Protecting Against Common Android Threats

- Accessbility Service
- Overlay Attacks

Accessibility Service

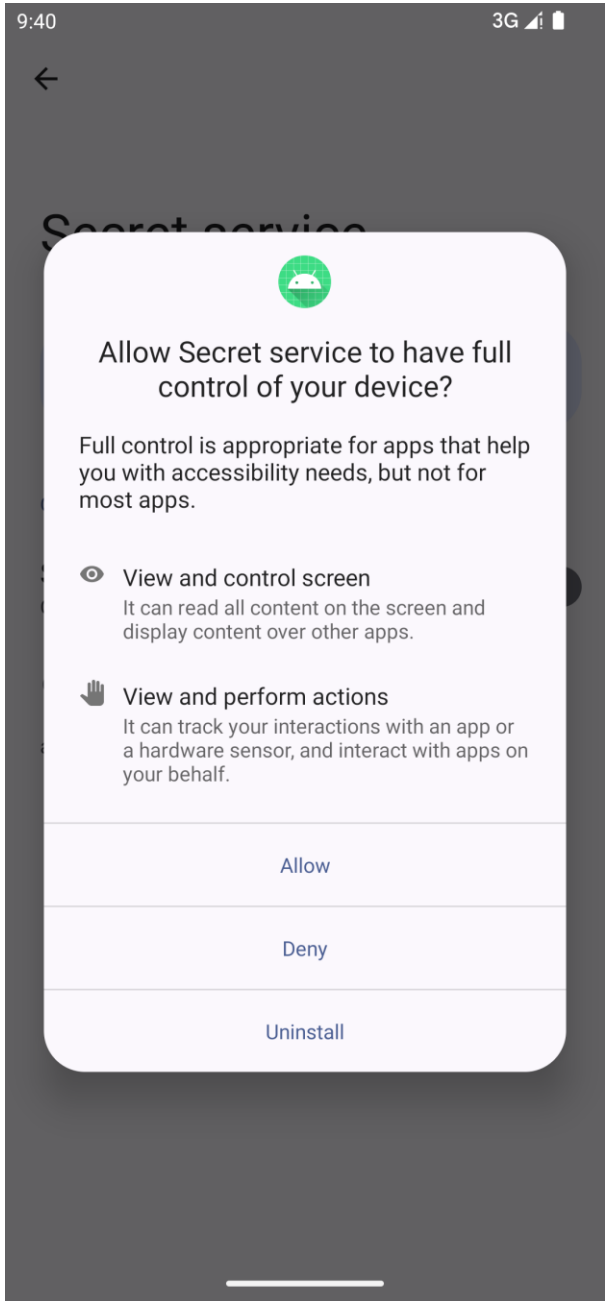
A powerful feature that allows users to interact with their devices in new ways.

Custom Accessbility Service

On Android, Developers can create custom background accessibility services to help users to use the device by:

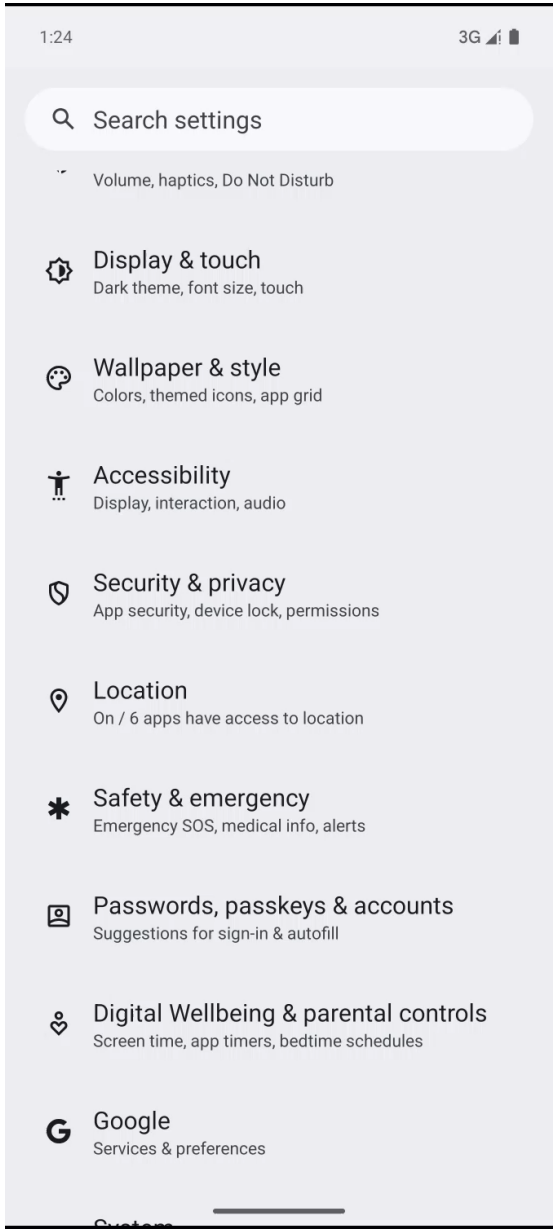
- Reading text aloud.
- Filling in forms.
- Clicking buttons for the users.

Risks



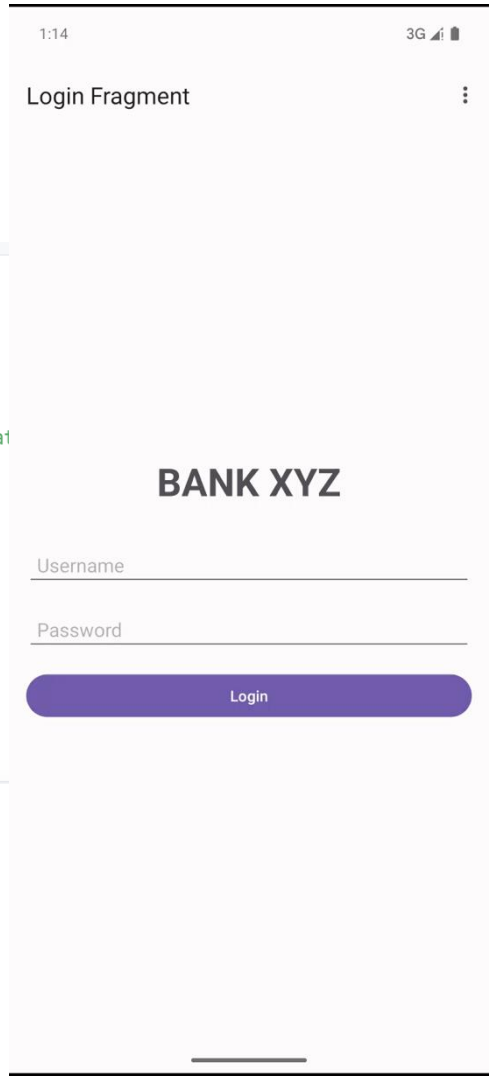
- Stealing senstive data.
- Controlling the device.

Controling Device

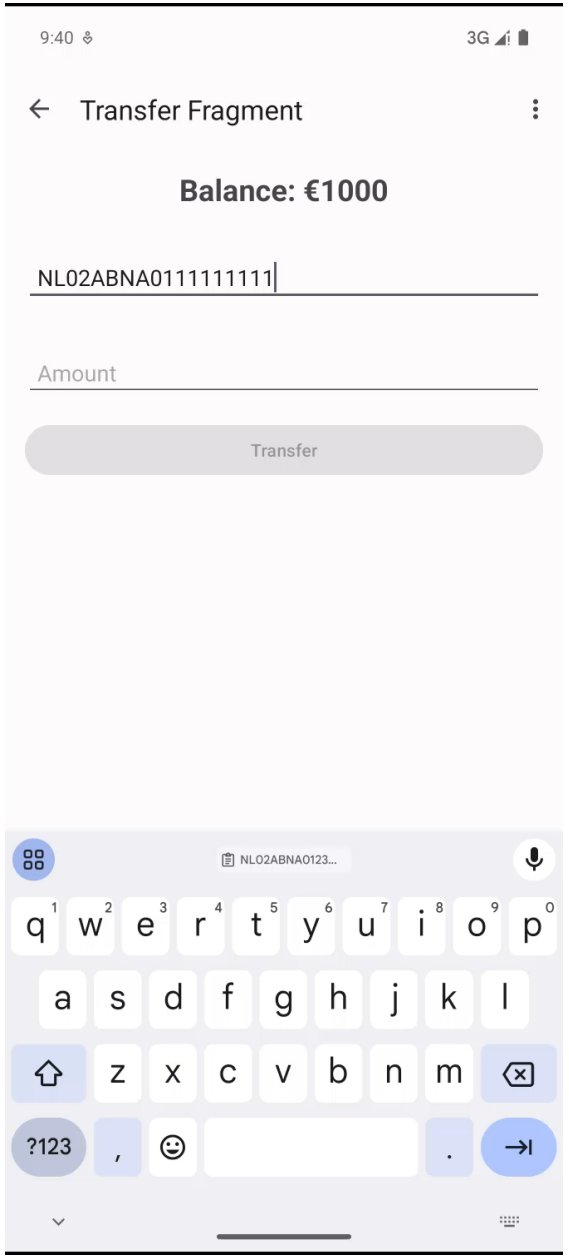


Reading Screen Content

```
AccessibilityService  I  Text: [Tes] Class Name: android.widget.EditText
AccessibilityService  I  Text: [Test] Class Name: android.widget.EditText
AccessibilityService  I  Text: [Test] Class Name: android.widget.EditText
AccessibilityService  I  Text: [] Class Name: android.widget.EditText
AccessibilityService  I  Text: [Password] Class Name: android.widget.EditText
AccessibilityService  I  Text: [Showing recently copied or captured content as candidat
AccessibilityService  I  Text: [•] Class Name: android.widget.EditText
AccessibilityService  I  Text: [p] Class Name: android.widget.EditText
AccessibilityService  I  Text: [p•] Class Name: android.widget.EditText
AccessibilityService  I  Text: [•a] Class Name: android.widget.EditText
AccessibilityService  I  Text: [••s•] Class Name: android.widget.EditText
AccessibilityService  I  Text: [•••s] Class Name: android.widget.EditText
AccessibilityService  I  Text: [•••s•] Class Name: android.widget.EditText
AccessibilityService  I  Text: [••••s] Class Name: android.widget.EditText
AccessibilityService  I  Text: [••••s•] Class Name: android.widget.EditText
AccessibilityService  I  Text: [•••••w] Class Name: android.widget.EditText
```



Modifying Content



Should I be concerned?

RESEARCH

Xenomorph Malware Strikes Again: Over 30+ US Banks

25 September 2023

Singapore

Nearly 2,000 victims fell for Android malware scams, at least S$34.1 million lost in 2023

The majority of victims were aged 30 to 49, and were most frequently targeted on Facebook [illegible] Instagram.

The malicious apps that become spies inside your phone: We download them for everything from games to fitness... experts warn some have spyware t[illegible] could cost you THOUSANDS

- Virus infected QR reader app reportedly made available in the Goog[illegible]
- The malware was designed to infiltrate mobile banking apps and ste[illegible] passwords
- It had already been installed by more than 10,000 users before it wa[illegible]
- Three other sinister phone apps removed from Google Play Store th[illegible]
- Fraudsters build apps for everyday uses, like games, battery saving[illegible]
- Then they infect them with a virus that invades every part of your ha[illegible]

Source: Daily Mail, 2022

BLOG

BrasDex: A new Brazilian ATS Android Banker with ties to Desktop malware

15 December 2022

Source: ThreatFabric

NEWS 4 JUL 2023

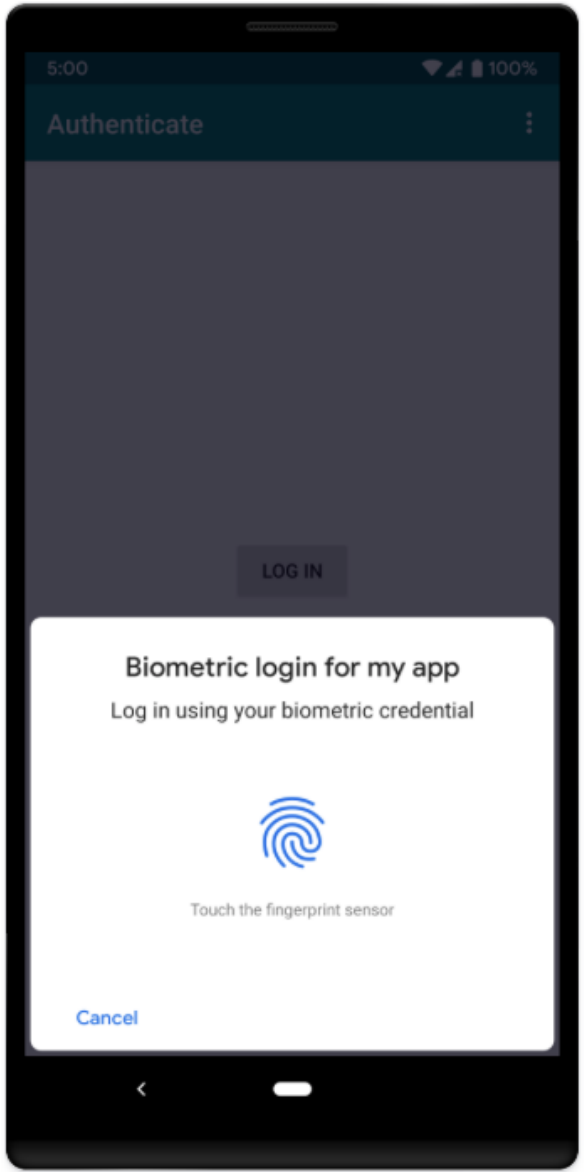
Mexican Hacker Unleashes Android Malware on Global Banks

Despite employing relatively unsophisticated tools, Neo_Net achieved remarkable success, stealing over €350,000 ($382,153) from victims' bank accounts and compromising the personal information of thousands of individuals.

Source: Infosecurity Magazine, 2023

Counter-measures

Use Biometrics



Countermeasures

- Detect

Simple Allow List

```
private fun hasDisallowedA11yServices(context: Context): Boolean {
    val allowedServices = setOf(
        "com.sample.assistant", "com.trusted.package"
    )
    val am = context.getSystemService(Context.ACCESSIBILITY_SERVICE)
            as AccessibilityManager
    val services = am.getEnabledAccessibilityServiceList(
        FEEDBACK_ALL_MASK
    )
    for (enabledService in services) {
        if (!allowedServices.contains(enabledService.id)) return true
    }
    return false
}
```

Countermeasures

- Detect

Allow List – Check Non System Accessibility Service

```
private fun getListOfNonSystemEnabledAccessibilityServices(context: Context): List<String> {

    val a11yServiceList = getListOfEnabledA11yServices(context)
    val nonSystemA11yAppList: MutableList<String> = ArrayList()
    var packageName: String
    val packageManager = context.packageManager
    var packageInfo: PackageInfo
    for (asi in a11yServiceList) {
        packageName = asi.id.split( ...delimiters: "/").first()
        try {
            packageInfo = packageManager.getPackageInfo(
                packageName,
                PackageManager.GET_META_DATA
            )
            if (packageInfo.applicationInfo.flags and ApplicationInfo.FLAG_SYSTEM == 0) {
                Log.d( tag: "APP_INSPECTOR",  msg: "[!] app '$packageName' has a11y and is not a System Service")
                nonSystemA11yAppList.add(packageName)
            }
        } catch (e: PackageManager.NameNotFoundException) {
            e.printStackTrace()
        }
    }
    return nonSystemA11yAppList
}
```

Countermeasures

- Detect

Allow List – Check Side Loaded services

```
val installerAllowList = setOf(
    "com.android.vending",
    "com.sec.android.app.samsungapps",
    // ...
    "com.huawei.appmarket"
)
for (asi in a11yServiceList) {
    packageName = asi.id.split( ...delimiters: "/").first()
    try {
        installer = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
            packageManager.getInstallSourceInfo(packageName)
                .installingPackageName
        } else {
            packageManager.getInstallerPackageName(packageName)
        }
        if (installer == null || !installerAllowList.contains(installer)) {
            sideLoadedA11yAppList.add(packageName)
        }
    } catch (e: PackageManager.NameNotFoundException) {
        // (...)
    }
```

Countermeasures

- Detect

Allow List – Check Device Admins

```
private fun getListOfDeviceAdminApps(context: Context): List<String> {
    val deviceAdminAppList: MutableList<String> = ArrayList()
    val devicePolicyManager = context.getSystemService(Context.DEVICE_POLICY_SERVICE)
                as DevicePolicyManager
    val activeDeviceAdminComp = devicePolicyManager.activeAdmins
    if (activeDeviceAdminComp != null) {
        for (cn in activeDeviceAdminComp) {
            deviceAdminAppList.add(cn.packageName)
        }
    }
    return deviceAdminAppList
}
```

Compare it with enabled Accessbility services package name.

Countermeasures

- Block

Custom Accessbility Delegate

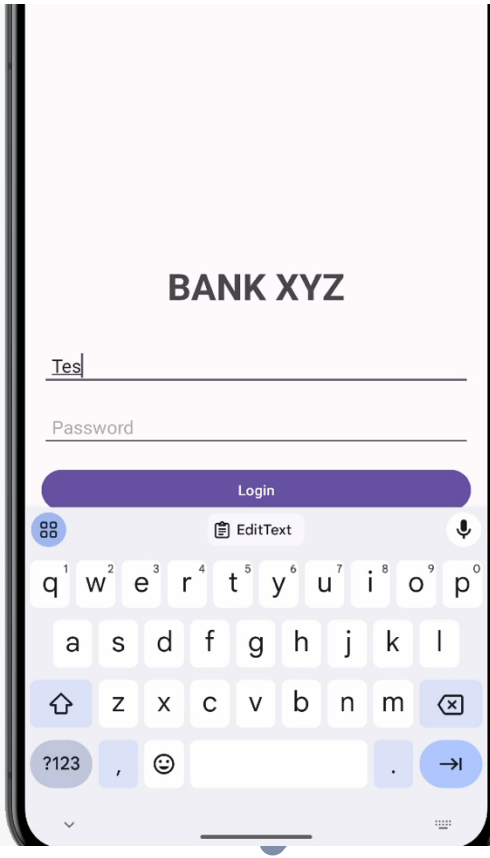
```
class CustomAccessibilityDelegate : View.AccessibilityDelegate() {

    override fun onPopulateAccessibilityEvent(host: View, event: AccessibilityEvent) {
        super.onPopulateAccessibilityEvent(host, event)
        if (event.text.isNotEmpty()) {
            event.text.clear()
            event.text.add("CENSORED")
        }
    }

    override fun onInitializeAccessibilityNodeInfo(host: View, info: AccessibilityNodeInfo) {
        super.onInitializeAccessibilityNodeInfo(host, info)
        if (info.text.isNotEmpty()) {
            info.text = "CENSORED"
        }
    }
}
```

Custom Accessbility Delegate

Countermeasures

- Block

```
ice    I  Text: [Showing recently copied or captured content as candidate] Class Name:
ce     I  Text: [Showing recently copied or captured content as candidate] Class Name:
```



Countermeasures

- Block

- On Android 14 a new Android property (accessibilityDataSensitive) is added to limit visibility of a specified view.
- It prevents critical actions from being executed unintentionally.
- Accessibility service with declared IsAccessibilityTool must consent Google Play requirement.

Other Countermeasures

- Use accessible captcha to verify that your app user is a human.

- Analyze usage behaviour (motion sensors, touch events, device orientation ..etc).

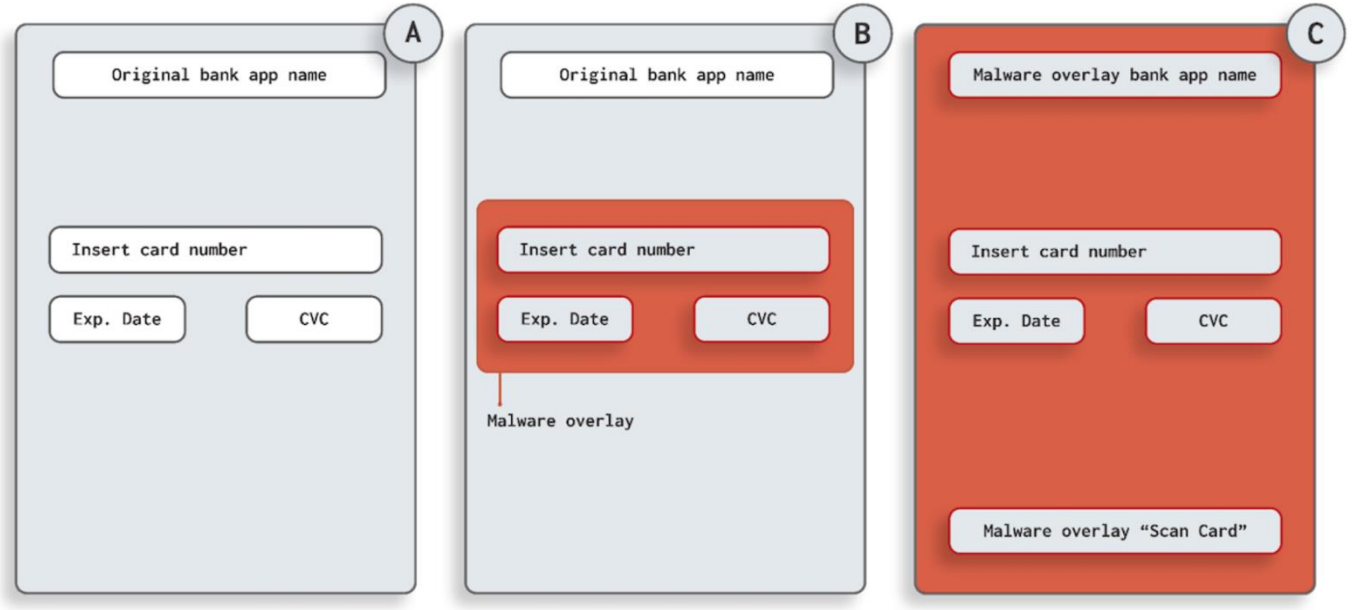
Overlay Attacks

- Feature used by an app to appear on top of another app.
- Commonly used for messaging Apps.

Risks

- Theft of sensitive data.

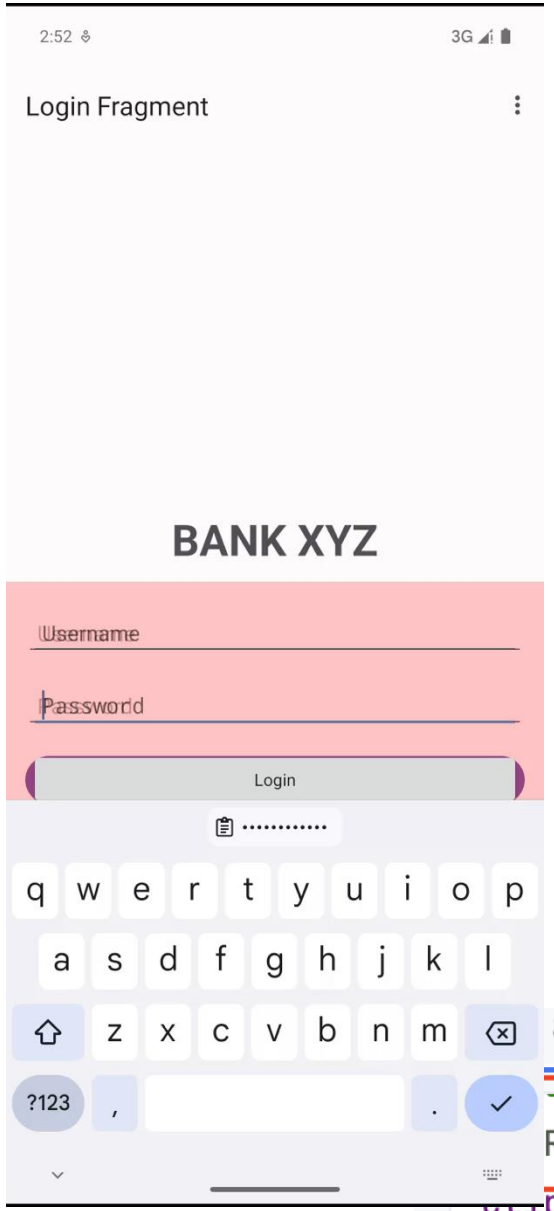
Overlay Attacks

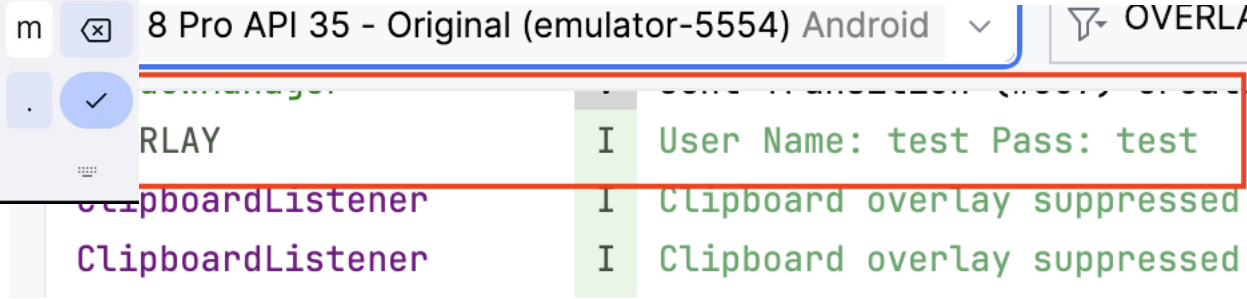


Source: guardsquare

Partial vs Full overlay attacks

Example


2:52
3G
Login Fragment
BANK XYZ
Username
Password
Login


8 Pro API 35 - Original (emulator-5554) Android
OVERLA
RLAY I User Name: test Pass: test
pboardListener I Clipboard overlay suppressed
ClipboardListener I Clipboard overlay suppressed

Countermeasures

API Level ≥ 31 :

```
activity.setHideOverlayWindows(true)
```

Countermeasures

Obscure touch detection:

- FLAG_WINDOW_IS_PARTIALLY_OBSCURED API Level ≥ **29**

- FLAG_WINDOW_IS_OBSCURED API Level ≥ **9**

```
binding.root.setOnTouchListener { _, event ->
    val flags = event.flags

    val badTouch = flags and FLAG_WINDOW_IS_PARTIALLY_OBSCURED != 0 ||
            flags and FLAG_WINDOW_IS_OBSCURED != 0

    return@setOnTouchListener badTouch
}
```

Countermeasures

Window Punching

- Works on API level < 33

2016 IEEE 40th Annual Computer Software and Applications Conference

Maintaining User Interface Integrity on Android

Abeer AlJarrah
College of Computing & Informatics
University of North Carolina at Charlotte
Charlotte, North Carolina 28223
Email: aaljarra@uncc.edu

Mohamed Shehab
College of Computing & Informatics
University of North Carolina at Charlotte
Charlotte, North Carolina 28223
Email: mshehab@uncc.edu

***Abstract*—The demand of having a multi-window and multi-tasking option in Android devices has been an emerging topic among Android users, especially with the trends toward larger hand-held screen sizes. One option to meet this demand, is to use floating windows. This feature enables users to perform more than one task at the same time while sharing the same screen. Device screens can be divided into multiple windows that can have different visual features in terms of size, location and transparency. While this feature addresses user complaints about Android on large screen devices, attention must be given to the security implications of such an option.**

In this work, we demonstrate how the current implementation of floating windows on Android can be abused to compromise user interface integrity through several attacks such as tapjacking, event eavesdropping and eventhijacking.

Although previous versions of Android have evolved to handle

simply through libraries that neither require installing a custom ROM nor root access, such as StandOut library [3]. This is an attractive feature for users who have large screen smartphones and tablets. Fig. 1 shows a screenshot of one of the floating apps from Google Play.

Countermeasures

Window Punching

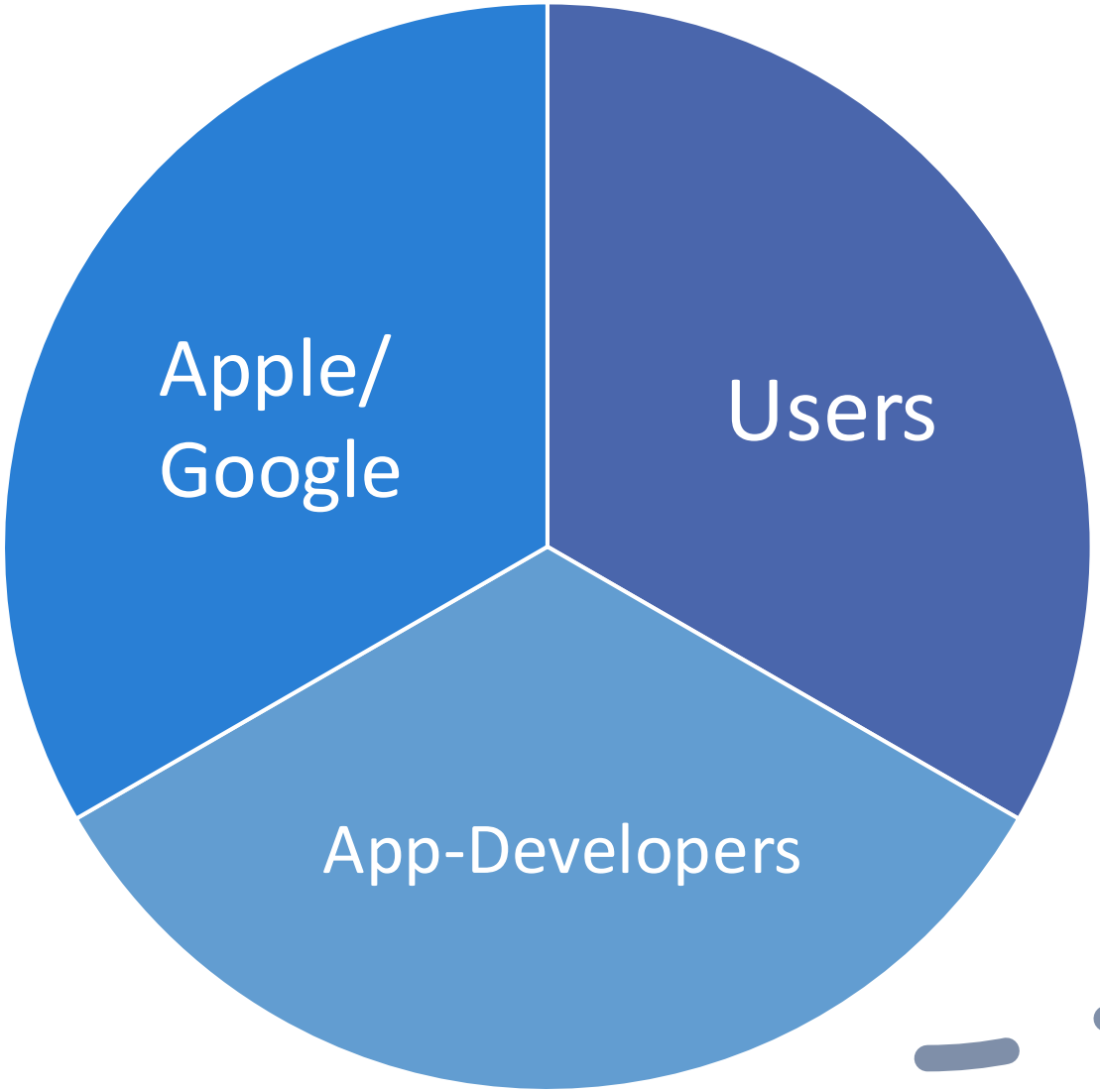
- An application can generate touches only on views that belong to itself.

- Using android.app.Instrumentation the app can simulate touches (“punches”).

- If the punch is on the overlay a SecurityException will be raised.

Summary

It's a joint responsibility.



Q&A

Thank you ☺

Refrences

- https://www.guardsquare.com/blog/protecting-against-android-overlay-attacks-guardsquare
- https://www.researchgate.net/publication/303328147_Maintaining_User_Interface_Integrity_on_Android
- https://faui1-files.informatik.uni-erlangen.de/public/publications/a11y_final_version.pdf
- https://www.threatfabric.com/blogs/double-trouble-in-latam
- https://www.threatfabric.com/blogs/double-trouble-in-latam
- https://android-developers.googleblog.com/2023/04/android-14-beta-1.html
- https://support.google.com/googleplay/android-developer/answer/10964491?hl=en
- https://www.guardsquare.com/blog/how-android-malware-works