INSIDE THE IOS/ADTHIEF MALWARE

Axelle Apvrille
Fortinet, France

Surprisingly (or maybe not), iOS malware isn’t very common. At the end of 2013, there were only four different families (Ikee, FindCall, Toires and Trapsms) as well as a dozen families of adware or spyware [1]. Ikee and Trapsms require jailbroken devices, whereas FindCall and Toires work on any device.

Thus, the discovery of new iOS malware is generally pretty hot news for an anti-virus analyst. In March 2014, Claud Xiao discovered iOS/AdThief, a.k.a. Spad, a piece of malware which hijacks advertisement revenues and redirects them to the attacker.

However, very little information was published at the time, and the little that was published [2, 3] was difficult to understand (even for technical readers). This paper attempts to provide a clear description of the virus. In doing so, it also provides some tips for reversing iOS malware, and a few new findings are also disclosed.

THE MALWARE’S GOAL

iOS/AdThief.A!tr works on jailbroken iOS devices. It implements a Cydia Substrate [4] extension to hijack the revenues from advertisements on the infected device. In other words, each time you view or click an ad on an infected device, the corresponding revenue goes to the attacker, and not to the developer or the legitimate affiliate.

Cydia Substrate, which only works on jailbroken devices, is a platform for modifying existing processes. It provides an API to hook the legitimate functions, and you can add your own tweaks. This is exactly what the malware does: it hooks various advertisement functions and modifies the developer ID (a.k.a. promotion ID) to match that of the attacker (see Figure 1).

TARGETED ADKITS

A list of mobile adkits targeted by the malware is provided in [2]: YouMi, Vpon, MobClick, Umeng, AdSage/MobiSage, MdotM, InMobi, Domob, AdWhirl, AdsMogo, Google Mobile Ads SDK, AderMob, Weibo, MIX SDK and Poly SDK. The majority of these are Chinese, four are based in the US, and two in India.

In [2], Xiao remarks that Weibo is a popular social network in China, but is unable to attribute MIX SDK and Poly SDK more precisely. In fact, Sina Weibo, introduced in 2013, is an advertisement SDK [5], so that solves one mystery. MIX SDK can be attributed to GuoHeAD. It probably refers to the GuoHe MIX platform for cross-promotion of mobile games [6]. This is also backed up by the name of a source file found in the malware: /Volumes/MacOsStore/Project/IOS/SpAd/SpAd/AD_GuoHe.xm.

Finally, Poly SDK is not a new adkit: it corresponds to AderMob. This is confirmed when downloading the AderMob iOS SDK [7].

Closer inspection of the malware shows that, as well as the adkits included on the list in [2], it also supports Komli Mobile. This hadn’t been identified earlier because the hook names were quite generic (‘APIManager’). However, when searching for strings in the malware, we noticed that the name of the source file of each adkit hook is shown close to the functions it hooks. For example, the source filename ‘AD MobiSage.xm’ appears next to hooks to MobiSageAdBanner and MobiSageAdPoster. Since ‘AD KomliMobile.xm’ appears next to ‘APIManager’ hooks, and that name has not yet been attributed to another adkit, we assume that ‘APIManager’ corresponds to a class name of the Komli Mobile SDK.

Figure 1: iOS/AdThief!tr hijacks advertisement revenues and redirects them to accounts owned by the attackers.
In total, therefore, the malware hijacks advertisements from 15 different adkits (see Table 1).

<table>
<thead>
<tr>
<th>Adkit source</th>
<th>Filename</th>
<th>Typical class names</th>
</tr>
</thead>
<tbody>
<tr>
<td>AderMob</td>
<td>AD Ader.xm</td>
<td>AderSDK*</td>
</tr>
<tr>
<td>AdMob and Google Mobile Ads SDK</td>
<td>AD AdMob.xm</td>
<td>GAD*</td>
</tr>
<tr>
<td>AdsMogo</td>
<td>AD AdsMongo.xm</td>
<td>AdMoGo*</td>
</tr>
<tr>
<td>AdWhirl</td>
<td>AD AdWhirl.xm</td>
<td>AdWhirl*</td>
</tr>
<tr>
<td>Domob</td>
<td>AD DoMob.xm</td>
<td>DM*</td>
</tr>
<tr>
<td>GuoHeAD</td>
<td>AD GuoHe.xm</td>
<td>MIXView*</td>
</tr>
<tr>
<td>InMobi</td>
<td>AD InMobi.xm</td>
<td>IMAd*</td>
</tr>
<tr>
<td>Komli Mobile</td>
<td>AD KomliMobile.xm</td>
<td>APIManager*</td>
</tr>
<tr>
<td>MdotM</td>
<td>AD MdotM.xm</td>
<td>MdotM*</td>
</tr>
<tr>
<td>MobClick</td>
<td>?</td>
<td>MobClick*</td>
</tr>
<tr>
<td>U Meng</td>
<td>AD UMeng.xm</td>
<td>UMUFP*</td>
</tr>
<tr>
<td>Vpon</td>
<td>AD Vpon.xm</td>
<td>VponAdOn*</td>
</tr>
<tr>
<td>Weibo</td>
<td>AD Weibo.xml</td>
<td>DXAdHWB*_delegated  to Google Ads</td>
</tr>
<tr>
<td>YouMi</td>
<td>AD Youmi.xm</td>
<td>YouMi*</td>
</tr>
</tbody>
</table>

Table 2: Implementation details of adkit hooks found in iOS/AdThief.

Figure 2: On a pediy.com forum, Rover12421, a.k.a. zerofile, admits to being the creator of the malware (translated from Chinese).
Figure 2). His answers aren’t very clear, but it seems he only wrote a basic ad ID replacement plug-in, and that someone else improved the code (see Figure 3). He denies having participated in the propagation of the malware.

According to [3], the malware is estimated to have infected 75,000 devices, stealing revenue from around 22 million ads.

Figure 3: Private email conversation with Rover12421.

IMPLEMENTATION

Overview

The main parts of iOS/AdThief’s implementation are detailed in [2]. However, Xiao’s post requires some familiarity with iOS, Substrate and Objective C. Here, we attempt to provide a clearer description of the implementation.

Each time an end-user views or clicks on a given advertisement, the corresponding application developer (or partner, or affiliate) receives a small payment. This is what advertisement companies refer to as ‘cost per thousand impressions’ (CPM) or ‘click-through rate’ (CTR). To credit the right developer when ads are viewed or clicked, adkits identify developers (or partners etc.) with a developer ID.

iOS/AdThief modifies this developer ID, replacing it with an identifier owned by the attacker. Revenues are consequently hijacked, with all of the revenue generated when an ad is viewed or clicked being assigned to the attacker’s identifier.

To modify the developer ID, the malware author implements a hook for each of the adkits he wants to hijack, where he replaces the developer ID as he wishes. To do so, he takes advantage of an existing process-hooking platform, Substrate, which is available on jailbroken devices. To implement a Substrate hook, one uses a function named ‘MSHookMessageEx’ with four arguments:

1. The class to hook.
2. A ‘selector’, i.e. the name of the function to hook.
3. The address of the replacement function, i.e. the hook.
4. A stub to call the old replaced function in case it is needed. (It may be left as NULL if not needed.)

Registering a hook

For example, in the code shown in Figure 4, the author creates a YouMiView class object (line 1). Then he calls objc_msgSend on that object (line 6).

In Objective C, one does not ‘call’ a function, but instead ‘send a message to a function’ [8]. Consequently, each function call consists of a call to objc_msgSend (which is why objc_msgSend appears so frequently).

So, the author sends a message to ‘instancesRespondToSelector:’ with the argument ‘adViewWithContentSizeIdentifier:delegate:’. For those more familiar with C or Java, this translates to:

```objc
objc_msgSend((id)adView, @selector(instancesRespondToSelector:));
```

InstancesRespondToSelector [9] is a standard iOS function that tests ‘whether instances of the receiver are capable of responding to a given selector’, i.e.
whether the instance implements a function named
‘adViewWithContentSizeIdentifier:delegate:’. If such a function exists, the malware author decides to
hook it (lines 6–7). He calls MSHookMessageEx (line 8),
says he wants to hook adViewWithContentSizeIdentifier:
delegate:, and that his implementation of the hook will
be in _logos_meta_method_ungrouped_YouMiView_
adViewWithContentSizeIdentifier_delegate_ (lines 11–12).

Implementing a hook
Continuing the previous example, the implementation of the
YouMi hook is in a function named logos_meta_method_ungrouped_YouMiView_
adViewWithContentSizeIdentifier_delegate_. Its code is shown in Figure 5.

At line 11, the malware author instantiates a
‘GADBannerView YouMi’ object. GAD stands for Google
Ads, and indeed if we inspect the code of this function,
we see that it corresponds to the AdMob kit, not YouMi.
Furthermore, online documentation for GADBannerView
exists in the AdMob SDK [10]. Next, the author initializes
the banner view by sending the object a message
‘initWithAdSize:origin:’.

The important part lies in line 14. Onto this Google Ads
banner, the author sets the ‘unit identifier’ (AdMob Publisher
ID) to his preferred ID (a1521215ab55cd2 in this particular
case). Then, he sets the root window (provided as the fourth argument – a4) and that same window as a delegate (line 16). Delegation is a mechanism by which a given object can
act on behalf of another one, and coordinate with it. This
explains the relationship between Google Ads and YouMi:
in this particular case, the author does not directly hook
YouMi ads, but Google ads, which act under the delegation
of YouMi:

Consolidated data
We have described the details of YouMi revenue hijacking.
The hijacking of other adkits is achieved using the same
strategy:
• registering one or multiple hooks for the adkit
• implementing each hook.

Like YouMi, three other adkit hooks work when Google Ads
acts under delegation: DoMob, Umeng and Weibo. The other
adkits work on their own – without delegation.

CONCLUSION
iOS/AdThief is a technical and malicious piece of code
which hijacks revenue from 15 different adkits. It is built on
top of the Cydia Substrate platform, available for jailbroken
devices, which provides it with an easy way to modify
advertisement SDKs. With Substrate, the malware needs only
to focus on the call and implementation of each hook.

At first, the identification of every adkit the malware targets
was difficult because the code mentions only class names
used by each adkit SDK. However, the fact that the malware
author did not strip out debugging information helped us to
identify all 15 adkits. In particular, this is how support for
Komli Mobile and GuoHeAD was detected.

The debugging information also helped us to track down the
malware’s creator, a Chinese hacker specializing in mobile
platforms. The hacker claims to have written parts of the
code some time ago, but that a third party then improved
it. He denies having participated in the spreading of the
malware. With 75,000 infected devices, iOS/AdThief is not
extremely prevalent. However, there are an estimated 22
million hijacked ads, so the malware has probably had a fair
amount of impact and generated significant revenue for the
owner(s).

ACKNOWLEDGEMENTS
I would like to thank Claud Xiao, Tim Strazzere, ‘baronpan7’
and Dong Xie for their help with this research.
REFERENCES


