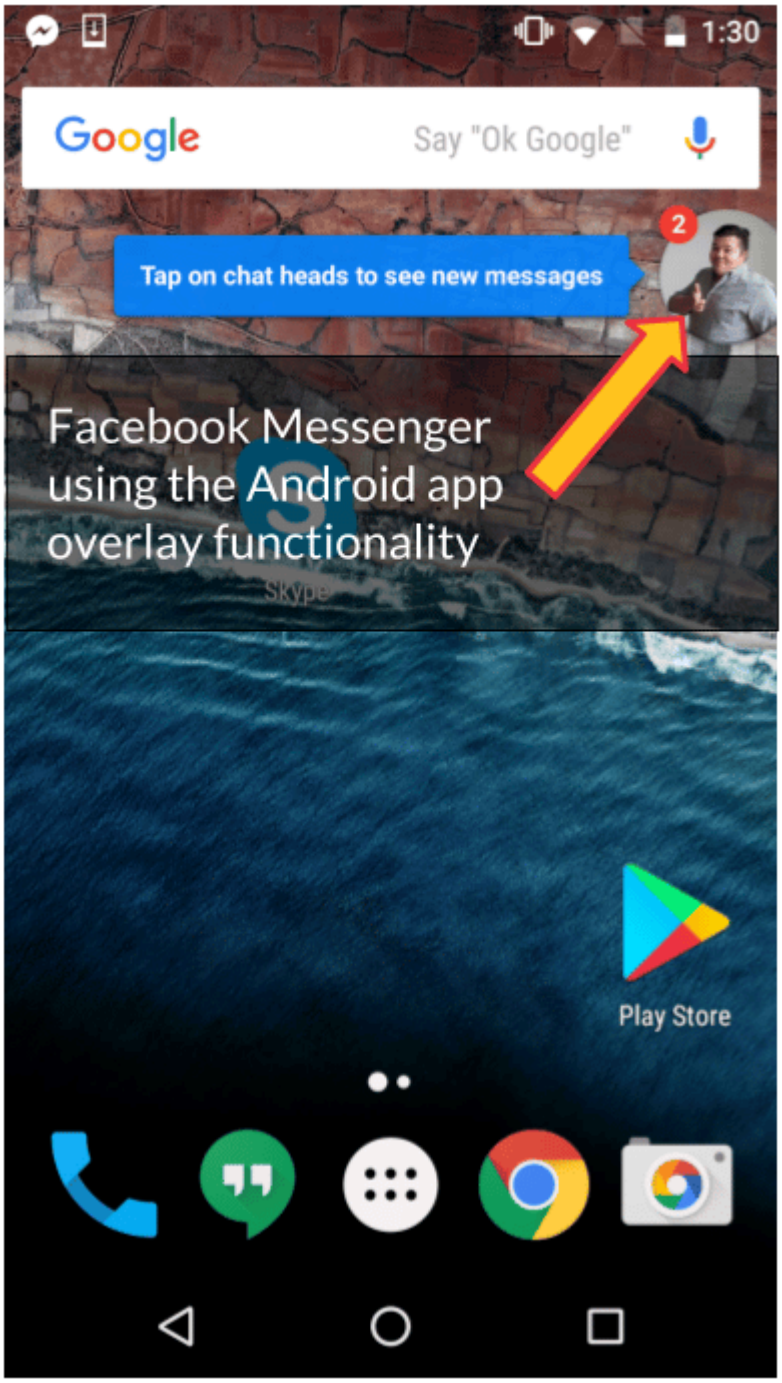
‘SAW’-ing through the UI: Android overlay malware and the System Alert Window permission explained

Published: 2017-05-25 · Archived: 2026-04-05 23:51:12 UTC

Over the past month, we’ve talked about a troublesome Android app permission called `SYSTEM_ALERT_WINDOW` a few times. This week, researchers from University of California Santa Barbara and Georgia Institute of Technology released details about “Cloak and Dagger” a collection of attacks that take advantage of this permission. Customers[1] have asked me about `SYSTEM_ALERT_WINDOW` (abbreviated SAW for the purposes of my oh-so-witty headline) and how malware might abuse the Android overlay view to steal log-in credentials and the like from Android users. I wanted to explain the Android app permission in plain language, how it might be abused, and finally what people can do to mitigate the risk.

What is Android screen overlay and what’s it for?

An **Android overlay**, screen overlay, or “Draw On Top”, allows an app to display content over another app. The Android app permission `SYSTEM_ALERT_WINDOW` makes this possible. If you’ve ever used an app like Facebook Messenger or Lastpass, you’ve experienced screen overlay in action. Basically, the permission allows a developer to display content on the screen of your Android device after some trigger event.



For example, you may have seen the cute notification bubble in Facebook Messenger (also called a “chat head” — see the screenshot above) that includes a contact’s photo. If you tap the chat head, it leads to a pop-out version of the Messenger app. Have you noticed that you never gave Facebook Messenger permission to display that bubble?

What Kinds of Testing Do You Need?

WHAT TO READ NEXT:

Use this mobile app security testing checklist to take the attacker’s point-of-view on real iOS and Android devices.

Android Marshmallow (specifically Android 6 API Level 23), introduced the requesting of permissions at runtime. If an app wanted to access your calendar, camera, and other functionalities, the user needed to grant the permission. Unfortunately, that's not always true of `SYSTEM_ALERT_WINDOW`. If a developer targets a lower API level like API level 22 in Android 5 (Lollipop), the Google Play store grants the permission without a runtime prompt. If your device uses Android 6.0.1 or higher and you install an app requesting the `SYSTEM_ALERT_WINDOW` permission, it's granted by default! What this all means is that the only time a user will manually grant the "Draw Over App" permission is if they run Android 6.0.0 and the app is built for API level 23 or above, or when side-loading an app from a third-party Android app store (something we usually recommend against for security's sake).

> In one example of Android overlay malware, a malicious flashlight app actually turned out to be banking trojan malware.

This blows my mind, because if someone were so inclined, they could potentially compromise a user's phone with the SYSTEM_ALERT_WINDOW permission using Android overlay malware. Overlay malware is not a new concept, and the Google Play Store has published a number of malicious apps that abused the Android screen overlay. The apps typically pose as something harmless. In one example of Android overlay malware discovered by ESET, a malicious flashlight app actually turned out to be banking trojan malware.

Potential abuses of the System Alert Window permission

Generally, these attacks play out something like this

1. A user downloads the malicious app
2. A trigger activates the use of the `SYSTEM_ALERT_WINDOW` permission to display content over the top of the UI
3. The screen asks the user for information, such as credentials, and the user divulges that information to make the screen overlay go away

The `SYSTEM_ALERT_WINDOW` permission makes a number of attacks possible, but overlay attacks seem to be the most prominent at this time. Once a malicious app has been installed on your device, the app will wait for the user to launch a target app. When the target app is launched, the malicious app produces an overlay to trick users into entering their credentials into a phony login screen. The user's credentials are then sent to the attacker's server.

The animation below demonstrates what an attack might look like on a victim's phone. You can find the sample malware I used for the demonstration on GitHub: https://github.com/geeksonsecurity/android-overlay-malware-example.

Here's what you're seeing in the video:

1. The user launches the Skype app
2. The phony login screen is overlaid on top of the authentic login page
3. After information is entered into the malicious screen, you see the actual Skype login page

Android overlay attacks and ransomware

With ransomware concerns rising in the wake of the WannaCry attack earlier this month, there have been some suggestions that overlay screens could be used to launch a ransomware attack on Android devices. This is pure speculation, but malware could potentially display an overlay screen demanding credit card information to make it go away. It's possible some users will fall for the ploy when in reality their files weren't encrypted or inaccessible and they could have exited the app via other means.

With that said, the Cloak and Dagger researchers have published a video demonstration of a clickjacking attack that might make such an attack possible. An overlay screen is displayed on top of the UI that includes "holes" that a user can be persuaded to click. Essentially, the user is "clicking through" the overlay screen to certain areas of a UI behind it. The clickjacking technique leverages both the `SYSTEM_ALERT_WINDOW` and `BIND_ACCESSIBILITY_SERVICE`. Such a technique could be repeated multiple times resulting in the user unknowingly clicking the settings interface to grant an app enough permissions culminating in the complete takeover of an Android device.

Also of concern is that nothing currently indicates to an app that another app is overlaying its UI. And this may be possible via apps published on the Google Play store, not a third-party app store. While it might be tempting to call this a bug that can be patched, it seems like a fundamental flaw in Android's permissions design.

How can I protect myself against abuse of the System Alert Window permission?

According to the Cloak and Dagger paper, an estimated 10 percent of apps on the Google Play store use the `SYSTEM_ALERT_WINDOW` permission. While an administrator might tempted to prohibit users from installing apps that request `SYSTEM_ALERT_WINDOW`, doing so would likely result in a staff revolt.

Some of the most popular apps on the store call on the `SYSTEM_ALERT_WINDOW` — not just malware. Users can see what apps have the permission using the instructions in the table below (instructions may not apply to all devices).

Version	**How to review apps with Draw Over Other Apps permission**
Android 7	Settings > Apps > “Gear symbol” > Special Access > Draw over other apps
Android 6	Settings > Apps > “Gear symbol” > Draw over other apps
Android 5	Settings > Apps > Select app and look for “draw over other apps”

In addition, before downloading an app, you can check to see if the app uses the `SYSTEM_ALERT_WINDOW` permission by reading the “Permission Details” on the app’s page in the Google Play Store and looking for a “draw over other apps” bullet.

What makes these `SYSTEM_ALERT_WINDOW`-based attacks so scary is Google’s app vetting process has failed to identify a few of these apps before they’re published on the public app store. Google has, however, started tackling this issue in Android O. There are reports that the beta version of Android O includes a mechanism that will notify users that app overlay is taking place. While this is a step in the right direction, Google might need to consider treating the overlay permission in the same way it treats camera permissions. Still, users stuck on older versions of Android could still fall victim.

Such security issues demonstrate the importance of enterprises vetting third-party mobile apps for security, privacy and compliance issues before employees install them on their mobile devices and continuously monitoring the entire mobile app portfolio for visibility into risks.

[1] *Special thanks to Dan Dumitrescu, security specialist at a financial institution.*
