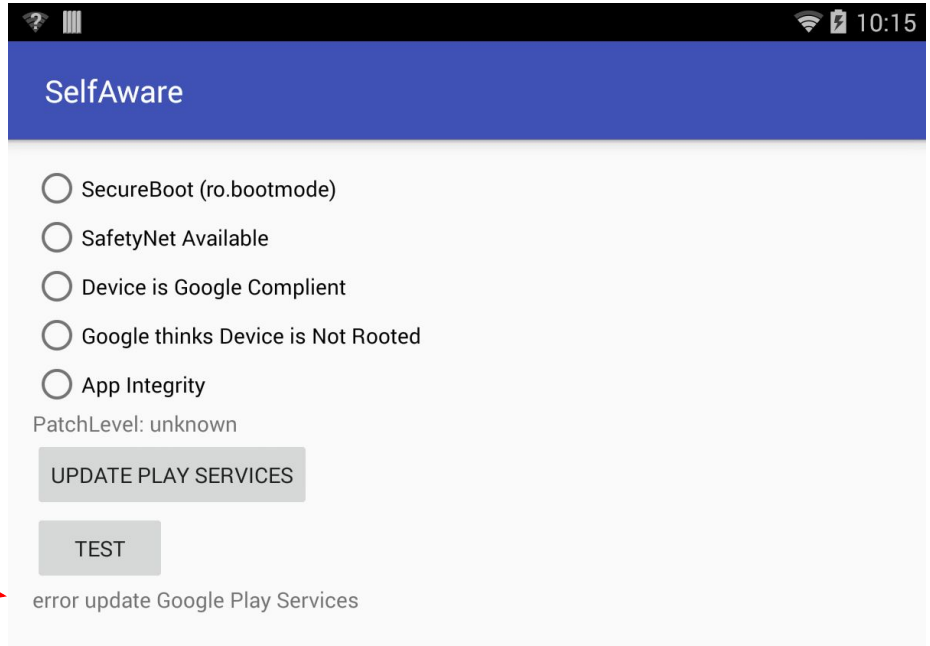
You wish, there is more!

Attestation: just an API Call away!?

- **All API calls can and WILL fail in the wild!**
  - Solution: report failure codes to your backend (only you can decide what to do)
- Connection to GoogleApiClient fails
  - General connection error → retry
  - Error code 2 → Google PlayServices doesn't support SafetyNet → UPDATE PlayServices
- SafetyNet attest() call fails
  - Nonce too short (SHOULD NOT HAPPEN TO YOU)
  - Rate limited (add API_KEY + request bigger quota)
  - **Generic error → this will happen to you**

PlayServices too old

Android 4.4 no SecureBoot!



API Failures...

- **Start with retrying everything** (generic errors and network errors!)
  - Be a good citizen and use exponential backoff!

- attest()
  - Inspect attestation result on the client to determine if JSON error field is present → base64 decode → parse json → error field present?
    - YES → retry

- If everything fails report to your backend … app specific behavior :-(
  - Have a plan for handling this otherwise I'll just "report an error and bypass your check"

Howto: App/APK Integrity

apkDigestSha256 and **apkCertificateDigestSha256**

- SHA-256 of your APK and the Certificate you signed your APK with

Easy mode:

- **APK Certificate Digest** is always the same (always signed with same cert)
  - Can hard code into your backend (you only have one data point)

If you have this you have APK integrity!

Howto: App/APK Integrity

apkDigestSha256

Advanced mode:

- Collect all APK Digests and compare against database

Features:

- Your devs can sign apps but don’t control APK digest database → you control what versions are allowed to speak to your backend
- Revoke APK versions by digest

WARNING: Need to have total control over your release process!

Implementation & Deployment Summary

Client

- Check error conditions and retry, report failure codes to backend

Backend

- Validate signature and attestation data
- Check all fields including timestamp and nonce

Make decision for failures that prevent attestation to happen (important!!!)

- Ask user to update PlayServices, have whitelisting mechanism for customers

Can we Trust SafetyNet Attestation?

I wanted to know how far we can trust this system

- Limitations (e.g. Android versions)
- Attacks & Bypasses

You really want to know how well your security system works!

SafetyNet vs. Android Versions

- Android 4 - Android 5
  - Can’t detect boot state (secure vs insecure)
  - roots/attacks that require an unlocked bootloader work
    - With limitations…

- Android 6 and up
  - Detect boot state and fail CTS on in-secure boot!

Android 4

- No Dm-Verity → root can remount and write files in /system

- SafetyNet Attestation inspects filesystem not running processes
  - Temp. move files such as “su” is enough to bypass it
    - Move /system/xbin/su to /data/local/tmp, run app (pass attest), restore su

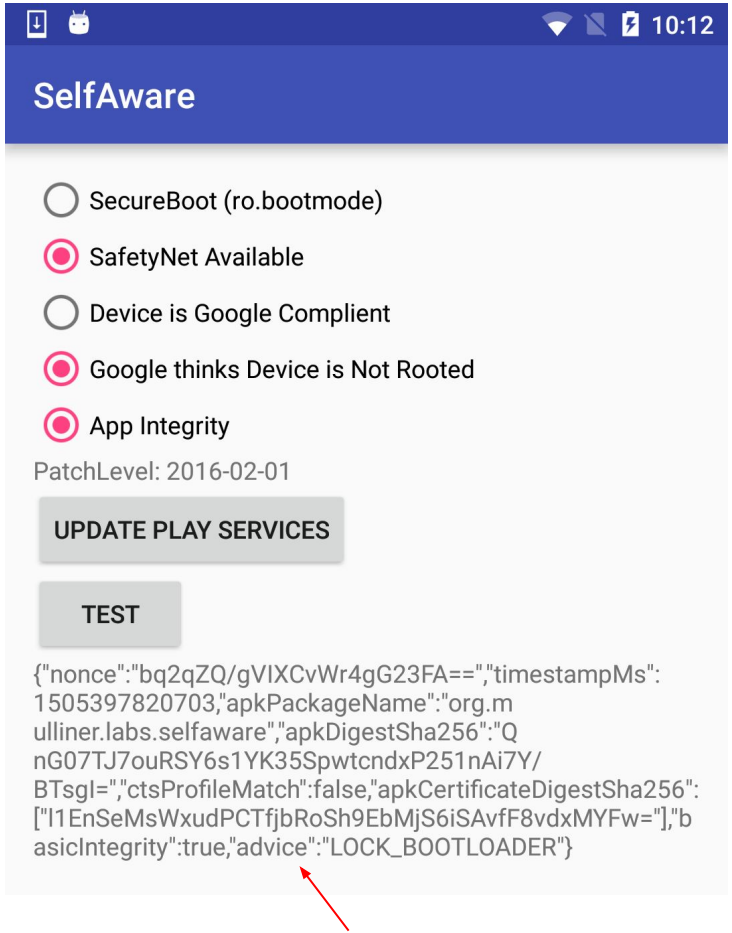
DEMO!

Boot Loader Unlocked

Nexus 5x with Android 6

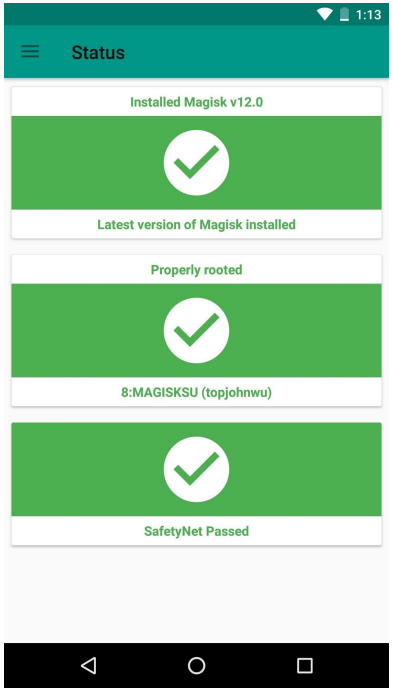
Note the advice field:

LOCK_BOOTLOADER



SuHide and Magisk

- SuHide was the first attempt to hide root from SafetyNet
  - *Reference: https://koz.io/hiding-root-with-suhide/*
- Magisk is the modern root that will bypass SafetyNet
  - Unlocked bootloader, selinux policy patch → all this is hidden
  - https://github.com/topjohnwu/Magisk
- Basically rootkit for Android
  - Need custom detections for those!
  - Google plays Cat'n Mouse with those



Attacking App Integrity

- So far all attacks target device integrity (CTS and basicIntegrity)
- Nobody looked at App integrity…

Let’s look at app integrity!

- Goal is modify app without triggering SafetyNet Attestation
  - APK + Cert Digest!
- Enables:
  - Modify app behavior
    - Remove TLS Cert-pinning (modify traffic)
    - ...

SafetyNet's App Integrity Checks

apkDigestSha256 and apkCertificateDigestSha256

- Calculated on the APK file on disk

Android doesn't execute the APK

- APK contains DEX files
- Until Android 4 DEX files are converted into ODEX (optimized byte code)
- Android 4.4/5 and later DEX files are compiled into native code

This can be attacked!

(Hiding behind ART by Paul Sabanal 2014 - rootkit via odex modification)

Running Code on Android

Android 4.4 and 5

- APK: /data/app/sa.apk
- Data: /data/data/org.mulliner.labs.selfaware/
- Code: /data/dalvik-cache/data@app@org.mulliner.labs.selfaware-1.apk@classes.dex
  - Owned by system

Android 6 and later

- APK: /data/app/org.mulliner.labs.selfaware-1/base.apk
- Data: /data/app/org.mulliner.labs.selfaware-1/
- Code: /data/app/org.mulliner.labs.selfaware-1/oat/ARM/base.odex ← native code
  - Owned by system and writable by installd

Running Code on Android

Android 4.4 and 5

- APK: /data/app/sa.apk
- Data: /data/data/org.mulliner.labs.selfaware/
- Code: /da classes.dex
  - Owned by sy

App can't read its own code on the disk.
Zygote loads it into memory.

Android 6 and later

- APK: /data/app/org.mulliner.labs.selfaware-1/base.apk
- Data: /data/app/org.mulliner.labs.selfaware-1/
- Code: /data/app/org.mulliner.labs.selfaware-1/oat/ARM/base.odex ← native code
  - Owned by system and writable by installd

ODEX Code Modification Attack: Overview (Generic)

- Actual code modification
  - Use apktool to unpack; MODIFY SMALI CODE; apktool to build APK; jarsigner to sign
    - Modified APK with wrong signature (but signature is not part of the ODEX file)
- Compile DEX code to ART code
  - Dex2oat --dex-file=sa.apk --oat-file=sa.odex
    - ODEX file based on modified APK
- Prevent the Android VM from re-compiling (aka patching the CRC32)
  - ODEX file contains CRC32 of DEX files it was generated from
  - Patch CRC32 in ODEX file to match the DEX code from the original DEX files in original APK
    - Made a tool for this!!!

Attacking ODEX files: all Android Versions

- Need to write ODEX files
  - Root device… any way to write those files will enable this attack!
- Overwrite ODEX files in dalvik cache
  - Android 4.4 /data/dalivk-cache
  - Android 6+ /data/app/APPNAME/oat/ARCH/base.odex
- Stop and start app → WIN
  - Tested on bunch of 4.4 and 6 devices
- Modification persists across reboots
  - Remove root (unroot)

Attacking ODEX files: all Android Versions

- Need to write ODEX files
  - Root device… any way to write those files will enable this attack!
- Overwrite ODEX files in dalvik cache
  - Andro
  - Android 6+ /data
- Stop and start app → WIN
  - Tested on bunch of 4.4 and 6 devices
- Modification persists across reboots
  - Remove root (unroot)

SafetyNet AppIntegrity is bypassed as checks are run on the APK!

Attacking ODEX files **without** Root (Android 6)

Goal: overwrite /data/app/org.mulliner.labs.selfaware/oat/arm/base.odex

Who can write?

- Users: system and installd (basically: installd and zygote)

Attacking ODEX files **without** Root (Android 6)

Goal: overwrite /data/app/org.mulliner.labs.selfaware/oat/arm/base.odex

Who can write?

- Users: system and installd (basically: installd and zygote)

Who else can write?

- Kernel → **dirtycow** (CVE-2016-5195)
  - Linux kernel bug that ultimately allowed writing ANY file that you can read

ODEX file Attack via Dirtycow

Same exact procedure as before!

File size is the only issue (dirtycow can’t write past file boundary, not append!)

- Patching the APK might add code
  - Remove code? → No!

Dex2Oat optimizes native code for the specific CPU
“--instruction-set=arm --instruction-set-variant=cortex-a53”

- **Trick: just don’t optimize the OAT file to make it small!**
  - I just run: dex2oat --dex-file=bad.apk --oat-file=patched.odex

DEMO: ODEX file Attack using Dirtycow

BLU device with Android 6 (also tested on Nexus 5x with Android 6)

- Works on every Android device with a kernel that is vulnerable to dirtycow
  - Should be plenty of Android devices

Overwrite the odex file via:

dirtycow base.odex /data/app/org.mulliner.labs.selfaware/oat/arm/base.odex

Remember: no root required!

Attack Impact

Limited to Android devices that are still vulnerable to dirtycow

- Likely many (I don’t have numbers)

Attack obviously goes bejoined SafetyNet Attestation

- Android 7 devices will not be vulnerable since dirtycow patch is required!

Notified Google over a year ago (about the generic attack), was told this is known!

CopperheadOS - hardened Android clone (www.copperhead.co)

- Mitigates by re-compiling apps before each start (can be slow)

Some Observations...

SafetyNet Attestation improves over Time

basicIntegrity (added ca. July 2016)

advice (added ca. May 2017)

{"nonce":"bq2qZQ/gVIXCvWr4gG23FA==","timestampMs":
1505397820703,"apkPackageName":"org.m
ulliner.labs.selfaware","apkDigestSha256":"Q
nG07TJ7ouRSY6s1YK35SpwtcndxP251nAi7Y/
BTsgI=","ctsProfileMatch":false,"apkCertificateDigestSha256":
["l1EnSeMsWxudPCTfjbRoSh9EbMjS6iSAvfF8vdxMYFw="],"b
asicIntegrity":true,"advice":"LOCK_BOOTLOADER"}

SafetyNet Attestation “Outage”

- Attestation is based on CTS data
  - CTS is run by manufacturers (including Google) for each OS release and patch
- Missing or false data → Attestation believes device is modified
- Google broke Attestation briefly for Nexus devices
  - I found Attestation was broken for YotaPhone with a specific security update (~1 year ago)

[Update: It's back] Google pulls March security update for Nexus 6, after it breaks SafetyNet and Android Pay

100

Corbin Davenport

Mar 10, 2017

G+102 f133 118

Total Shares 353

Google's Cat'n Mouse with the Rooting Community

- Google improves Attestation all the time

- Detect and lock out new rooting methods and tools
  - To protect their own apps such as AndroidPay

- Google improves the security of your app if you use SafetyNet
  - You don't have to do anything in addition!

SafetyNet Attestation is Free

- It can go down, no SLA
- It can rate limit, if over capacity
- It is FREE
  - Compare to 3rd party services that you have to pay for!

SafetyNet Attestation vs APK repackaging Malware

Malware commonly repackages Android apps

- Add malicious functionally while keeping the actual app running

AppIntegrty kills repacking because:

- PackageName change detected on backend (also make sure to check it)
- PackageName not changed → APK digest will change

Make sure to embed attestation result in protocol between your app and backend

- Prevent app from working if attestation fails (e.g., fail login)

Summary & Conclusions

SafetyNet Attestation is an essential platform security service on Android: USE IT!

- Implementation is relatively easy, I hope this talk helps!

Bypasses exist but depend on number of conditions

- The majority of apps will benefit!

It will only get better over time!

- Google is constantly improving the platform and thus SafetyNet Attestation

Questions?

Slides and material will be available soonish:

- https://www.mulliner.org/android

Checkout my blog:

- https://www.mulliner.org/blog

Bonus round: more Dirtycow fun!

Replace run-as with a version that can switch to an arbitrary selinux context

- Basic idea is not from me, I just added selinux context switching support!

New command: run-as UID SELinux_Context

- Access private data of any Android app (/data/app/org.mulliner.labs.selfaware/files/*)

ps -Z |grep mulliner → u:r:untrusted_app:s0:c512,c768 u0_a115 9835 276 org.mulliner.labs.selfaware

run-as 10115 u:r:untrusted_app:s0 → shell as org.mulliner.labs.selfaware

- App is built as release app (debuggable=false)

References:

https://mulliner.org/android/

https://developer.android.com/training/safetynet/attestation.html

https://koz.io/inside-safetynet

https://github.com/topjohnwu/Magisk

http://www.blackhat.com/docs/asia-15/materials/asia-15-Sabanal-Hiding-Behind-ART.pdf

Further reading (nice attack that uses ODEX patching):

https://www.fireeye.com/blog/threat-research/2017/05/gaining-root-on-lenovo-vibe.html