Fingerprint-Jacking:
Practical Fingerprint Authorization Hijacking in Android Apps

Xianbo Wang[1], Wing Cheong Lau[1], Yikang Chen[1], Shangcheng Shi[1], Ronghai Yang[2]

[1]*The Chinese University of Hong Kong,*

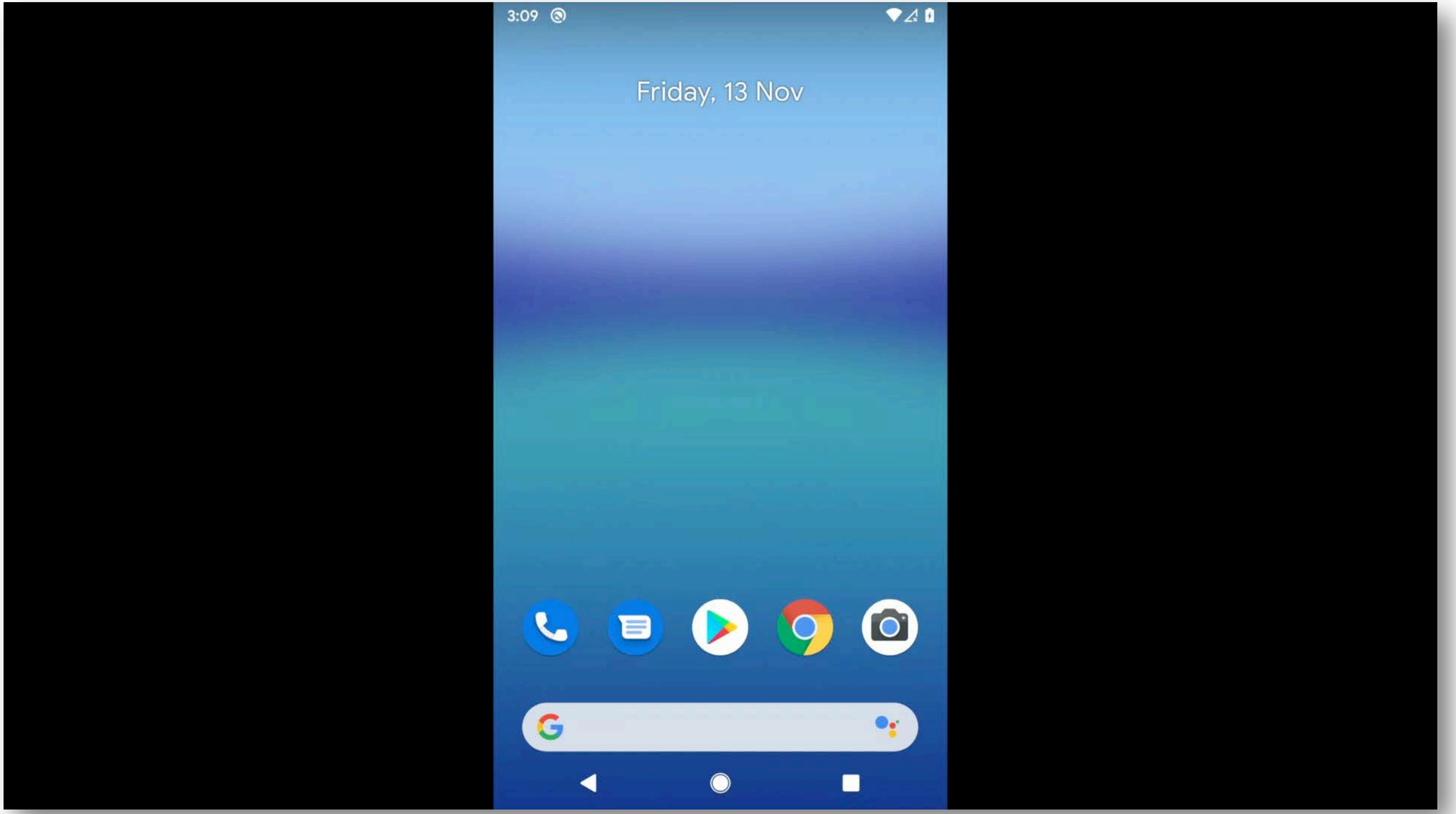
[2]*Sangfor Technologies Co., Ltd*

About Me

- 2nd Year PhD Student in The Chinese University of Hong Kong
- Our lab: MobiTeC. We did a number of works on OAuth security.
  - Check our talk on Blackhat Asia 2019 and Europe 2016 about OAuth security.
- Personal interest mostly on Web/Android app security.
- Love CTF, Bug Bounty, Pentest.

@sanebow

Demo Time!



Everyone knows clickjacking …

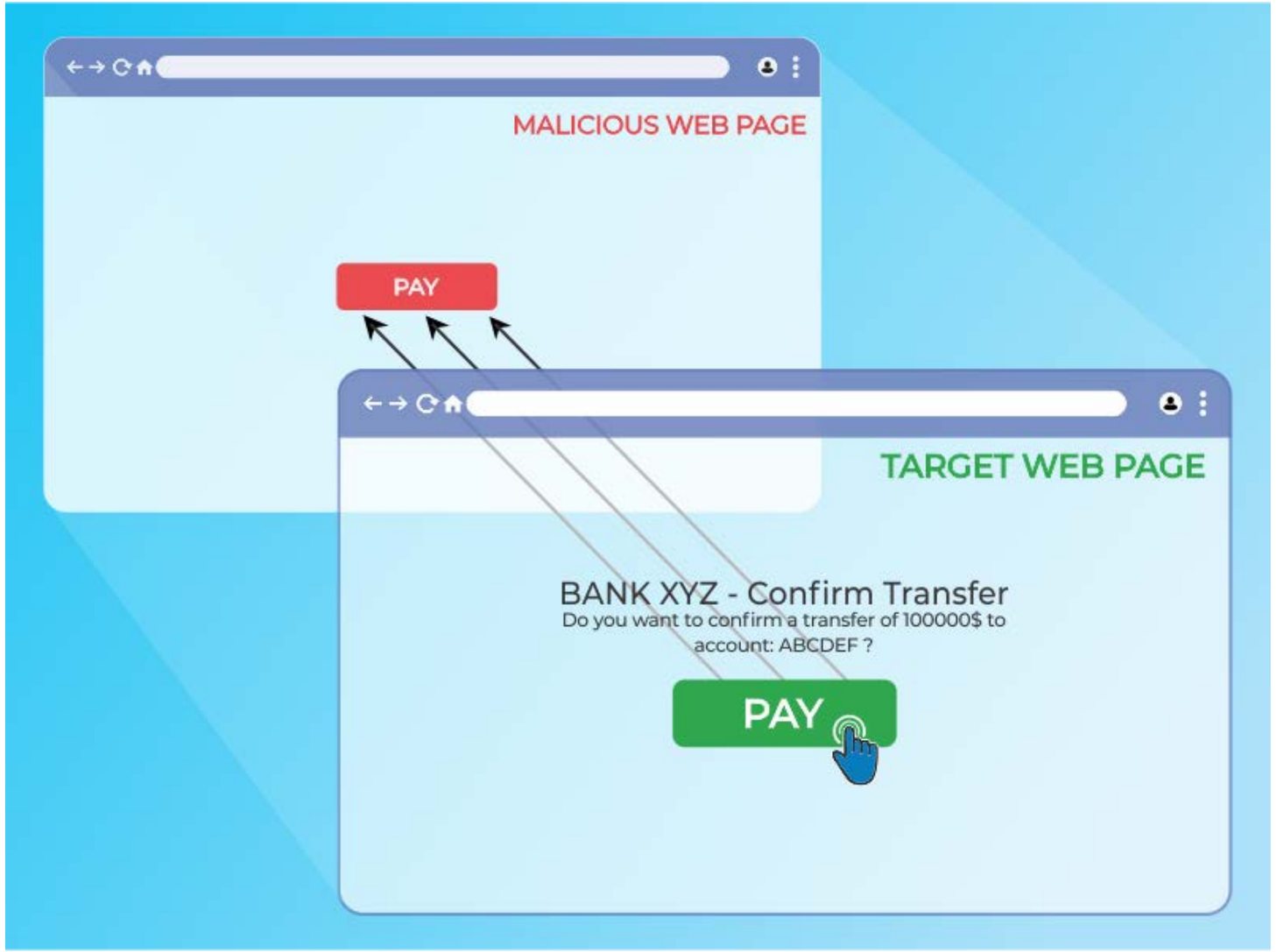
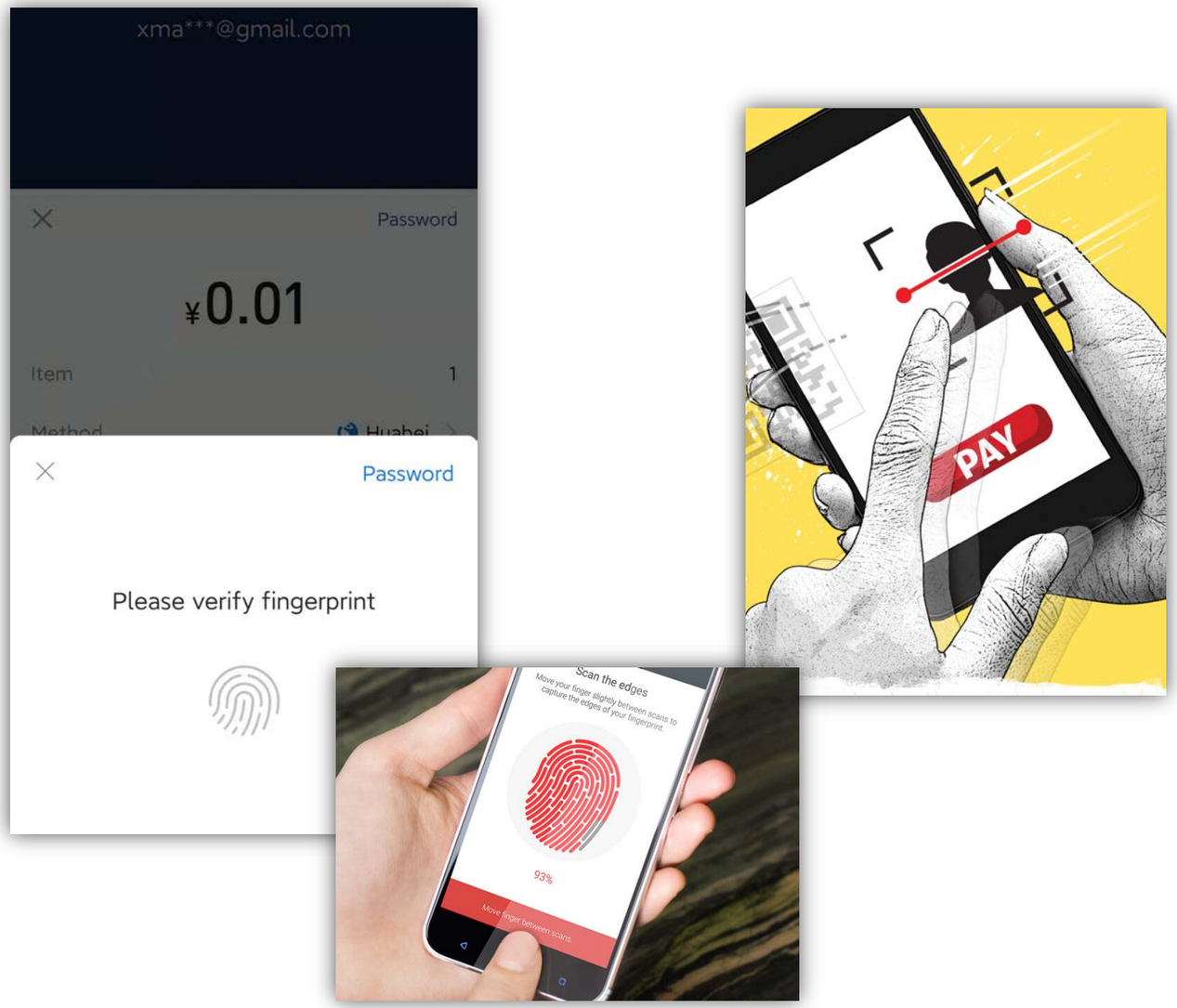


Image from: https://owasp.org

Nowadays, we use our fingerprint everywhere



So, fingerprint-jacking is:

- A UI attack (targeting only Android devices in this talk)
- For hijacking your fingerprint inputs

It can deceive users into:

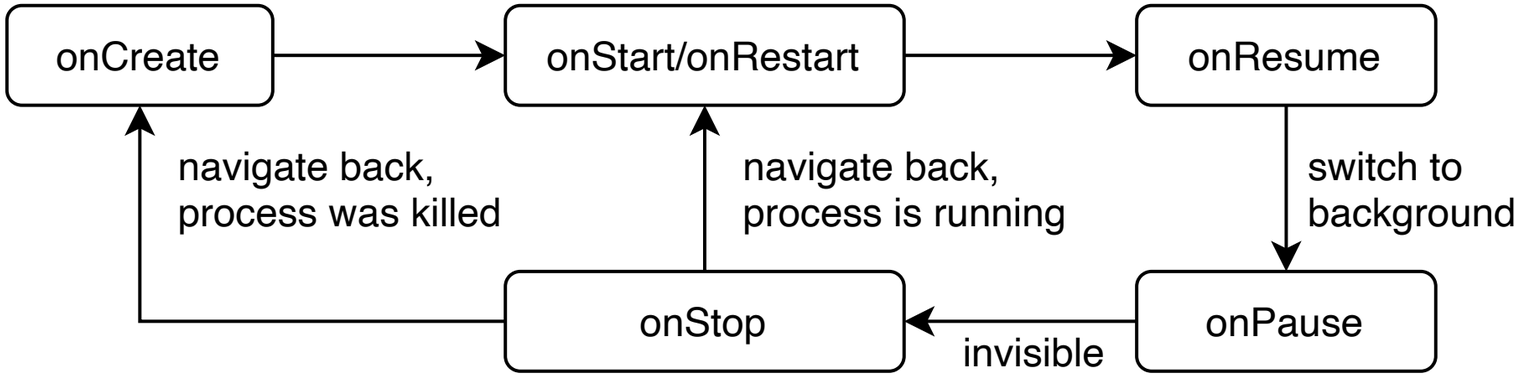
- Authorizing dangerous actions **unknowingly**

Interesting findings we want to share:

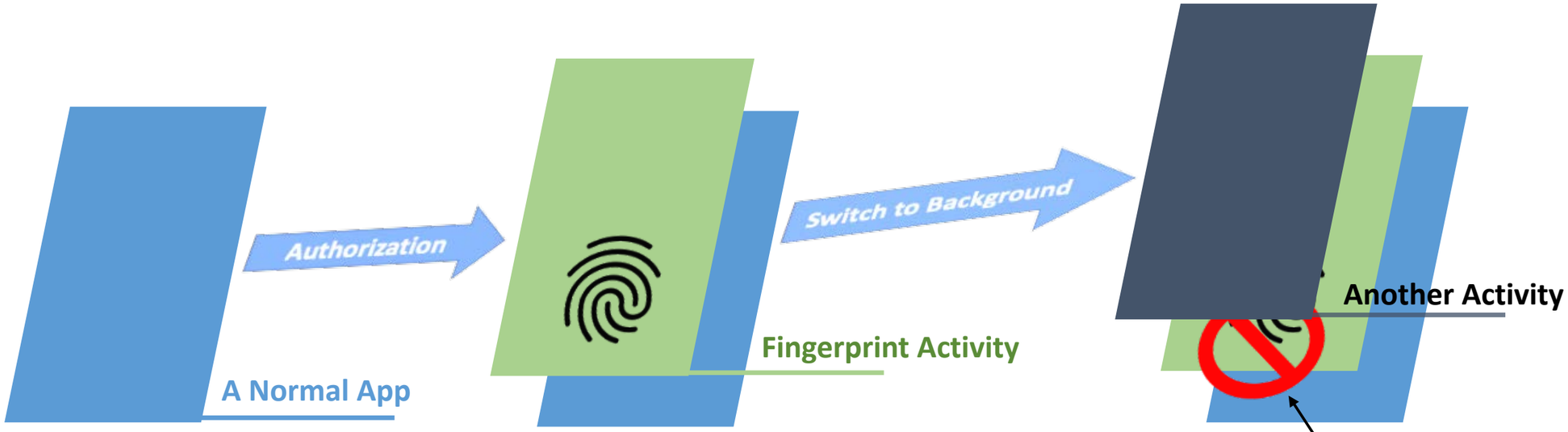
- Different techniques to construct **practical fingerprint-jacking** attack.
- Android is supposed to block this kind of attack, but we managed to **bypass the mitigation**.

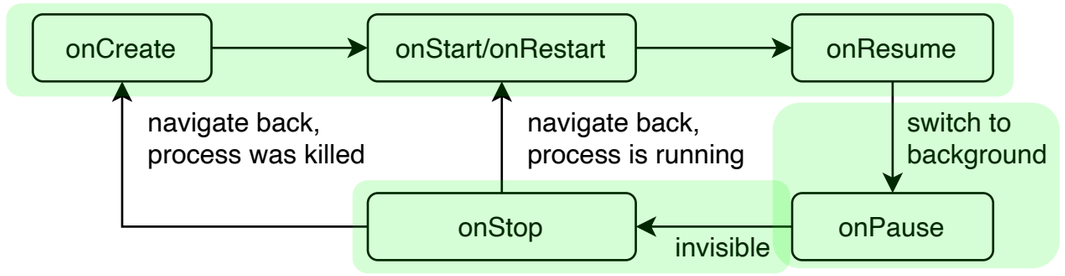
Background: Android Activity Life Cycle

- A state machine model for Android Activities (windows).
- Only one Activity can be in the **resumed** (running) state at a time.
- When the Activity is not in the foreground, it must have been **paused**, but may not be **stopped** (if still visible)

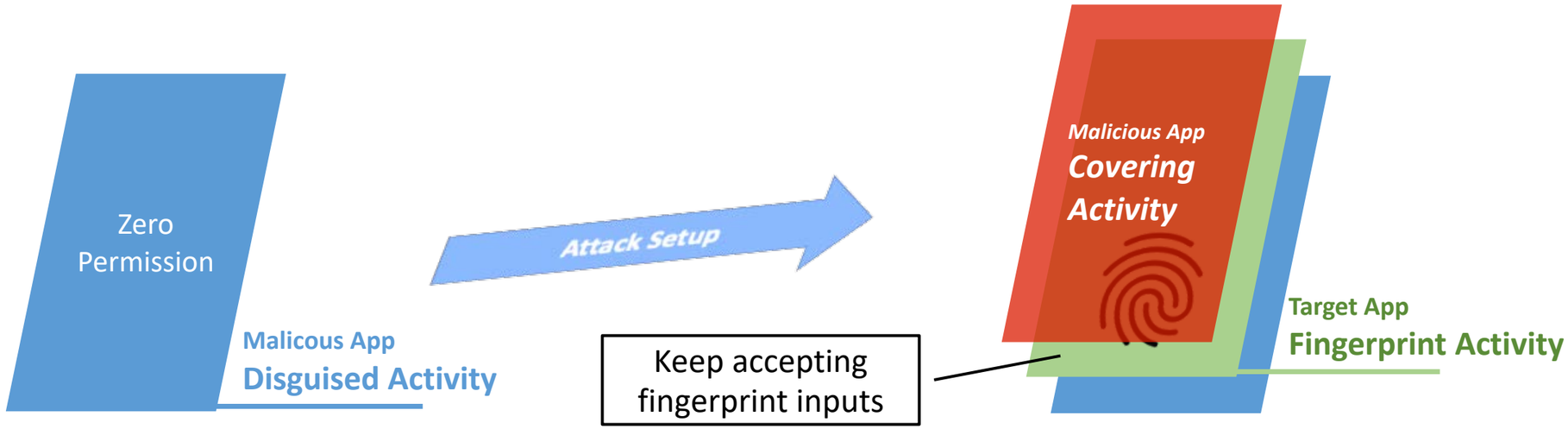


Typical Fingerprint Authorization Behaviour





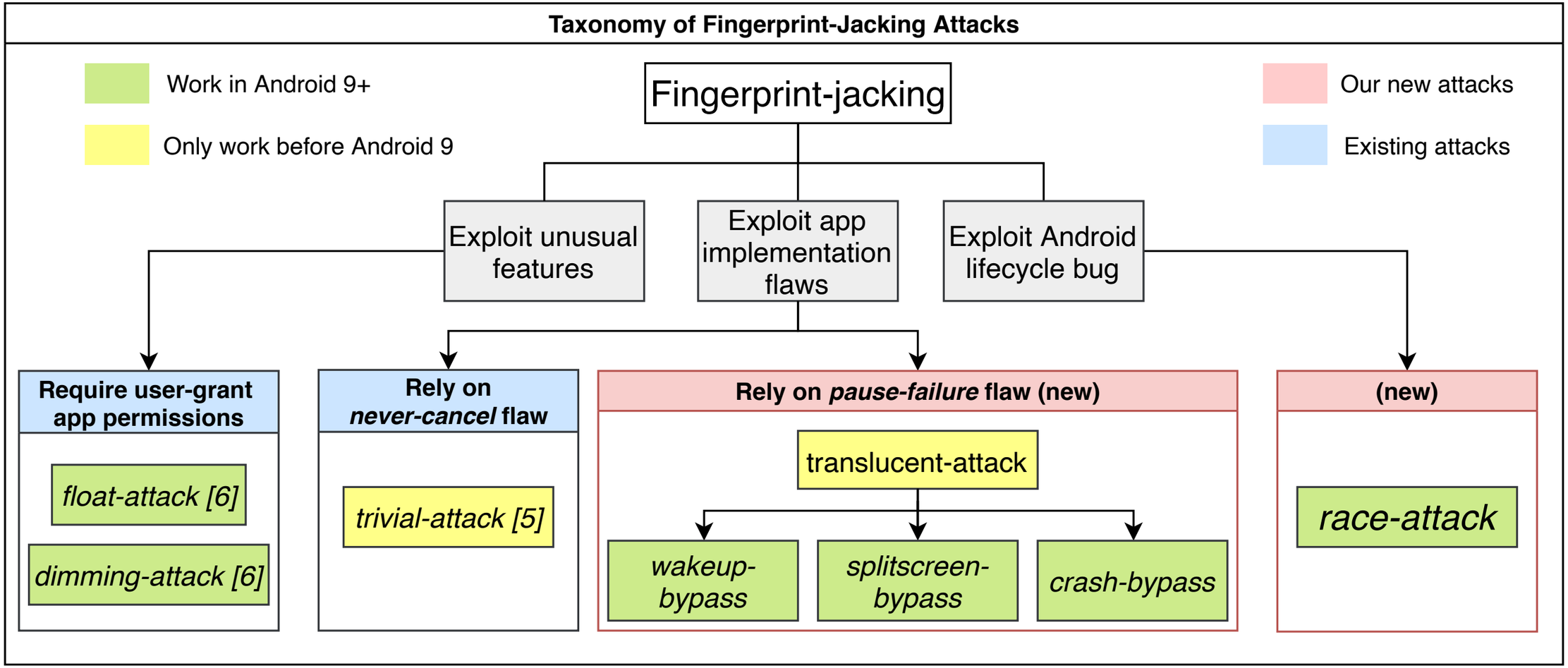
Blueprint of Fingerprint-Jacking Attack



What kind of "Attack Setup" makes the attack work? Considering different:

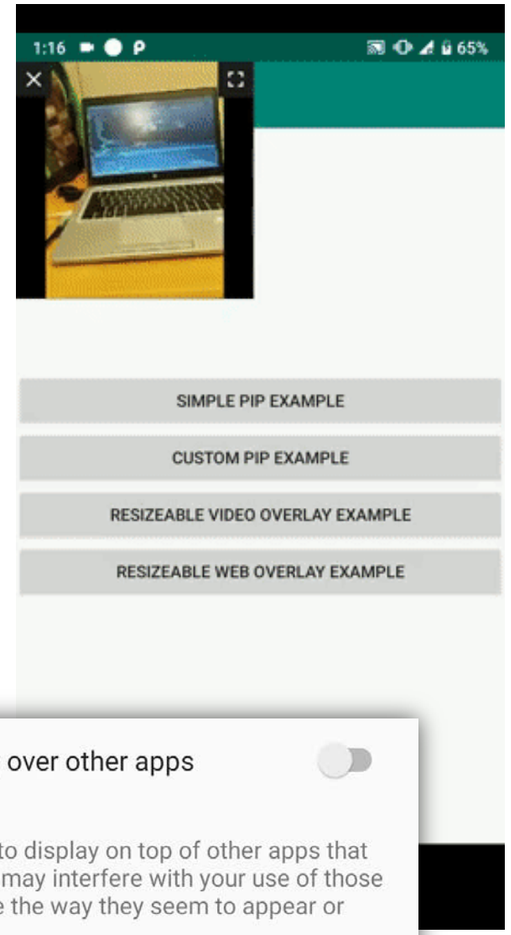
- Apps' implementation
- Android OS versions

Overview of Fingerprint-Jacking Techniques



Existing Techniques

- Simply put to the background ([5] BlackHat USA 2015)
  - On very old Samsung device
- Floating window [6]
  - Requires SYSTEM_ALERT_WINDOW **permission** (draw-over-other-apps)
- Dimming the screen by controlling brightness [6]
  - Requires WRITE_SETTINGS **permission**
  - Less practical, less effective



* Gif from https://github.com/KoderLabs/overlay-service/

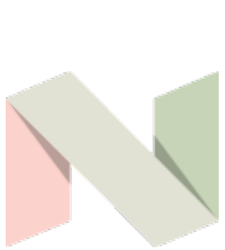
What we want for our fingerprint-jacking attack:

- Zero-permission malicious app
- Work on modern Android versions

Android Fingerprint API

Provide ***FingerprintManager*** API since Android 6.0 (API 23): developers need to build their own UI.
New ***BiometricPrompt*** API introduced in Android 9.0 (API 28): unified UI, more secure, but **backward incompatible**

Can apps listen to fingerprint input in the background?



Android 7 (Nougat)

Android 8 (Oreo)

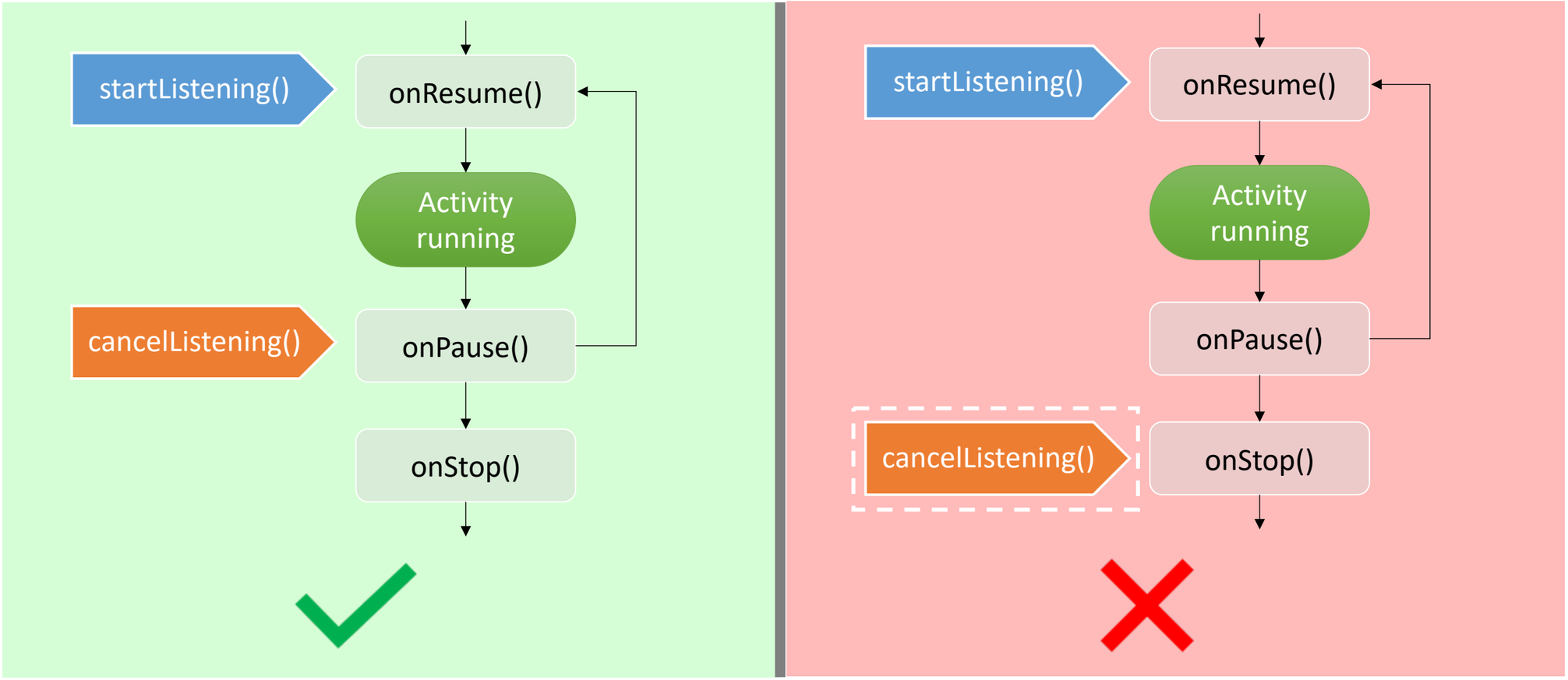
Android 9 (Pie)

? Android 10

Add mitigation in FingerprintManger API

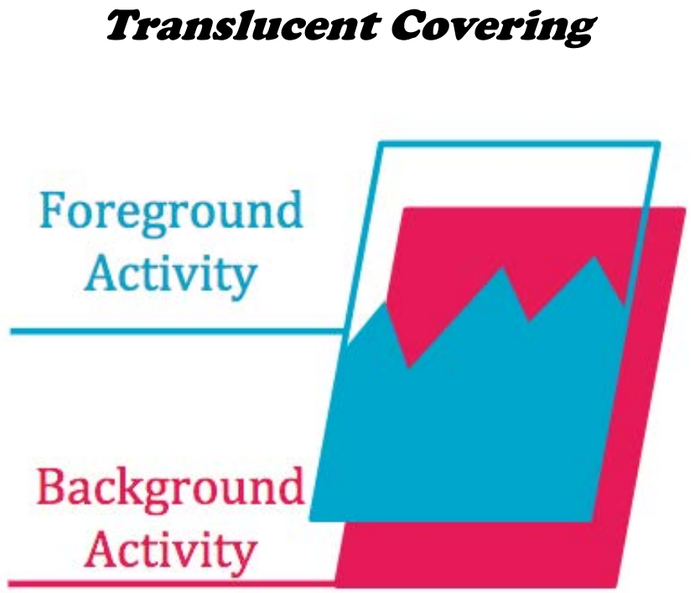
Can apps do it correctly?

Before Android 9, apps need to block background fingerprint inputs by themselves

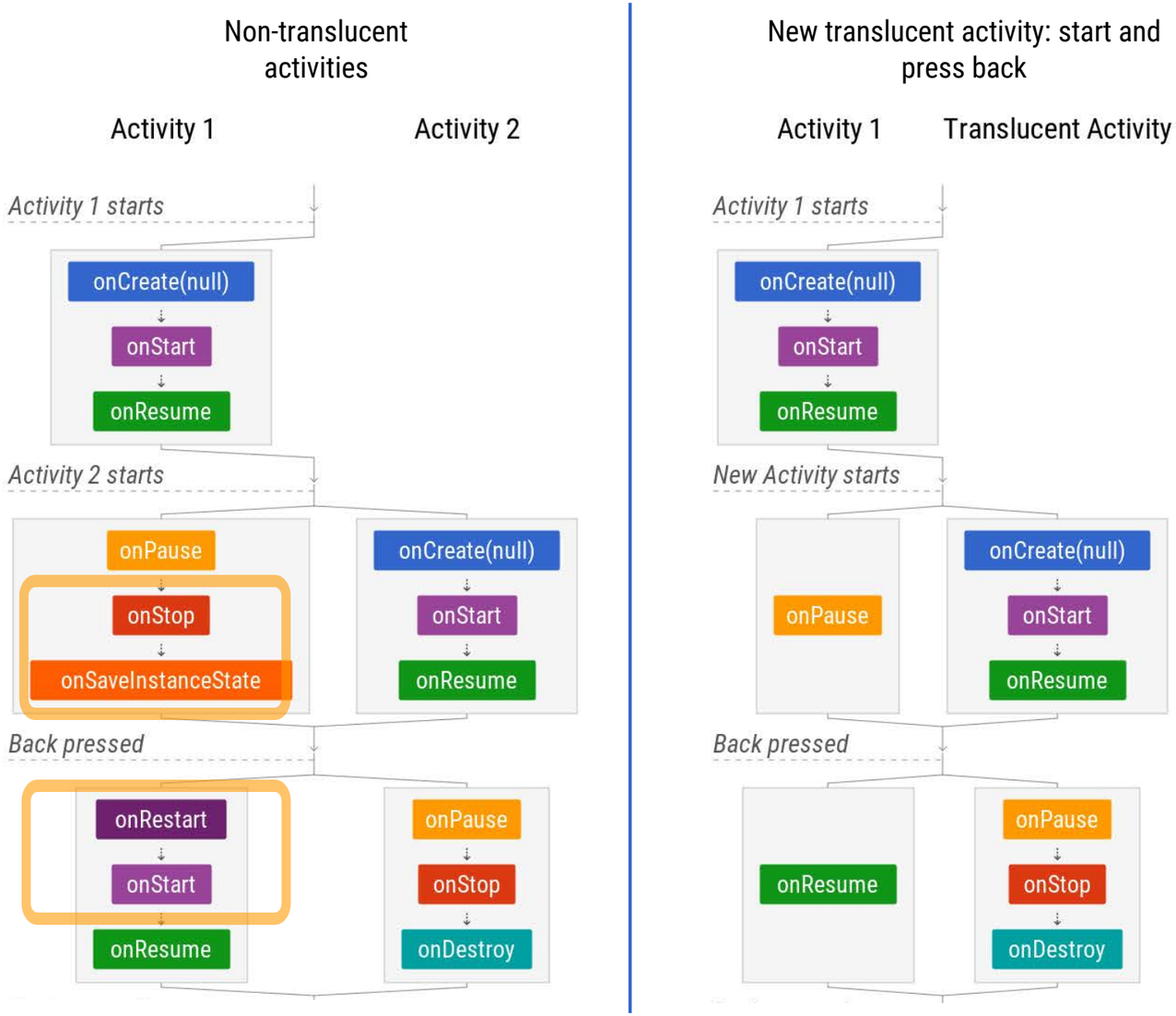


What's wrong with cancelling fingerprint in onStop?

- Background Activity covered by translucent Foreground Activity is considered **visible**.
- Background Activity goes into **paused** state, but not **stopped** state.



Translucent Activity has slightly different lifecycle model



* Based on the diagram from https://github.com/JoseAlcerreca/android-lifecycles

Exploit implementation flaw: *translucent-attack*

Targeting apps that don't cancel fingerprint in onPause

How many apps in the market do it incorrectly?

# of collected apps	# of analyzed apps (API call found)	No API-Activity links	No implementation flaw	Found implementation flaw = *never-cancel* + *pause-failure*
2024	1630	920	363	347 = 68 (19.6%) + 279 (80.4%)
Average analysis time	Max analysis time	Average memory consumption	Max memory consumption	
76.8 seconds	11.1 hours	1.2 GB	7.7 GB	

Large Scale Evaluation with Static Analysis

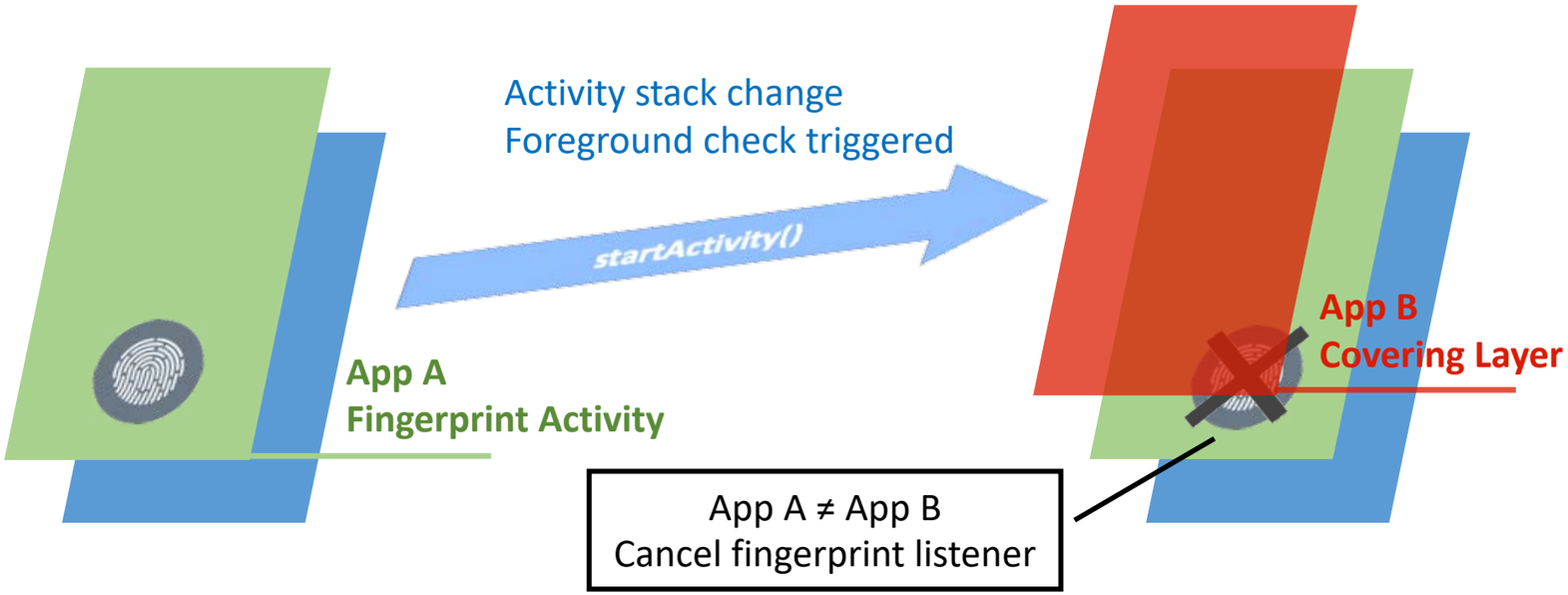
(Based on FlowDroid)

- Evaluation set: 2024 sample apps that declared the USE_FINGERPRINT permission
- Average time per app: 76.8 seconds
- Among analyzable apps, **48.9% contain implement flaws**
- Conservative analysis, no false positives

What happened in Android 9+?

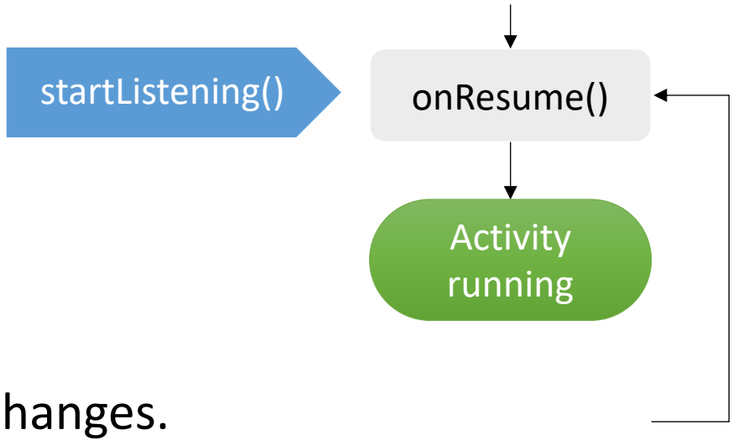
New mitigation code in FingerprintManager API to block background fingerprint inputs

- Mitigation logic:
  - Check current foreground Activity whenever the **Activity stack changes**.
  - Interrupt fingerprint listener if foreground package is different from fingerprint listening package.



Can we bypass it?

Want bugs? Look in corners.



Think about corner cases:

- The mitigating check is only triggered when Activity stack changes.
- What about fingerprint **starts after the Activity is already in the background**?
  → No stack change!
- Remember when do many apps start fingerprint listening? In **onResume** event!

Put the fingerprint Activity in the background and let it resume

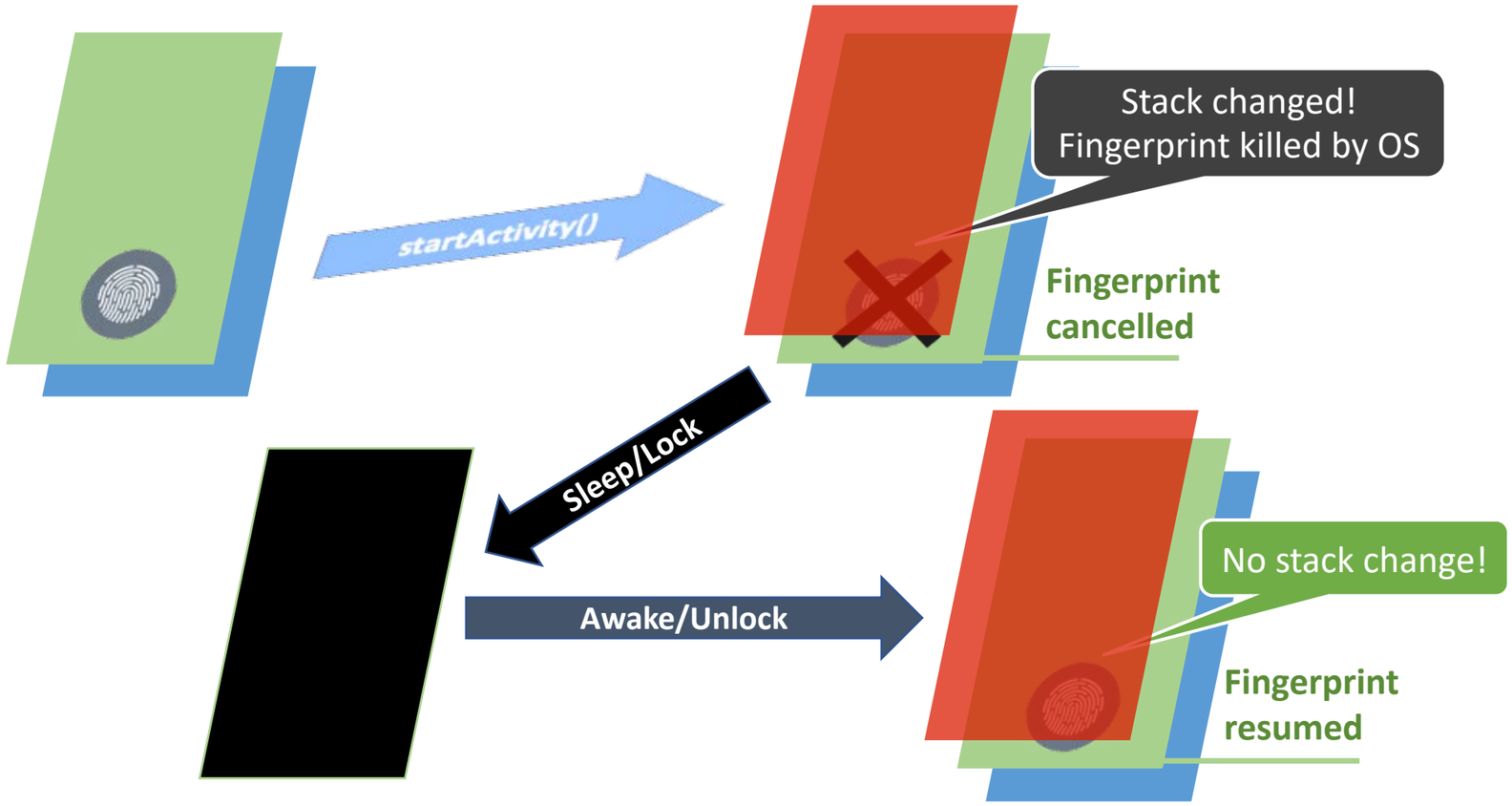
Resume Activities when Wakeup: *wakeup-bypass*

When device wakes up:

- All visible Activities' onResume are triggered

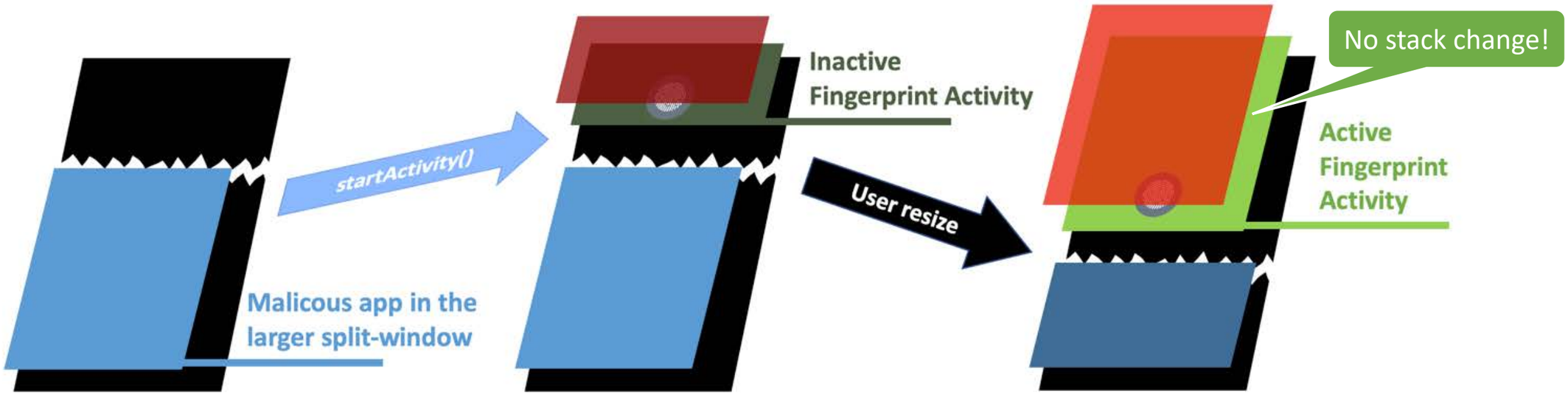
More practical way:

- Listen to the `ACTION_SCREEN_OFF` broadcast. Set up attack right before screen lock.



Use new features to bypass: *splitscreen-bypass*

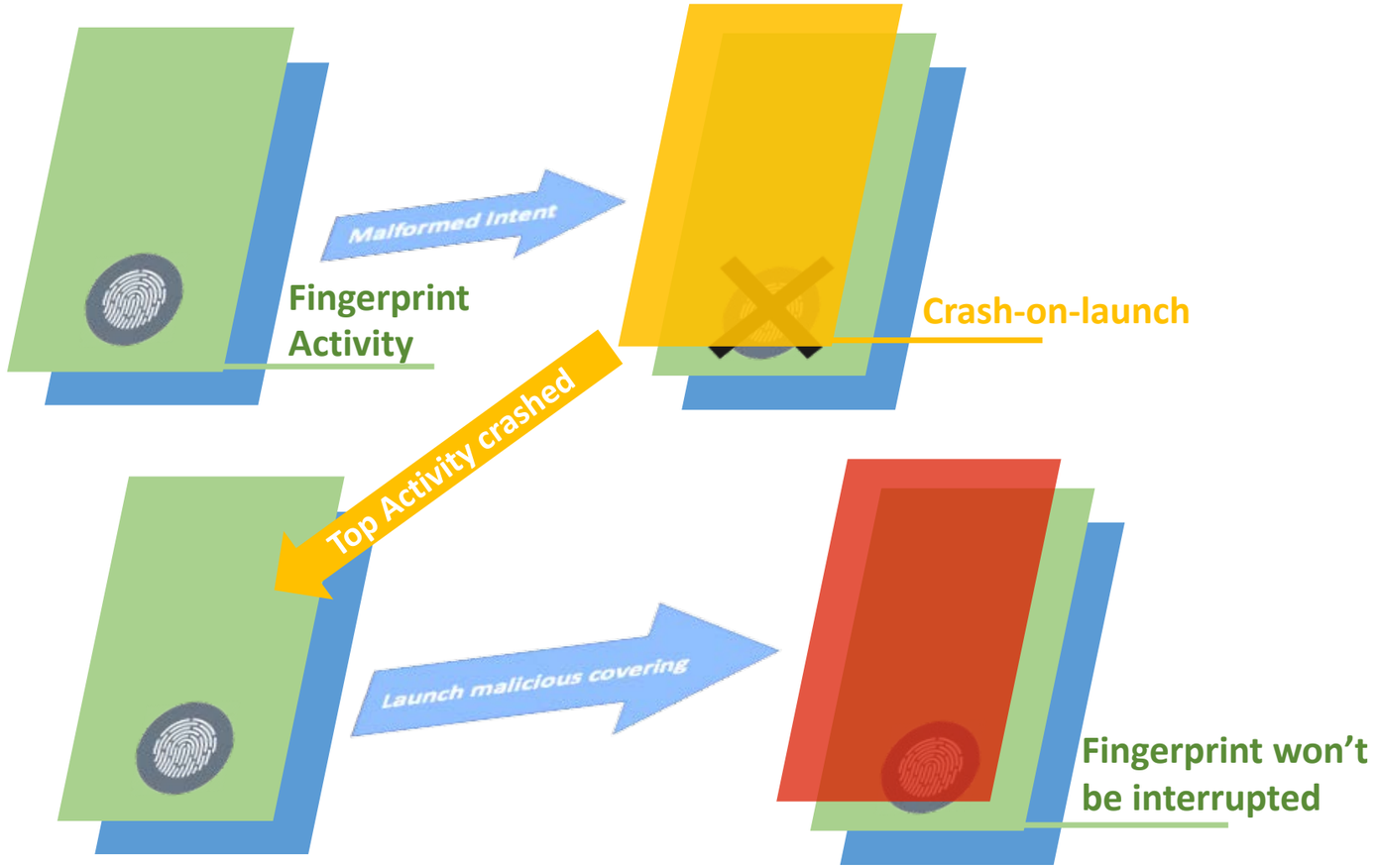
- Android 7+ supports split-screen. Multiple windows, but still only one resumed Activity.
- Resizing split-screen results in Activity lifecycle updates.
- Apps can put Activities into another screen with `Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT`.



Non-Standard Behaviour on OEM ROM: *crash-bypass*

Discovered when I tested on MIUI 11 (an OEM based on Android 9)

- Crash the covering Activity once will invalidate Android 9's mitigation.
- Only work on this OEM.
- Reason unclear.



Exploit Race-Condition: *race-attack*

- Bizarre lifecycle behavior when start two Activities **within very short period of time.**
  - `startActivities({victimIntent, maliciousIntent})`
- Background Activity never being paused
  - →Cancellation in onPause() becomes useless.
  - →Even correct app implementation cannot rescue.
- Invalidate Android 9+ mitigation.

```
1  # Failed to trigger race condition:
2
3  20:38:55.065 fjLog: Activity.onCreate: <VictimActivity>
4  20:38:55.085 fjLog: Activity.onStart: <VictimActivity>
5  20:38:55.085 fjLog: Activity.onResume: <VictimActivity>
6  20:38:55.155 fjLog: Activity.onPause: <VictimActivity>
7  20:38:55.189 fjLog: Activity.onStart: <MaliciousActivity>
8  20:38:55.190 fjLog: Activity.onResume: <MaliciousActivity>
9
10 # Race condition triggered:
11
12 20:39:25.314 fjLog: Activity.onCreate: <VictimActivity>
13 20:39:25.378 fjLog: Activity.onStart: <VictimActivity>
14 20:39:25.379 fjLog: Activity.onResume: <VictimActivity>
15 20:39:25.454 fjLog: Activity.onCreate: <MaliciousActivity>
16 20:39:25.477 fjLog: Activity.onStart: <MaliciousActivity>
17 20:39:25.480 fjLog: Activity.onResume: <MaliciousActivity>
```

Patch will be available in Jan 2021, currently 0day!

- We reported this issue to Google in June 2020
  - We provided PoC video and source code
  - They requested us to test on the latest Android, so we tested with Android 11 and it worked.
- Recently confirmed:
  - Will be assigned CVE-2020-27059
  - Will release a patch in the January 2021 Android Security Bulletin
- We may release more technical details of this bug then:
  - Different ways to trigger the race-condition bug
  - It breaks Activity lifecycle, it may break other things apart from fingerprint

What? You dispise attacks with malicious apps?

It is possible to launch fingerprint-jacking attack from **web browsers**, with a bunch of conditions:

1. You need to find some ***covering-gadget.***

2. The target app's fingerprint authorization can be invoked from the browser.
   - A common case: mobile payment app that supports payment from webpages in mobile browsers.

What is *covering-gadget*?

We define *covering-gadgets* to be Activities in benign installed apps that

- Allows invocation from browser
- With attacker-controllable visual content
- Having the `translucent` property

They are usually in the form of deep-links (URLs that link to mobile apps/Activities)

```
benignapp://webview/?url=http://malicious.com
```

They are not easy to find ☹. Currently, I only found one with partial content control.

Attacking HTML source

```
<html>
<!-- Invoke victim Activity -->
<a href="targetapp://fingerprint" onclick=intent()>CLICK ME</a>
<!-- Try invoking different covering gadgets -->
<a href="app1://webview/?url=http://evil.com" id="a1"></a>
<a href="app2://fullscreen/?color=black" id="a2"></a>
<a href="app3://evil.com" id="a3"></a>
<script>
function intent(e){setTimeout(
    function(){
        document.getElementById("a1").click();
        document.getElementById("a2").click();
        document.getElementById("a3").click();
    }, 0);
}
</script>
</html>
```

Summary of Attack Techniques and Conditions

Attacks		Implementation flaw dependency		System requirement	Implementation pattern dependency		Attacker capability requirement	
		Rely on *never-cancel*	Rely on *pause-failure*	Require Android<9	Require *auto-resume*	Require *no-button*[1]	Require a malicious app	Malicious app's permission
Known attacks	*trivial-attack*[4][5]	✓	✓	✓	✗	✗	✗	None
	float-attack [6]	✗	✗	✓[3]	✗	✗	✓	SYSTEM_ALERT_WINDOW[3]
	dimming-attack [6]	✗	✗	✗	✗	✓	✓	WRITE_SETTINGS[3]
New attacks	*translucent-attack*	✗	✓	✓	✗	✗	✗	None
	wakeup-bypass	✗	✓	✗	✓	✓	✓	None
	splitscreen-bypass	✗	✓	✗	✓	✓	✓	None
	crash-bypass	✗	✓	✗[2]	✓	✓	✗	None
	race-attack	✗	✗	✗	✗	✗	✓	None

Suggestions to Developers

- Use AndroidX's `androidx.biometric` API. It's a wrapper for `FingerprintManager` and `BiometriPrompt` API with secure implementation.

- Use third-party fingerprint libraries carefully. We tested some unofficial fingerprint libraries and found them vulnerable to the fingerprint-jacking attack.

- Check your existing implementations, if you use `FingerprintManager` API, make sure your app explicitly cancel the fingerprint authentication process in the onPause event.

Demo with Covering Revealed

FINGERPRINT-JACKING

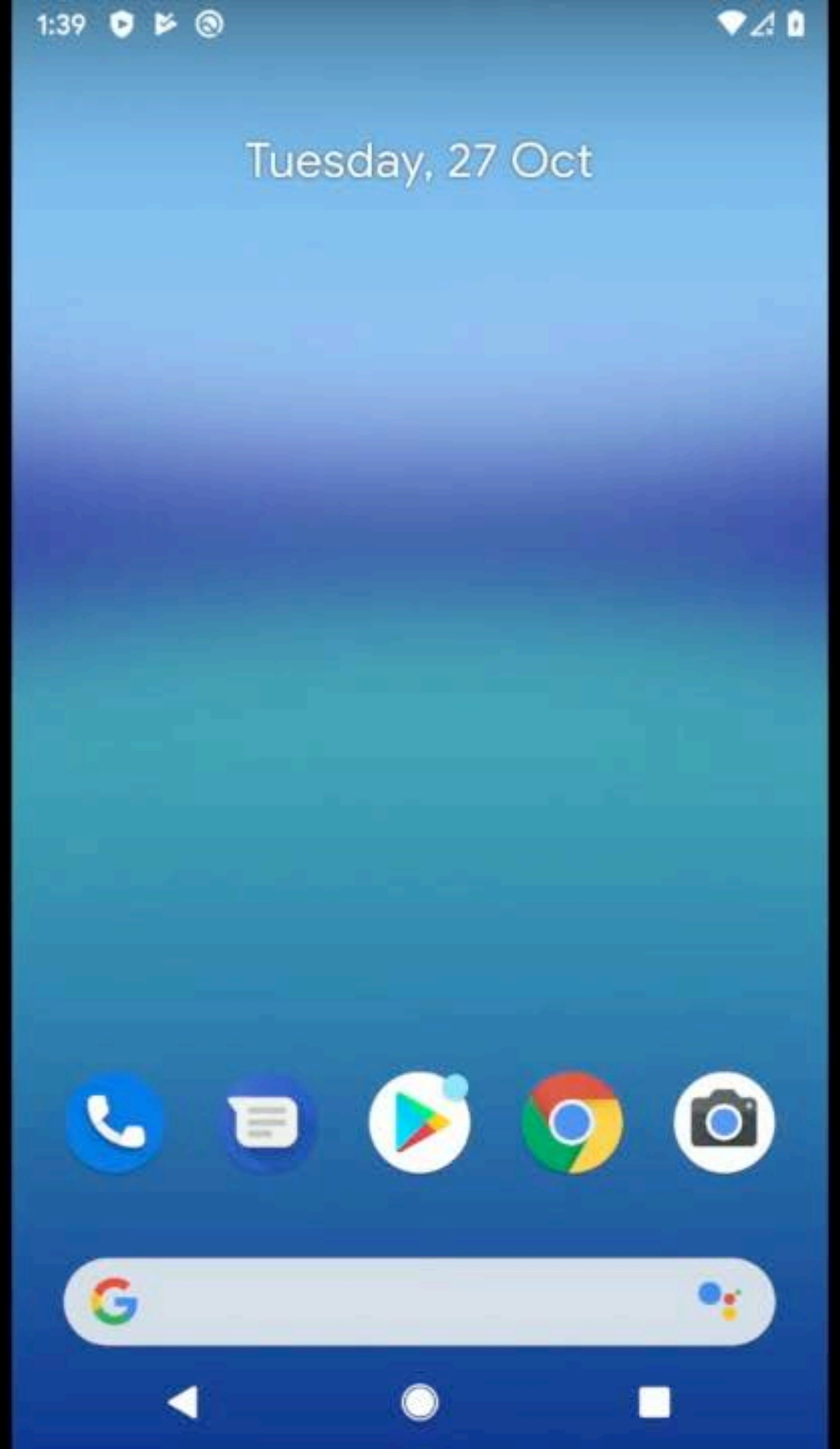
Gain root permission in Magisk

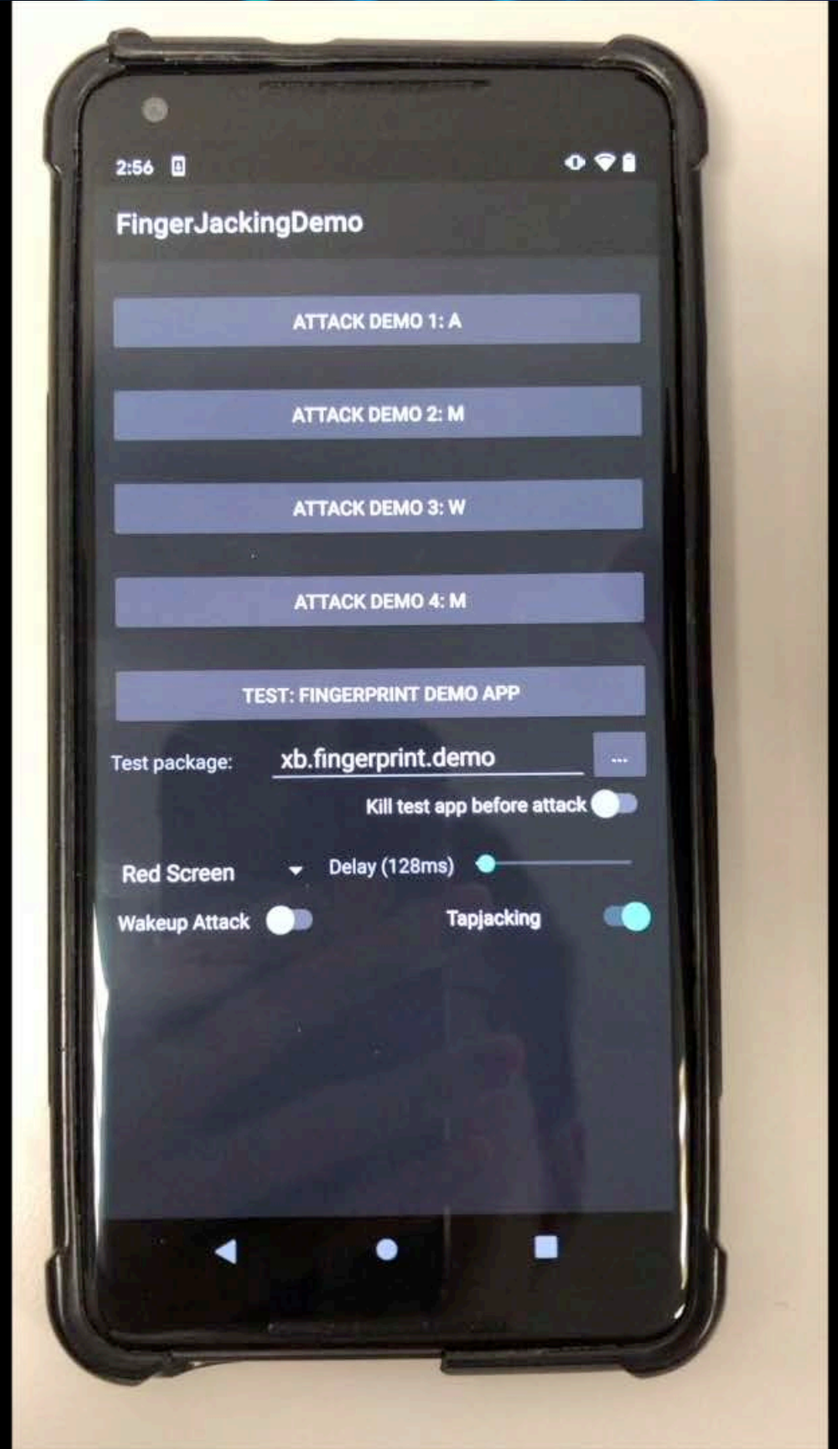
* Latest Magisk Manager is not vulnerable

FINGERPRINT-JACKING & TOUCHJACKING

Money stealing in a Payment App

* An open-source 3rd party fingerprint payment plugin


1:39
Tuesday, 27 Oct


2:56
FingerJackingDemo
ATTACK DEMO 1: A
ATTACK DEMO 2: M
ATTACK DEMO 3: W
ATTACK DEMO 4: M
TEST: FINGERPRINT DEMO APP
Test package:
xb.fingerprint.demo
...
Kill test app before attack
Red Screen
Delay (128ms)
Wakeup Attack
Tapjacking

Thank you!
Q&A Time

* We have more details in our whitepaper

References

- [1] Niemietz, Marcus, and Jörg Schwenk. "Ui redressing attacks on android devices." *Black Hat Abu Dhabi* (2012).
- [2] Fratantonio, Yanick, et al. "Cloak and dagger: from two permissions to complete control of the UI feedback loop." *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017.
- [3] Yan, Yuxuan, et al. "Understanding and Detecting Overlay-based Android Malware at Market Scales." *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. 2019.
- [4] Zheng, Cong, et al. “Android Toast Overlay Attack: “Cloak and Dagger” with No Permissions.” *Paloalto Networks.* 2017.
- [5] Zhang, Chen, et al. “Fingerprints On Mobile Devices: Abusing and Leaking.” *BlackHat USA*. 2015.
- [6] Bianchi, Antonio, et al. "Broken Fingers: On the Usage of the Fingerprint API in Android." *NDSS*. 2018.