

# Pawn Storm Update: iOS Espionage App Found

In our continued research on Operation Pawn Storm, we found one interesting poisoned pawn—spyware specifically designed for espionage on iOS devices. While spyware targeting Apple users is highly notable by itself, this particular spyware is also involved in a targeted attack.

## ***Background of Operation Pawn Storm***

**Operation Pawn Storm** is an active economic and political cyber-espionage operation that targets a wide range of entities, like the military, governments, defense industries, and the media.

The actors of Pawn Storm tend to first move a lot of pawns in the hopes they come close to their actual, high profile targets. When they finally successfully infect a high profile target, they might decide to move their next pawn forward: advanced espionage malware.

The iOS malware we found is among those advanced malware. We believe the iOS malware gets installed on already compromised systems, and it is very similar to next stage SEDNIT malware we have found for Microsoft Windows' systems.

We found two malicious iOS applications in Operation Pawn Storm. One is called *XAgent* (detected as IOS\_XAGENT.A) and the other one uses the name of a legitimate iOS game, *MadCap* (detected as IOS\_XAGENT.B). After analysis, we concluded that both are applications related to SEDNIT.

The obvious goal of the SEDNIT-related spyware is to steal personal data, record audio, make screenshots, and send them to a remote command-and-control (C&C) server. As of this publishing, the C&C server contacted by the iOS malware is live.

## ***Analysis of XAgent***

The XAgent app is fully functional malware. After being installed on iOS 7, the app's icon is hidden and it runs in the background immediately. When we try to terminate it by killing the process, it will restart almost immediately.

Installing the malware into an iOS 8 device yields different results. The icon is not hidden and it also cannot restart automatically. This suggests that the malware was designed prior to the release of iOS 8 last September 2014.

## ***Data Theft Capabilities***

The app is designed to collect all kind of information on an iOS device. It is able to perform the following routines:

- Collect text messages
- Get contact lists
- Get pictures
- Collect geo-location data
- Start voice recording
- Get a list of installed apps
- Get a list of processes
- Get the Wi-Fi status

```

+ [Reachability reachabilityForInternetConnection]
+ [Reachability reachabilityForLocalWiFi]
+ [Reachability reachabilityWithAddress:]
+ [Reachability reachabilityWithHostName:]
+ [SCRFTPRequest initialize]
+ [SCRFTPRequest requestWithURL:toCreateDirectory:]
+ [SCRFTPRequest requestWithURL:toDownloadFile:]
+ [SCRFTPRequest requestWithURL:toUploadFile:]
+ [SCRFTPRequest sharedRequestQueue]
- [PhotoLibrary .cxx_destruct]
- [PhotoLibrary allPhotosCollected:]
- [PhotoLibrary buildMsg:size:cmdId:]
- [PhotoLibrary dateImage]
- [PhotoLibrary getAllPicturePhotoLibrary]
- [PhotoLibrary getAllPictures]

- [Reachability connectionRequired]
- [Reachability currentReachabilityStatus]
- [Reachability dealloc]
- [Reachability localWiFiStatusForFlags:]
- [Reachability networkStatusForFlags:]
- [Reachability startNotifier]
- [Reachability stopNotifier]
- [RecordVoice .cxx_destruct]
- [RecordVoice StartRecord:]
- [RecordVoice dateString]
- [RecordVoice getRecordFile:file_path:]
- [RecordVoice record:]
- [RecordVoice recordCall]
- [RecordVoice startAudioSession]
- [RecordVoice stopRecordCall]

- [GeoLocation .cxx_destruct]
- [GeoLocation CurrentLocation]
- [GeoLocation currentLocation]
- [GeoLocation locationManager]
- [GeoLocation setCurrentLocation:]
- [GeoLocation setLocationManager:]

```

Figure 1. XAgent code structure

### C&C Communication

Besides collecting information from the iOS device, the app sends the information out via HTTP. It uses POST request to send messages, and GET request to receive commands.

### Formatted Log Messages

The malware's log messages are written in HTML and color coded, making it easier for human operators to read. Error messages tend to be in red, while others are in green as shown in the figure below.

```

<font size=4 color=green><pre>Command get info success</pre></font>
<font size=4 color=red><pre>Command start record voice unsuccess, record voice active!!!</pre></font>
<font size=4 color=green><pre>Command start record voice success</pre></font>
<font size=4 color=green><pre>Command get audio file success</pre></font>
<font size=4 color=red><pre>Command contact book got unsuccess</pre></font>
<font size=4 color=green><pre>Command contact book got success</pre></font>
<font size=4 color=red><pre>Command get current location unsuccess</pre></font>
<font size=4 color=green><pre>Command get current location success</pre></font>
<font size=4 color=red><pre>Command get installed app unsuccess</pre></font>
<font size=4 color=green><pre>Command get installed app success</pre></font>
<font size=4 color=red><pre>Command status wifi unsuccess</pre></font>
<font size=4 color=green><pre>Command status wifi success</pre></font>
<font size=4 color=green><pre>Command getting file list of directory is success</pre></font>
<font size=4 color=red><pre>Command get file unsuccess</pre></font>
<font size=4 color=red><pre>Get process list unsuccess</pre></font>
<font size=4 color=green><pre>Get process list success</pre></font>
<font size=4 color=red><pre>Get sms message unsuccess</pre></font>
<font size=4 color=green><pre>Get sms message success</pre></font>

```

Figure 2. Color-coded HTML log messages

### A Well-Designed Code Structure

We can see that the code structure of the malware is very organized. The malware looks carefully maintained and consistently updated.

```

[f] -[XAppDelegate .cxx_destruct]
[f] -[XAppDelegate activeAPP]
[f] -[XAppDelegate appBackgrounding:]
[f] -[XAppDelegate appForegrounding:]
[f] -[XAppDelegate application:didFinishLaunchingWithOptions:]
[f] -[XAppDelegate applicationDidBecomeActive:]
[f] -[XAppDelegate applicationDidEnterBackground:]
    [f] -[XA_HTTP_Channel .cxx_construct]
    [f] -[XA_HTTP_Channel .cxx_destruct]
    [f] -[XA_HTTP_Channel clear:]
    [f] -[XA_HTTP_Channel count]
    [f] -[XA_HTTP_Channel createCryptPacket]

```

Figure 3. XAgent code structure

The app uses the commands *watch*, *search*, *find*, *results*, *open*, and *close*.

__data:00032E18	00000008	C	watch/?
__data:00032E2C	00000009	C	search/?
__data:00032E40	00000007	C	find/?
__data:00032E54	0000000A	C	results/?
__data:00032E68	00000007	C	open/?
__data:00032E7C	00000009	C	search/?
__data:00032E90	00000008	C	close/?

Figure 4. List of base URIs

### Randomly Generated URI

The full uniform resource identifier (URI) for C&C HTTP requests is randomly generated, according to a template agreed upon with the C&C server. The base URI can be seen in Figure 4, and parameters are chosen from the list below and appended to the base URI.

	__data:00032EA8 00000006	C	text=
	__data:00032EB2 00000006	C	from=
	__data:00032EC6 00000005	C	ags=
	__data:00032EE4 00000006	C	btnG=
	__data:00032EEE 00000007	C	oprnd=
	__data:00032F02 00000005	C	utm=
	__data:00032F0C 00000009	C	channel=

Figure 5. List of parameters used with URIs

Here are corresponding implementations we got during our reversing:

```

ADD      R1, SP, #0xB38+var_7DC
MOV      R2, (__ZN8httpvarsL9URL_PATHE - 0x2277E) ; "watch/?"
ADD      R2, PC ; "watch/?"
MOVS     R3, 0
MOV      R9, 7
MOV      R12, (_objc_msgSend_ptr_0 - 0x22796) ; _objc_msgSend_ptr_0
ADD      R12, PC ; _objc_msgSend_ptr_0
LDR.W   R12, [R12] ; __imp__objc_msgSend
MOV      LR, (selRef_random_end_ - 0x227A4) ; selRef_random_end_
ADD      LR, PC ; selRef_random_end_

ADD.W   R3, R2, R2, LSL#2
MOV      R9, (__ZN8httpvarsL9URL_TYPESE - 0x22A3E) ; "text="
ADD      R9, PC ; "text="
ADD.W   R1, R9, R3, LSL#1
MOVS     R3, #8
STR.W   R3, [SP, #0xB38+var_48]
MOVS     R2, #0xA
BL      __ZN5Coder7setDataEPhj ; Coder::setData(uchar *,uint)
B       loc_22A4E

```

Figures 6 and 7. Code for URI generation

### Token Format and Encoding

The malware uses a token to identify which module is communicating. The token is Base64 encoded data, but padded with a 5-byte random prefix so that it looks like valid Base64 data. See the first line “ai=” part in the figure below.

```
GET /close/?ai=he1IAss1t\MTDSG1Z_E1f-
XrR&aq=fdI7B994eL_&aq=0M3AP1&oprnd=zE&text=vFXJ7qDHEC8&s9Y=mJf-3TPyYg HTTP/1.1
Host: ██████████
Connection: keep-alive
Accept-Encoding: gzip, deflate
User-Agent: XAgent/1.0 CFNetwork/672.1.14 Darwin/14.0.0
Accept-Language: zh-cn
Accept: */*

HTTP/1.1 200 OK
Date: Tue, 13 Jan 2015 09:05:08 GMT
Server: Apache/2.2.15 (CentOS)
Content-Length: 3
Connection: close
Content-Type: text/plain; charset=UTF-8

400
```

Figure 8. Client (XAgent) request

Reverse engineering also revealed additional communication functions.

```
f| Coder::base64UrlDecode(char *,uint,uint *)
f| Coder::createAlphabet(uchar)
f| Coder::b64Decode(char *,uint,uint *,char *)
f| Coder::base64UrlEncode(uchar *,uint,uint *)
f| Coder::b64Encode(uchar *,uint,uint *,char *)
f| CryptoContainer::cryptData(uchar *,uint,uint *)
f| CryptoContainer::random(uint,uint)
f| CryptoContainer::cryptRc4(uchar *,uint,uint)
f| CryptoContainer::decryptData(uchar *,uint,uint *)
f| -[XA_HTTP_Chanel getCryptoRawPacket]
f| -[XA_HTTP_Chanel post]
f| -[XA_HTTP_Chanel createCryptPacket]
f| -[XA_HTTP_Chanel createDecryptPacket:]
```

Figure 9. HTTP communication functions

```
f| Coder::base64UrlDecode(char *,uint,uint *)
f| Coder::createAlphabet(uchar)
f| Coder::b64Decode(char *,uint,uint *,char *)
f| Coder::base64UrlEncode(uchar *,uint,uint *)
f| Coder::b64Encode(uchar *,uint,uint *,char *)
f| CryptoContainer::cryptData(uchar *,uint,uint *)
f| CryptoContainer::random(uint,uint)
f| CryptoContainer::cryptRc4(uchar *,uint,uint)
f| CryptoContainer::decryptData(uchar *,uint,uint *)
f| -[XA_HTTP_Chanel getCryptoRawPacket]
f| -[XA_HTTP_Chanel post]
f| -[XA_HTTP_Chanel createCryptPacket]
f| -[XA_HTTP_Chanel createDecryptPacket:]
```

Figure 10. C2 server

## FTP Communication

The app is also able to upload files via FTP protocol.



Figure 11. FTP communication functions

### ***Analysis of “MadCap”***

“Madcap” is similar to the XAgent malware, but the former is focused on recording audio. “Madcap” can only be installed on jailbroken devices.

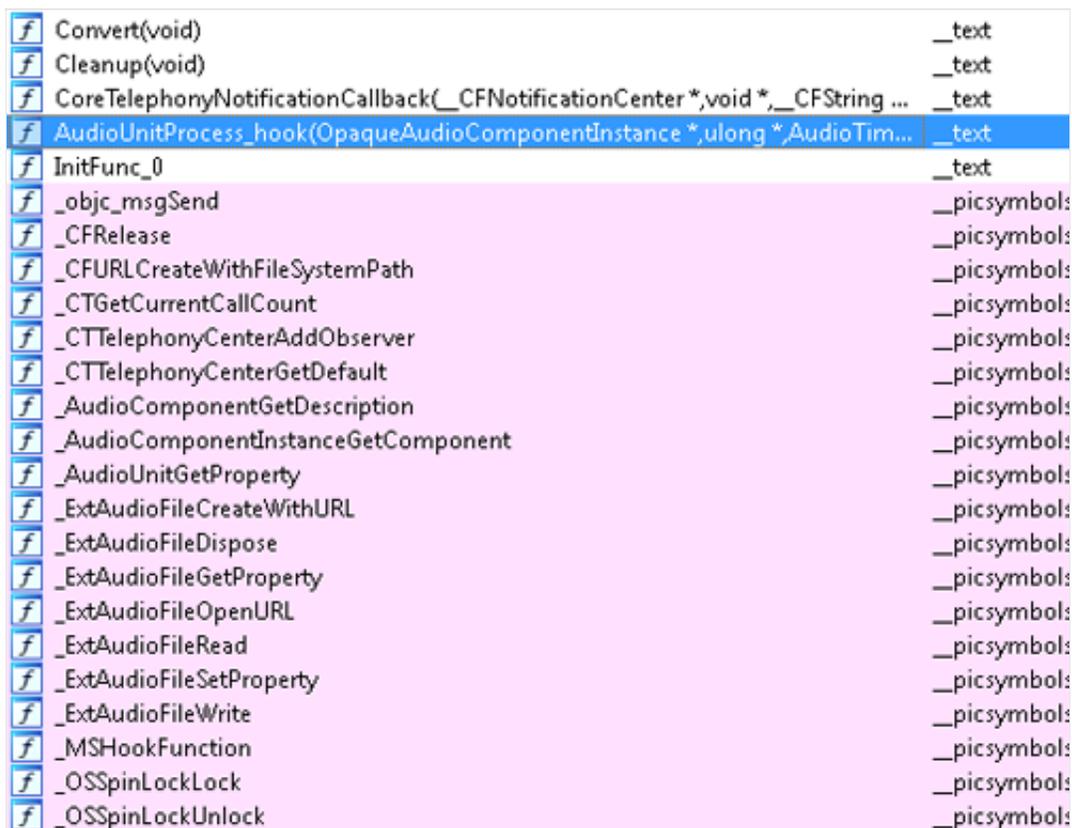
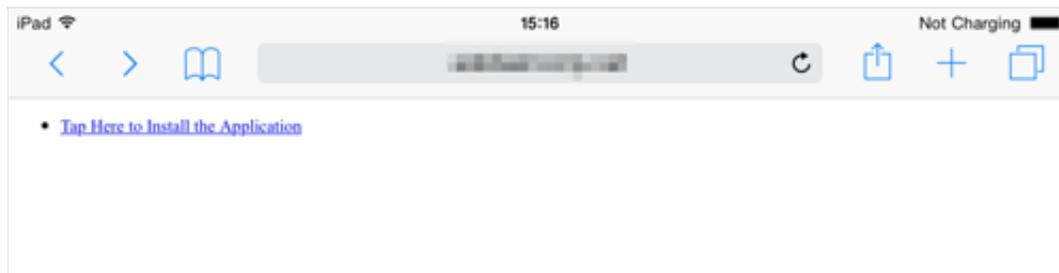


Figure 12. Code structure of Madcap

### ***Possible Infection Methods***

The exact methods of installing these malware is unknown. However, we do know that the iOS device doesn't have to be jailbroken per se. We have seen one instance wherein a lure involving XAgent simply says "Tap Here to Install the Application." The app uses Apple's ad hoc provisioning, which is a standard distribution method of Apple for iOS App developers. Through ad hoc provisioning, the malware can be installed simply by clicking on a link, such as in the picture below. The link will lead to <https://www.{BLOCKED}/adhoc/XAgent.plist>, a service that installs applications wirelessly.



*Figure 13. Site used in downloading XAgent*

There may be other methods of infection that are used to install this particular malware. One possible scenario is [infecting an iPhone](#) after connecting it to a compromised or infected Windows laptop via a USB cable.

*To learn more about this campaign, you may refer to our report, [Operation Pawn Storm Using Decoys to Evade Detection](#).*

The hashes of the related files are:

- 05298a48e4ca6d9778b32259c8ae74527be33815
- 176e92e7cfc0e57be83e901c36ba17b255ba0b1b
- 30e4decd68808cb607c2aba4aa69fb5fdb598c64

***Special thanks to Loucif Kharouni and Fernando Mercas for additional insights.***