Update: XcodeGhost Attacker Can Phish Passwords and Open URLs through Infected Apps

By Claud Xiao

Published: 2015-09-18 · Archived: 2026-04-06 03:32:32 UTC

On Thursday we posted the initial analysis report on XcodeGhost malware and then found it had infected 39 iOS apps, potentially impacting hundreds of millions of users. XcodeGhost embedded malicious code into those infected iOS apps. In the first report, we noted that the malicious code uploads device information and app information to its command and control (C2) server. But that isn't all it does.

Today, inspired by a post by "@Saic" on Sina Weibo, we analyzed the malicious code in more detail and found additional capabilities in the malware. In summary, the malicious code that XcodeGhost embedded into infected iOS apps is capable of receiving commands from the attacker through the C2 server to perform the following actions:

- Prompt a fake alert dialog to phish user credentials;
- Hijack opening specific URLs based on their scheme, which could allow for exploitation of vulnerabilities in the iOS system or other iOS apps;
- Read and write data in the user's clipboard, which could be used to read the user's password if that password is copied from a password management tool.

*(**UPDATE** September 21: In the current version of the code, XcodeGhost cannot be directly used to phish iCloud passwords. However, by changing a few simple lines of code, it can do that.)*

Based on this information, we believe XcodeGhost is a very harmful and dangerous malware that has bypassed Apple's code review and made unprecedented attacks on the iOS ecosystem. The techniques used in this attack could be adopted by criminal and espionage focused groups to gain access to iOS devices.

Technical Details

XcodeGhost added code to some system APIs that are used by the infected apps. After the malware sends device and app information to it's C2 servers, XcodeGhost will decrypt the content returned by the server and parse it as a piece of JSON formatted data.

```
v86 = objc_msgSend(&OBJC_CLASS___NSUserDefaults, "standardUserDefaults");
v87 = (void *)objc_retainAutoreleasedReturnValue(v86);
v88 = objc_msgSend(v87, "objectForKey:", CFSTR("SystemReservedData"));
v89 = objc_retainAutoreleasedReturnValue(v88);
v90 = objc_msgSend(&OBJC_CLASS___NSMutableData, "dataWithData:", v89);
v91 = (void *)objc_retainAutoreleasedReturnValue(v90);
objc_release(v89);
if ( (unsigned int)objc_msgSend(v91, "length") >= 9 )
{
  v190 = v85;
  v92 = (char *)objc_msgSend(v91, "length") - 8;
  v189 = v91;
  v93 = objc_msgSend(v91, "subdataWithRange:", 8, v92);
  v94 = objc_retainAutoreleasedReturnValue(v93);
  v95 = objc_msgSend(v192, "Decrypt:", v94);
  v96 = objc_retainAutoreleasedReturnValue(v95);
  v226 = 0;
  v97 = objc_msgSend(&OBJC_CLASS___NSJSONSerialization, "JSONObjectWithData:options:error:", v96, 1, &v226);
  v191 = (void *)objc_retainAutoreleasedReturnValue(v97);
  v188 = objc_retain(v226);
```

Figure 1. XcodeGhost decrypts response JSON data

In the JSON data, XcodeGhost will look for these keys:

- alertHeader
- alertBody
- appID
- cancelTitle
- confirmTitle

The malware uses the specified title and body texts to create a fake alert dialogue box.. Using this technique, XcodeGhost can be used to “phish” information from the user, or trick them into inputting sensitive data. For example, it can create a dialog that asks the victim to input their password. Since the dialog is a prompt from the running application, the victim may trust it and input a password without suspecting foul play.

```
v1 = a1;
v2 = objc_msgSend(&OBJC_CLASS___UIAlertView, "alloc");
v3 = objc_msgSend(
       v2,
       "initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:",
       *(_DWORD *)(v1 + 20),
       *(_DWORD *)(v1 + 24),
       *(_DWORD *)(v1 + 28),
       *(_DWORD *)(v1 + 32),
       *(_DWORD *)(v1 + 36),
       0);
objc_msgSend(v3, "show");
v4 = objc_msgSend(*(void **)(v1 + 40), "integerValue");
objc_msgSend(v3, "setTag:", v4);
return objc_release(v3);
```

Figure 2. XcodeGhost prompts a alert dialog with specific title and message text

If the returned JSON data from the server contains the key “url”, XcodeGhost will open the URL specified.

```
v3 = objc_retain(a3);
v4 = objc_msgSend(&OBJC_CLASS___UIApplication, "sharedApplication");
v5 = (void *)objc_retainAutoreleasedReturnValue(v4);
v6 = objc_msgSend(v5, "applicationState");
objc_release(v5);
if ( !v6 )
{
  v7 = objc_msgSend(&OBJC_CLASS___UIApplication, "sharedApplication");
  v8 = (void *)objc_retainAutoreleasedReturnValue(v7);
  v9 = objc_msgSend(&OBJC_CLASS___NSURL, "URLWithString:", v3);
  v10 = objc_retainAutoreleasedReturnValue(v9);
  objc_msgSend(v8, "openURL:", v10);
  objc_release(v10);
  objc_release(v8);
}
return objc_release(v3);
```

Figure 3. XcodeGhost opens remotely specified URL

Note that the specified URL doesn’t have to be only HTTP or FTP URLs but also the URLs used by iOS system with any scheme that local system can handle. (i.e. itunes:// or twitter://) The URL scheme is one of the main Inter-App Communication mechanisms in iOS system. Any iOS apps, include system apps, can define any scheme it can handle. Multiple previous vulnerabilities in iOS and various iOS apps have been caused by scheme handling flaws that can be exploited by opening specific URLs. These vulnerabilities must be exploited locally by a

malicious app, which led many people to not treat them as serious as iOS malware is uncommon. XcodeGhost has broken this assumption by infecting many popular iOS apps that are widely used.

Finally, XcodeGhost will also use the clipboard functionality provided by iOS to temporarily store some data it needs. In fact, every time an infected app is launched, XcodeGhost will retrieve persistently stored data from the clipboard with a paste board named by app's bundle ID and a fixed string "UIPasteBoard", then store new data inside it. Although this behavior is not harmful to users, a slightly code change would allow the same technique to steal passwords from password management apps such as 1Password.

When people use apps like 1Password to manage their passwords in iOS, they often open 1Password, copy the stored password to system clipboard, then open the app they want to use and paste the password to the login window. At this moment, a malicious app can directly read the password from system clipboard. 1Password's main security design for this situation is that, the password stored in the clipboard will only stay there for a very short time. However, since the malware can read it when the app launches, the attack can be successful.

Attack in the Wild

Earlier today on a popular Chinese forum V2EX, [a user "realpg" mentioned his experience](#) when developing iOS apps using the malicious Xcode package. His write-up disclosed that XcodeGhost's attacker has used the malware to phish victims' iCloud passwords.

In the discussion, "realpg" said that when they were developing a very simple iOS app that had no Internet functionality and didn't use any iCloud APIs, the app would frequently display a dialog to ask the developer to input his iCloud password. They tested the app in their special testing iPhone without jailbreak. Then they tried to capture the network traffic and found the exactly the same C2 domain name used by XcodeGhost infected apps.

Based on "realpg"'s account of the events, we believe that stealing passwords or potentially exploiting vulnerabilities in iOS and in legitimate applications may be the true purpose of XcodeGhost.

Acknowledgement

Thanks to Luyi Xing from Indiana University for providing knowledge about password management tools. Also, many thanks to @Saic on Sina Weibo for identifying potential behaviors in XcodeGhost.
