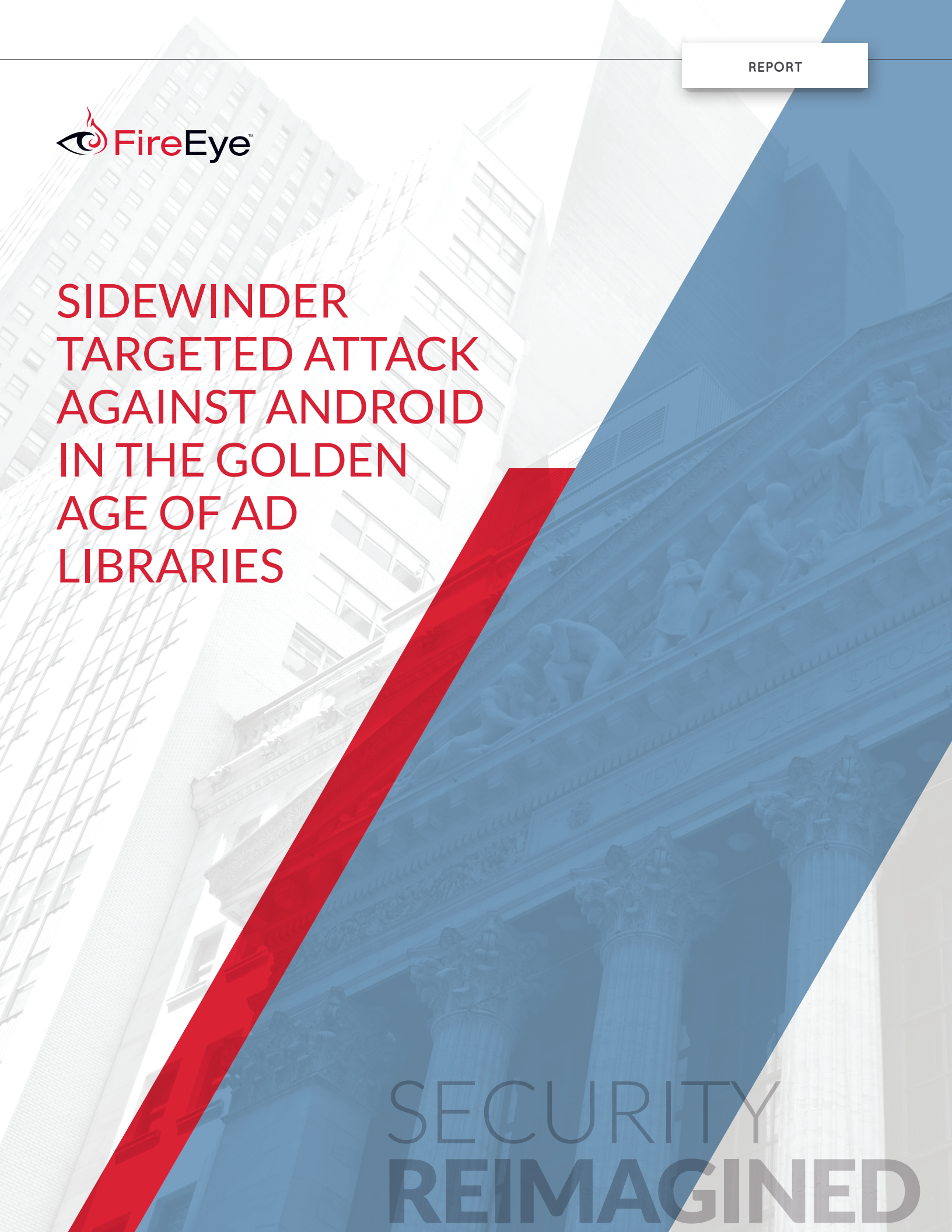
REPORT

FireEye

SIDEWINDER TARGETED ATTACK AGAINST ANDROID IN THE GOLDEN AGE OF AD LIBRARIES



SECURITY REIMAGINED

CONTENTS

Introduction

By 2014, the number of Android users has grown to 1.1 billion and the number of Android devices has reached 1.9 billion[1]. At the same time, enterprises are also embracing Android-based Bring Your Own Device (BYOD) solutions. For example, in Intel's BYOD program, there are more than 20,000 Android devices across over 800 combinations of Android versions and hardware configurations[2].

Although little malware has been found in Google Play, both Android apps and the Android system itself contain vulnerabilities. Aggressive ad libraries also leak the user's private information. By leveraging all these vulnerabilities, an attacker can conduct more targeted attacks, which we call "Sidewinder Targeted Attacks." In this paper we explain the security risks from such attacks, in which an attacker can intercept and use private information uploaded from ad libraries to precisely locate targeted areas such as a CEO's office or specific conference rooms. When the target is identified, a "Sidewinder Targeted Attack" exploits popular vulnerabilities in ad libraries, such as Javascript-binding-over-HTTP or dynamic-loading-over-HTTP, etc.

It is a well-known challenge for an attacker to call Android services from injected native code that doesn't have Android application context. Here, we explain how attackers can invoke Android services for tasks including taking photos, calling phone numbers, sending SMS, reading from/writing to the clipboard, etc. Furthermore, the attackers can exploit several Android vulnerabilities to get valuable private information or to launch more advanced attacks.

Finally, we show that this threat is not only real but also prevalent due to the popularity of Android ad libraries. We hope this paper kickstarts the conversation on how to better protect the security and privacy in third-party libraries and how to further harden the Android security framework in the future.

Sidewinder Targeted Attack Overview

To understand the security risks brought by a Sidewinder Targeted Attack, we first explain one possible attack mechanism (illustrated in Figure 1) that is similar to that of Sidewinder missiles. The attacker can hijack the network where the targeted victim resides. Like an infrared homing system, the attacker then seeks "emission" from ad libraries running on the target device to track and lock on it. Once the target is locked on, the attacker can launch advanced persistent attacks. To minimize detection chances, the attacker can choose to take action on important targets only, ignoring all other devices. In later sections, we discuss attacking ("warhead") and targeting ("homing") components in detail and show how a combination of these components can launch powerful and precise attacks on target devices.

Table 1 proposes different attacks that an attacker can launch remotely on target devices through vulnerable ad libraries. Figure 2 shows a proof-of-concept attack control interface. This attack targets one of the ad libraries described in this paper. The security risks become obvious by looking at what the attacker can do with this control interface. The left panel enables the attacker to command the victim's device,

[1] Ranjit Atwal, Lillian Tay, Roberta Cozza, Tuong Huy Nguyen, Tracy Tsai, Annette Zimmermann, and CK Lu. Forecast: Pcs, ultramobiles and mobile phones, worldwide, 2010-2017, 4q13 update. Gartner, 2013.
[2] Rob Evered, Steve Watson, Paul Dockter, and Derek Harkin. Android devices in a byod environment. Intel White Paper, 2013.

including uploading local files, taking pictures, recording audio/video, manipulating the clipboard, sending SMS, dialing numbers, implanting bootkit, or installing the attacker's apps uploaded to Google Play, etc. The right panel lists all information stolen from the victim's device. In this screenshot, the victim's installed app list, clipboard, a photo taken from the back camera, an audio clip, and a video clip have been uploaded, with the GPS location intercepted from the ad library. The panel also pins down the GPS location of the victim's device onto a Google Map widget.

Figure 1: Illustration of the Sidewinder Targeted Attack Scenario

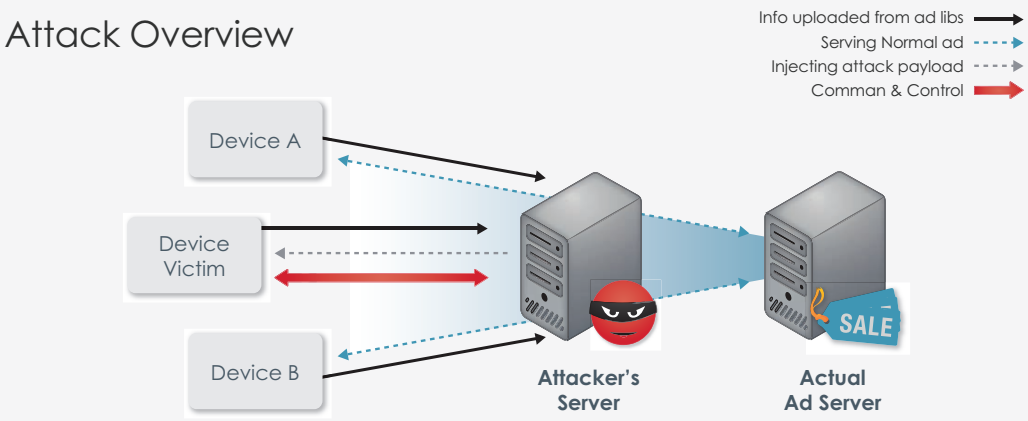
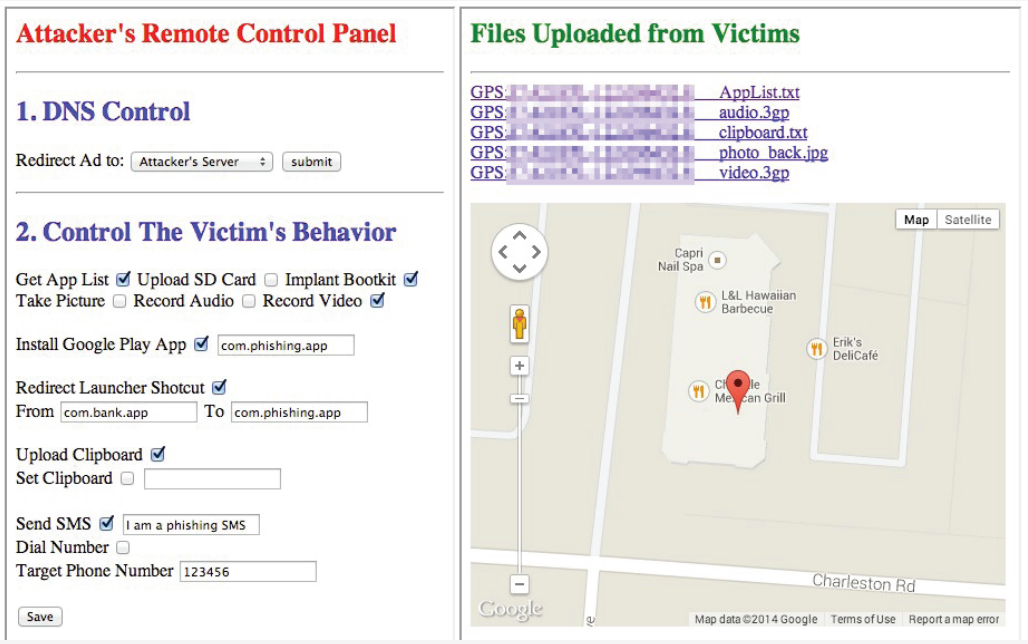


Table 1: Outline of the Sidewinder Targeted Attack through Vulnerable Ad Libraries

API Level	≤ API 16		> API 16
Attack Vector	JBOH and DLOH		JS Sidedoor
Attacks	(w/ Android Context) Clipboard manipulation Launcher settings modification Proxy modification Taking pictures Audio & video recording Stealthy app installation	(w/o Android Context) Local files uploading Root exploit & Code injection Implanting bootkit Sending SMS Making phone calls	Abusing privileged interfaces

Based on this precise position information, it is easy to identify individuals or groups of "VIP" targets by which offices they are in.

Figure 2: The control panel of the attacker, and the files uploaded from the victim



Warhead: Attacking Vulnerabilities of Android

Piercing The Armor

In this section, we explain in more detail the risks of remote attacks on the Android devices.

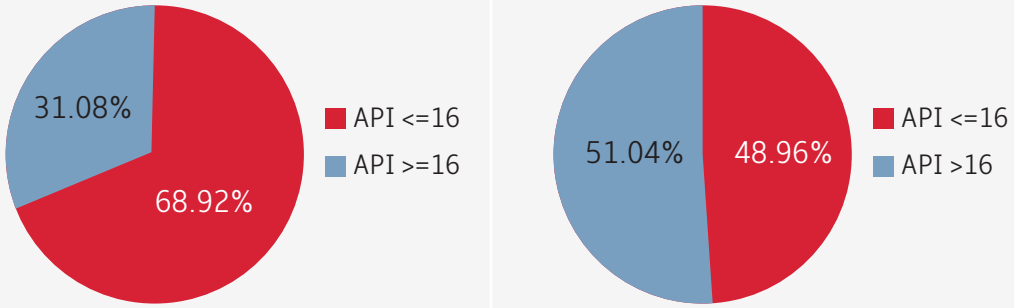
Attacking JavaScript Binding over HTTP (JBOH)

Android uses the JavaScript binding method `addJavascriptInterface` to enable JavaScript code running inside a WebView to access the app's Java methods (also known as the Javascript bridge). However, it is widely known that this feature, if not used carefully, presents a potential security risk when running on Android API 16 (Android 4.1) or below. As noted by Google: "Use of this method in a WebView containing untrusted content could allow an attacker to manipulate the host application in unintended ways, executing Java code with the permissions of the host application."[3]

In particular, if an app running on Android API 16 or below uses the JavaScript binding method `addJavascriptInterface` and loads the content in the WebView over HTTP, an attacker over the network could hijack the HTTP traffic (e.g., through WiFi or DNS hijacking) to inject malicious content into the WebView and to control the host application. Listing 1 is a sample Javascript snippet to execute shell command.

[3] http://developer.android.com/reference/android/webkit/WebView.html# addJavascriptInterface(java.lang.Object,%20java.lang.String).

Figure 3: Target SDK statistics of popular Google Play apps



(a) Statistics by app number

(b) Statistics by app download count

Listing 1: Sample Javascript snippet to execute shell command

```
jsObj.getClass().forName("java.lang.Runtime")
    .getMethod("getRuntime",null).invoke(null,null).exec(cmd)
```

We call this the JavaScript-Binding-Over-HTTP (JBOH) vulnerability[4]. This applies to insecure HTTPS channels as well. If an app containing such vulnerability has sensitive Android permissions such as access to the camera, a remote attacker could exploit it to perform sensitive tasks such as taking photos or recording video, over the Internet, without consent. Based on the official data in June 2014[5], ~60% of Android devices are still running API≤16.

Note that API>16 platforms are not necessarily secure. If the app is targeting at a lower API level, Android will still run it with the lower API level for compatibility reasons. Figure 3 shows the targeted API of popular Google Play apps, each of which has over 50,000 downloads. We can see that a large portion of apps are targeting at API≤16.

Attacking Annotated JavaScript Binding Interfaces

Starting with Android 4.2 (API>16), Google introduced the `@JavascriptInterface` anno- tation[6] to explicitly designate and restrict which public Java methods in the app were accessible from JavaScript running inside a WebView. However, if an ad library uses the `@JavascriptInterface` annotation to expose security-sensitive interfaces, and uses HTTP to load content in the WebView, it is vulnerable to attacks where an attacker over the network could inject malicious content into the WebView to misuse the interfaces exposed through the JS binding annotation. We call these exposed JS binding annotation interfaces "JS Sidedoors."

For example, we found a list of sensitive Javascript interfaces that are publicly ex- posed from certain versions of a real-world ad library:

[4] http://www.fireeye.com/blog/technical/2014/01/js-binding-over-http- vulnerability-and-javascript-sidedoor.html.
[5] https://developer.android.com/about/dashboards/index.html.
[6] http://developer.android.com/reference/android/webkit/ JavascriptInterface.html.

`createCalendarEvent`, `makeCall`, `postToSocial`, `sendMail`, `sendSMS`, `takeCameraPicture`, `getGalleryImage`, `registerMicListener`, etc[4]. Given that this ad library loads ads using HTTP, if the host app has the corresponding permissions (e.g., CALL PHONE), attackers over the network can abuse these interfaces to do malicious things (e.g., utilizing the makeCall interface to dial phone numbers without the user's consent).

Security Issues with DEX Loading over HTTP (DLOH)

Similar to JBOH, DEX loading over HTTP or insecure HTTPS (DLOH) is another serious issue raised by ad libraries. If the attackers can hijack the communication channels and inject malicious DEX files, they can then control the behaviors of the victim apps.

Detonation without Android Context

After getting local access, the attacker can upload private and sensitive files from the victim's device, or modify files that the host app can write to (e.g., the directory of the host app and SD Card with FAT file system).

To launch more sophisticated attacks like sending SMS or taking pictures, the attackers may use Java reflection to call other APIs from the Javascript bridge. It appears this method makes sending SMS easy. However, some other operations require Android context [7] or registering Java callbacks. Android context provides an interface to the global information about an app's environment. Many Android functionalities, especially remote call invocations, are encapsulated in the context. We discuss attacks requiring context in a later section. In this section, we explain attacks that don't need Android context, and discuss their security risks.

Root Exploits and Code Injection

One direct threat posed by JBOH is to use the JBOH shell (Listing 1) to download exe- cutables and use them to root the device. Commercial one-touch root apps claim they can root more than 1,000 brands (>20,000 models) [8]. `towelroot`[9], which exploits a bug found recently in Linux kernel, claims that it can root most new devices released before June 2014. Thus, as long as attackers can get the JBOH shell, they have the tools to obtain root on most Android phone models.

Even if the attackers can't obtain root, they can attempt ptrace[10] to control the host app. Although only processes with root privilege can ptrace others, child processes are able to ptrace their parents. Because the shell launched from the Javascript bridge is a child process of the host app, it can ptrace the host app's process. Note that only apps with `android:debuggable` set as "true" in the manifest can be ptraced, which limits its adoption.

Sending SMS and Dialing Numbers without User Consent

Sending SMS does not require context or user interaction. A simple call does the job, as shown in Listing 2

Listing 2: Sending SMS without user consent

```
SmsManager.getDefault().sendTextMessage(phoneNumber,null,message,null,null);
```

[7] http://developer.android.com/reference/android/content/Context.html.
[8] http://shuaji.360.cn/root/.
[9] http://towelroot.com/.
[10] http://linux.die.net/man/2/ptrace.

To make calls from the Javascript bridge without user consent, we can invoke the telephony service to dial numbers directly via binder, as shown in Listing 3, where phone is the remote Android telephony service and the number 2 represents the second remote call. `s16` is the type marker represents "16 bit string," and `packageName` is the host app's package name, where we can obtain from the information posted from the ad libraries. The sequence number of the remote calls can be found in the corresponding Android Interface

Definition Language (AIDL) files[11]. Many other Android services can be invoked in the same way, including sending SMS

Detonation with Android Context

As mentioned, it is more convenient to directly obtain the Android context via the Javascript bridge. Code in Listing 4, for example, is an easy way to get context from anywhere of the application.

Operations like taking pictures and recording videos need to register Java callbacks. The attackers either need to boot a Java VM from the Javascript bridge, or to inject code into the host app's Java VM.

Fortunately, Android Runtime offers another way to load Java Native Interface (JNI) code into the host app using `Runtime.load()`. As shown in Listing 5, an attacker can load executables compiled from JNI code. Once loaded, the code can obtain context as described in Listing 4, or call `DexClassLoader load`[12] to inject new classes from the attackers' DEX files to register callbacks to take pictures/ record videos.

Listing 3: Dial numbers without user consent

```
Runtime.getRuntime()
    .exec("service call phone 2 s 16 "+ packageName +" s16" + phoneNumber);
```

Listing 4: Sample code to obtain context

```
// We omit all try-catch statements and other unimportant code in this paper

public ContextgetContext(){
    finalClass<?>activityThreadClass=Class
        .forName("android.app ActivityThread");
    finalMethodmethod=activityThreadClass
        .getMethod("currentApplication");
    return(Application)method.invoke(null,(Object[])null);
}
```

[11] http://developer.android.com/guide/components/aidl.html.
[12] http://developer.android.com/reference/dalvik/system/DexClassLoader. html.

There are other ways to obtain Android context, like reflecting to the private static context variable of WebView[13]. However, without Java VM instances, it's difficult to take pictures and record videos. After our submission to Black Hat in April 2014, we noticed that MWR was also concurrently and independently working on this issue. They published a similar mechanism in June 2014[14].

Clipboard Monitoring nd Tampering

With the Android context, an attacker can monitor or tamper with the clipboard. Android users may perform copy-paste on important text content. For example, there are many popular password-management apps in Google Play, enabling the users to click-and-copy passwords and paste them into login forms. Malicious apps can steal the passwords if they can read the contents on clipboard. Android has no permissions restricting apps from accessing the global clipboard. Any UID has the capability to manipulate clipboard via the API calls in Listing 6:

Using these APIs, the attackers can monitor changes to a clipboard and transfer the clipboard contents to some remote server. They can also alter the clipboard content to achieve phishing goals. For example, the user may copy a link to visit and the background malicious service can change that link to a phishing site. We have notified Google about this issue.

Launcher Settings Modification

Android Open Source Project (AOSP) classifies Android permissions into several protec- tion levels: "normal,""dangerous," "system," "signature" and "development"[15,16,17]. Dangerous permissions"may be displayed to the user and require confirmation before pro- ceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities."In contrast, normal permissions are automatically granted at installation, "without asking for the user's explicit approval (though the user always has the option to review these

Listing 5: Sample Javascript snippet to load JNI binary into the host app's Java VM

```
jsObj.getClass().forName("java.lang.Runtime")
    .getMethod ("getRuntime",null).invoke(null,null).load(binaryPath );
```

Listing 6: API calls to peek into/ tamper with the clipboard

```
ClipboardManager.getText()
ClipboardManager.hasPrimaryClip()
ClipboardManager.setText()
Clipboard Manager.setPrimaryClip()
ClipboardManager.hasText()
ClipboardManager.addPrimaryClipChangedListener()
ClipboardManager.getPrimaryClip()
```

[13] http://www.weibo.com/p/1001603724694418249344?utm_source=weibolife.
[14] https://labs.mwrinfosecurity.com/blog/2014/06/12/putting-javascript- bridges-into-android-context.
[15] http://developer.android.com/guide/topics/manifest/permission-element.html.
[16] https://android.googlesource.com/platform/frameworks/base/+/master/ core/res/AndroidManifest.xml.
[17] https://android.googlesource.com/platform/packages/apps/Launcher2/+/ master/AndroidManifest.xml.

permissions before installing)"[15]. If an app requests both dangerous permissions and normal permissions, Android only displays the dangerous per- missions by default. If an app requests only normal permissions, Android doesn't display any permission to the user.

We have found that certain "normal" permissions have dangerous security impacts[18]. For example, the attackers can manipulate Android home screen icons using two normal permissions: launcher READ SETTINGS and WRITE SETTINGS permissions. These two permissions enable an app to query, insert, delete, or modify all launcher configuration settings, including icon insertion or modification.

As a proof-of-concept attack scenario, a malicious app with these two permissions can query/insert/alter the system icon settings and modify legitimate icons of some security- sensitive apps, such as banking apps, to a phishing website.

After our notification, Google has patched this vulnerability in Android 4.4.3 and has released the patch to its OEM partners. However, according to Google[5], by 7 July 2014, 17.9% Android devices are using Android 4.4. Given that Android 4.4.2 and below has this vulnerability, over 82.1% Android devices are vulnerable.

Proxy Modification

With the CHANGE WIFI STATE permission, Android processes can change the proxy settings of WIFI networks (not solely the currently connected one). To do this, the attacker can use the remote calls exposed by `WifiManager` to obtain the `WifiConfiguration` objects, then create new `proxySettings` to replace to a corresponding field. Note that the `proxySettings` field is a private Java field not intended to be accessed by other processes. Unfortunately, the flexible and powerful Java reflection mechanism (especially the `forName()`, `getField()`, `setAccessible()` calls) exposes such components to the attackers for arbitrary read or write operations.

Taking Pictures and Recording Audio/Video without User Interaction

Android audio recording via the `MediaRecorder` APIs does not need user interaction or consent, which makes it easy to record sound in the background.

On the contrary, taking pictures and recording videos are more challenging. First, this requires registering Java callbacks. Second, Android warns that "Preview must be started before you can take a picture"[19]. It seems that taking pictures and recording videos without user notification is impossible. However, security largely depends on the correct implementation and enforcing a flawless implementation is difficult. On some of the popular phones (models anonymized for security consideration), `startPreview()` is required to take pictures/record videos; However, it's highly possible that on these devices `takePicture()` fails to check whether a view has been presented to the user. Fortunately, we have never witnessed a case where the MediaRecorder can shoot videos without calling `setPreviewDisplay`. But we were able to create and register a dummy `SurfaceView` to the `WindowManager`, which made taking photos and videos possible even on devices that properly checked for an existing preview.

[18] http://www.fireeye.com/blog/technical/2014/04/occupy_your_icons_silently_on_android.html.
[19] http://developer.android.com/reference/android/hardware/Camera.html.

Stealthy App Installation by Abusing Credentials

With both the GET ACCOUNTS and the USE CREDENTIALS permissions, Android pro- cesses can get secret tokens of services (e.g., Google services) from the `AccountManager` and use them to authenticate to these services[20]. We verified that Android apps with these two permissions can authenticate themselves with the user's Google account, allowing access to Google Play and the ability send app installation requests. Through the Javascript bridge, attackers can install apps of choice (e.g., an attacker's phishing app) to any devices registered in user's account in the background without user consent. Combined with the launcher modification attack introduced earlier, the attackers can redirect other app icons (e.g., bank or email app icons) to the phishing app and steal the user's login credentials.

Targeting Victims Based on Ad Traffic

In this section, we explain the risks of victims' devices being tracked and targeted through ad traffic.

Communication Channels Prone to Hijack

It is well known that communication via HTTP is prone to hijacking and data tamper- ing. Though ad libraries may not have the incentive to abuse users' private and sensitive data, this is not the case with the attackers eavesdropping or hijacking the HTTP traffic. Switching to HTTPS may not solve this issue since the HTTPS security relies on a flawless implementation, which is difficult. For example, there are cases where the developer failed (intentionally or unintentionally) to check the server's certificate[21]. We found that some of the most popular ad libraries (see Table 3) have this issue. We successfully launched Man-in-the-Middle (MITM) attacks and intercepted the data uploaded to the remote server. Note that even if the ad libraries have a correct and rigorous implementation, the SSL library itself may contain serious vulnerabilities that can be exploited by MITM attacks[22,23].

Information Leakage from Ad Libraries

Almost every ad library uploads local information from Android devices. Based on our observations, they do so mostly for purposes such as checking for platform compatibility and user interest targeting. The information most frequently uploaded includes IMEI, Android version, manufacturer, Android ID, device specification, carrier information, host app information, installed app list, etc. Table 3 lists the info uploaded from the top five popular ad libraries.

Listing 7 is a captured packet posted to the remote ad server by one of the ad libraries. It is

Listing 7: API calls to peek into/ tamper with the clipboard

```
requestactivity=AdRequest&d-device-screen-density=1.5&d-device-screen-
size=320X533&u-appBId=com.example.app&u-appDNM=Example&u-appVer=1.2&h-user-
agent=Mozilla
%2F5.0+%28Linux%3B+U%3B+Android+4.1.2%3B+en-us%3B+sdk+Build%2FMASTER%
29+AppleWebKit%2F534.30+%28KHTML%2C+like+Gecko%29+Version%2F4.0+Mobile+Safari
%2F534.30&d-localization=en_us&d-netType=umts&d-orientation=1&u-latlong-accu=
37.410835%2C-121.920514%2C
```

[20] http://seclists.org/bugtraq/2014/Mar/52.
[21] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumga¨rtner, Bernd Freisleben, and Matthew Smith. Why eve and mallory love android: An analysis of android ssl (in) security. In **Proceedings of the 2012 ACM conference on Computer and communications security**, pages 50–61. ACM, 2012.
[22] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0224.
[23] http://www.fireeye.com/blog/technical/2014/04/if-an-android-has-a- heart-does-it-bleed.html.

captured from a popular Google Play app. From this packet we can tell the device's screen density (d-device-screen-density), screen size (d-device-screen-size), host app's pack- age name (u-appBId), host app's name (u-appDNM)1, host app's version (u-appVer), user agent (h-user-agent), localization (d-localization), mobile network type (d-netType), screen orientation (d-orientation), and GPS location (u-latlong-accu). The most important infor- mation is the GPS location, where the victim's latitude, longitude and the location precision are shown. It is reasonable for an ad to obtain this information to improve the ad-serving experience. However, with this information, an attacker can precisely locate the victim and acquire the device's specifications.

Large-scale Monitoring and Precise Hijacking

To locate victims effectively, an attacker needs to monitor large-scale network traffic containing such private information. Unfortunately, several well-known attacks can be used to achieve large-scale monitoring, including DNS hijacking, BGP hijacking, and ARP hijacking in IDC.

In this context, DNS hijacking is done to subvert the resolution of Domain Name System (DNS) queries through modifying the behavior of DNS servers so that they serve fake DNS information.

DNS hijacking is legally and maliciously used in many situations including traffic management, phishing and censorship. Attackers successfully compromised many DNS servers, including the ones from Google and Godaddy[24]. By DNS hijacking, attackers can effectively access all the traffic to ad servers.

BGP hijacking takes over groups of IP addresses, corrupting Internet routing tables by breaking BGP sessions or injecting fake BGP information. This enables attackers to monitor all traffic to specific IPs. Historically, there were many BGP hijacking attacks that affected YouTube, DNS root servers, Yahoo, and many other important Internet services[25].

ARP hijacking (or spoofing) in IDC[26] is done to hijack the traffic to the ad server in the IDC where the ad server locates through fake ARP packets. Attackers may rent servers close to the target servers, and use fake ARP packets to direct all the traffic to go through the hijacking servers first for monitoring and hijacking..ARP hijacking is a well-known approach used in network attacks.

Using the large-scale traffic intercepted from the above methods, attackers can iden- tify potential victims based on information leakage such as GPS

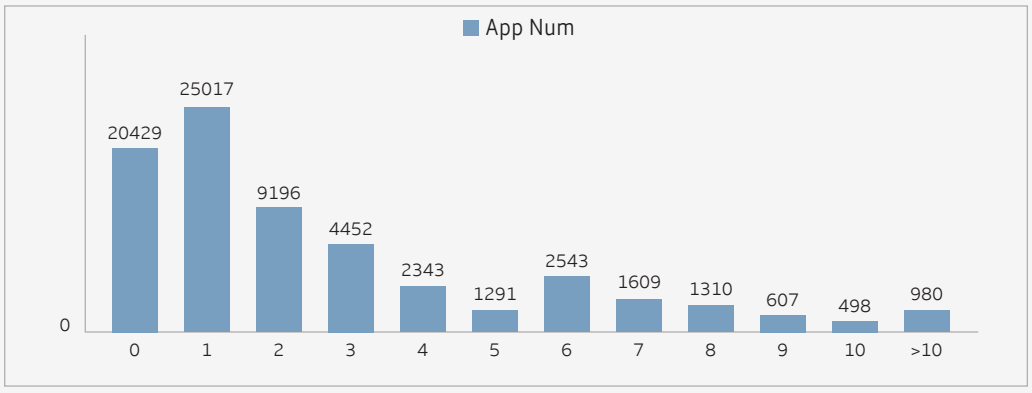


Figure 4: Number of ad libraries included in Google Play apps (with more than 50,000 downloads

[24] https://isc.sans.edu/diary/Domaincontrol+(GoDaddy)+Nameservers+DNS+ Poisoning+/5146.
[25] http://www.networkworld.com/article/2272520/lan-wan/six-worst-internet- routing-attacks.html.
[26] http://en.wikipedia.org/wiki/ARP_spoofing.

location described in Sec- tion 4.2. After that, they can inject exploits only into the targeted traffic to launch further attacks. Attackers keep a low profile by allowing all other irrelevant network traffic to pass without being modified.

Targetable and Exploitable Google Play Apps

We used the FireEye Mobile Threat Prevention (MTP) engine to analyze all of the ~73,000 popular apps from Google Play with more than 50,000 downloads, and identified 93 ad libraries. The detailed ad library inclusion statistics are shown in Figure 4. Seventy-one% of the apps contain at least one ad library, 35% have at least two ad libraries, and 22.25% include at least three ad libraries. The largest ad inclusion number is 35. Since Google is cautious about the security of the products it directly controls, we exclude Google Ad from the following discussion. For security considerations, in this paper we anonymize the names of the other 92 ad libraries, using Ad1, Ad2, ..., Ad92 to refer to them, where the subscripts represent the rankings of how many apps include the ad libraries. The top five popular ad libraries' inclusion and download statistics are listed in Table 2.

We analyzed the 92 ad libraries found in the popular Google Play apps, and summa- rized the communication channel vulnerabilities in Table 3. Combined with the uploaded information column we can learn about the data the attackers can obtain.

Fifty-seven of the 92 ad libraries in the popular Google Play apps have the JBOH issue. Specifically, four of the top five ad libraries are subject to this problem (shown in Table 2). Seven of the 92 ad libraries are prone to DLOH attacks. Particularly, some versions of Ad5 in Table 3 have this problem. The affected Google Play apps number and the accumulated download counts are listed in Table 4.

Table 2: The inclusion statistics of the top five Android ad libraries excluding Google Ad. Their JBOH statistics are also listed (discussed in the earlier JBOH section.).

Ad Library	Number of Apps	JBOH Apps	Total Downloads	JBOH Downloads
Ad_1	9,702	2,802	8,781M	2,348M
Ad_2	8,856	4,204	7,865M	4,754M
Ad_3	8,818	2,117	8,499M	1,611M
Ad_4	5,519	1,112	4,687M	617M
Ad_5	5,170	0	4,519M	0

Table 3: The uploaded data, communication channel vulnerabilities, and JBOH/DLOH details of the top five ad libraries.

Ad Library	Uploaded Info	Protocol	SSL Vuln	JBOH	DLOH
Ad_1	IMEI/device id, device model, Android version, location	HTTP/ HTTPS	●	●	
Ad_2	device specification, Android version, host app info, location	HTTP		●	
Ad_3	IMEI/device id, device model, Android version, device manufacturer, carrier info, location, ip	HTTP		●	
Ad_4	IMEI/device id, device model, device specification, Android version	HTTP		●	
Ad_5	IMEI/device id, device model, device specification, Android version, coun- try, launguage	HTTPS	●		●

Table 4: Assessment statistics of Google Play apps (downloads ≥50,000) that are vulnerable to the Sidewinder Targeted Attack. Type I apps are those subject to JBOH or DLOH attacks; Type II apps are those not only JBOH/DLOH exploitable but also have the LOCATION leakage (thus vulnerable to the Sidewinder Targeted Attack). Note that an app is counted in the total statistics if it is subject to any of the attacks, including uploading files and root exploits.

Subject to attack type	Type I #	Type I Downloads	Type II #	Type II Downloads
Code injection via ptrace	2,055	444M	272	67M
Send SMS	349	340M	229	254M
Make phone calls	572	399M	426	324M
Launcher modification	111	95M	81	37M
Proxy modification	644	792M	419	378M
Record audio	1,097	1,408M	654	621M
Take pictures/record videos	1,141	1,380M	622	665M
Install apps stealthily	351	552M	197	332M
Total(incl. root exploits)	**16,579**	**11,706M**	**4,201**	**3,207M**

Conclusion

In the current golden age of Android ad libraries, Sidewinder Targeted Attacks can target victims using info leakage and other vulnerabilities of ad libraries to get valuable, sensitive information. Millions of users are still under the threat of Sidewinder Targeted Attacks. First we need to improve the security and privacy protection of ad libraries. For example, we encourage ad libraries' publishers to use HTTPS with proper SSL certificate validation, and to properly encrypt network traffic. They also need to be cautious about which privileged interfaces are exposed to the ad providers, in case of malicious ads or attackers hijacking the communication channels.

Sidewinder Targeted Attacks can target victims using info leakage and other vulnerabilities of ad libraries to get valuable, sensitive information. Millions of users are still under the threat of Sidewinder Targeted Attacks.

Meanwhile, Google itself needs to further harden the security framework. This may prove difficult because:

1. Android is a complex system. Any sub-component's vulnerability may impact the security of the whole system. Fragmentation makes the situation even more challenging.

2. The trade-off between usability, performance and security always matters, and market demand frequently dictates that security comes last. Many Android developers do not even understand how to program securely (as shown in the JBOH issue).

3. Many security patches are not back-ported to old versions of Android (like the launcher settings problem described earlier), even though older versions are widely used.

4. There is always information asymmetry in the development chain. For example, it usually takes several months for vendors to apply security patches after Google releases them.

Albeit challenging, we hope that this work can kickstart a conversation, both on improved security and privacy protection in third-party libraries and on a hardened Android security framework.

About FireEye, Inc.

FireEye has invented a purpose-built, virtual machine-based security platform that provides real-time threat protection to enterprises and governments worldwide against the next generation of cyber attacks. These highly sophisticated cyber attacks easily circumvent traditional signature-based defenses, such as next-generation firewalls, IPS, anti-virus, and gateways. The FireEye Threat Prevention Platform provides real-time, dynamic threat protection without the use of signatures to protect an organization across the primary threat vectors and across the different stages of an attack life cycle. The core of the FireEye platform is a virtual execution engine, complemented by dynamic threat intelligence, to identify and block cyber attacks in real time. FireEye has over 1,900 customers across more than 60 countries, including over 130 of the Fortune 500.

FireEye, Inc. | 1440 McCarthy Blvd. Milpitas, CA 95035 | 408.321.6300 | 877.FIREEYE (347.3393) | info@fireeye.com | **www.fireeye.com**

FireEye