Emotet Moves to 64 bit and Updates its Loader

By Oleg Boyarchuk, Jason Zhang, Stefano Ortolani

Published: 2022-05-16 · Archived: 2026-04-06 00:16:29 UTC

For the last three weeks, security researchers have been noticing Emotet migrating to new 64-bit modules https://twitter.com/Cryptolaemus1/status/1516261512372965383. While the change initially affected Epoch 4, by the second week of May both Epoch 4 and Epoch 5 had been fully migrated. Figure 1 shows one of the first waves leveraging 64-bit modules targeting, among others, one of our customers in the Healthcare vertical. At the time of writing, we can also confirm that both the DLL loader and stealing modules are now 64-bit binaries. While it is true that 32-bit operating systems have been on their way out for quite a while, it is still not clear why Emotet's authors decided to carry out this transition at this specific moment in time; the benefits are, however, quite clear from a software development perspective: debugging is easier, there is no intermediate emulation layer anymore (WOW64), and accessing native system directories does not require selectively disabling the emulation layer.

In this blog post, we analyze one of the latest samples and investigate whether the changes introduced by migrating to pure 64-bit binaries are only cosmetic or they are rather also functionally relevant.

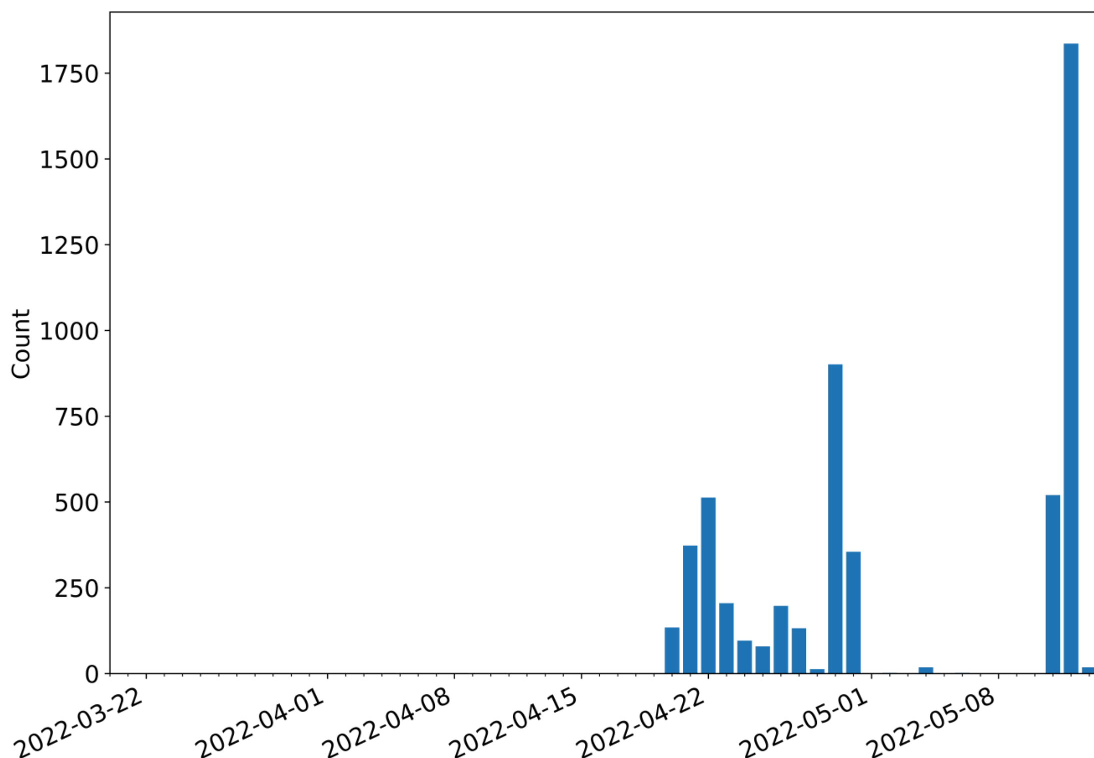

Figure 1: Timeline of Emotet DLL 64-bit payloads.

What Did Not Change

As we described in a previous blog post, the DLL payload is the key component of any Emotet execution chain:

- The payload is an obfuscated DLL; it is meant to be executed by rundll32.exe or regsvr32.dll, and it holds an encrypted DLL inside.
- The inner DLL is responsible for persistence and C2 communication.
- The inner DLL relies on functions imported from bcrypt.dll to encrypt and decrypt the network traffic.
- Network communication is still achieved via functions imported from wininet.dll.

Our initial assessment suggests that the recompilation to target the 64-bit architecture natively did not introduce any major changes. For instance, the logic handling encryption and network communication have been kept the same, including the way in which the malware accesses WinAPI functions. Figure 2 shows the code snippets tasked with dynamically retrieving the pointers to the WinAPI functions, extracted from both a 32-bit (above) and a 64-bit (below) sample; as one can see, the implemented logic is basically identical.

```
int __usercall FindProcAddress@<eax>(int a1@<edx>, int a2, int a3, int a4, int a5)
{
  int v5; // edi
  int v6; // eax

  v5 = a1;
  if ( !dword_6A3E4220[a4] )
  {
    v6 = sub_6A3D0028(124, a5);
    dword_6A3E4220[a4] = (int)sub_6A3CCD9B(425424, 650657, v5, 714860, v6);
  }
  return dword_6A3E4220[a4];
}

__int64 __fastcall FindProcAddress(__int64 a1, __int64 a2, unsigned int a3, int a4, unsigned int a5)
{
  __int64 v5; // rbx
  __int64 result; // rax
  int v8; // eax

  v5 = a3;
  result = qword_7FF86A8D7260[a3];
  if ( !result )
  {
    v8 = sub_7FF86A8CA95C(a1, a5);
    result = sub_7FF86A8B1BD4(954708, v8, a4, 102461, 244323, 819810);
    qword_7FF86A8D7260[v5] = result;
  }
  return result;
}
```

Figure 2: Function to retrieve pointers to WinAPI functions in a 32-bit sample e597f6439a01aad82e153e0de647f54ad82b58d3 (above) vs. the same function in a 64-bit sample C6fe1cf52c7f3299f07a1e1c05e19e2013330e4c (below).

This is also substantiated by looking at the API call sequence across 32-bit and 64-bit samples; both logics to encrypt the traffic and establish network communications use the exact same APIs, in the exact same order:

bcrypt!BCryptCreateHash

bcrypt!BCryptHashData

bcrypt!BCryptFinishHash

bcrypt!BCryptDestroyHash

kernel32!GetProcessHeap

kernel32!HeapFree

bcrypt!BCryptCloseAlgorithmProvider

kernel32!GetProcessHeap

ntdll!RtlAllocateHeap

ntdll!memcpy

ntdll!memcpy

bcrypt!BCryptEncrypt

kernel32!GetProcessHeap

ntdll!RtlAllocateHeap

bcrypt!BCryptEncrypt

…

wininet!InternetOpenW

kernel32!GetProcessHeap

kernel32!HeapFree

wininet!InternetConnectW

wininet!HttpOpenRequestW

kernel32!GetProcessHeap

kernel32!HeapFree

wininet!InternetSetOptionW

wininet!InternetQueryOptionW

wininet!InternetSetOptionW

wininet!HttpSendRequestW

Even the location where the configuration data is stored does not exhibit any update; the data blob at the very beginning of the .data section of a 64-bit payload is still the same encrypted C2 configuration (see Figure 3); we refer to our previous [blog post] for more details about how to decrypt the configuration.

```
.text:00007FFA1B6650CA
.text:00007FFA1B6650CA loc_7FFA1B6650CA:                  ; CODE XREF: sub_7FFA1B665028+78↑j
.text:00007FFA1B6650CA          mov     [rbp+57h+arg_10], 0A7F143h
.text:00007FFA1B6650D1          lea     r8, unk_7FFA1B667000
.text:00007FFA1B6650D8          lea     rcx, [rbp+57h+var_64]
.text:00007FFA1B6650DC          shr     [rbp+57h+arg_10], ; Alignment     : default
.text:00007FFA1B6650E0          shr     [rbp+57h+arg_10], ; ==========================================================
.text:00007FFA1B6650E3          xor     [rbp+57h+arg_10],
.text:00007FFA1B6650EA          mov     [rbp+57h+arg_8], 0; Segment type: Pure data
.text:00007FFA1B6650F1          add     [rbp+57h+arg_8], 0; Segment permissions: Read/Write
.text:00007FFA1B6650F8          or      [rbp+57h+arg_8], 2_data          segment para public 'DATA' use64
.text:00007FFA1B6650FF          shr     [rbp+57h+arg_8], 9                assume cs:_data
.text:00007FFA1B665103          xor     [rbp+57h+arg_8], 1                ;org 7FFA1B667000h
.text:00007FFA1B66510A          mov     dword ptr [rbp+57 unk_7FFA1B667000 db  70h ; p          ; DATA XREF: sub_7FFA1B665028+A9↑o
.text:00007FFA1B665111          add     dword ptr [rbp+57                db  8Ch
.text:00007FFA1B665118          add     dword ptr [rbp+57                db 0A8h
.text:00007FFA1B66511F          shl     dword ptr [rbp+57                db  3Ch ; <
.text:00007FFA1B665123          xor     dword ptr [rbp+57                db  88h
.text:00007FFA1B66512A          mov     eax, dword ptr [r                db  8Dh
.text:00007FFA1B66512D          mov     r9d, [rbp+57h+arg                db 0A8h
.text:00007FFA1B665131          mov     edx, [rbp+57h+arg                db  3Ch ; <
.text:00007FFA1B665134          mov     dword ptr [rsp+0E                db 0FAh
.text:00007FFA1B665138          call    DecryptConfig                    db  45h ; E
.text:00007FFA1B66513D          mov     ecx, [rbp+57h+var                db  26h ; &
.text:00007FFA1B665140          add     rcx, rax                         db  75h ; u
.text:00007FFA1B665143          mov     r13, rax                         db  6Fh ; o
.text:00007FFA1B665146          mov     r8, rax                          db  1Ch
.text:00007FFA1B665149          mov     [rbp+57h+var_60],                db 0A8h
.text:00007FFA1B66514D          mov     [rbp+57h+var_50],                db  3Dh ; =
.text:00007FFA1B665151          mov     eax, 92A79h                      db 0FAh
.text:00007FFA1B665156          jmp     loc_7FFA1B66506B
```

Figure 3: By looking at xrefs to the data blob from .data (c6fe1cf52c7f3299f07a1e1c05e19e2013330e4c) it is possible to locate the config decryption function.

What Changed

While the main logic did not change, some of the constants used by the DLL loader have been updated. To understand which constants are changed, and how they affect detection and analysis, one needs to look at the Emotet's update process.

One of the main functionalities of the payload DLL is to pull updates from a C2 server; these updates are also DLLs but featuring a custom entry point: while normal DLLs perform their initialization during the DLL_PROCESS_ATTACH call and free the resources during the DLL_PROCESS_DETACH call, the 32-bit DLLs distributed by the botnet as updates execute their initialization routine only when a custom value, fdwReason=16, is given as input (see Figure 4); furthermore, the loading routine requires custom data structures passed via the lpReserved pointer. Failure to comply with any of these requirements (by, for example, loading the DLL using rundll32.exe) will either crash the sample or make it skip the initialization routine.

```
signed int __stdcall DllEntryPoint(int a1, int fdwReason, int a3)
{
  if ( fdwReason == 16 )
  {
    sub_1006994C(807095, 860132);
  }
  else if ( fdwReason == 32 )
  {
    sub_1006A899(1000503, 888472);
  }
  return 1;
}
```

Figure 4: Entry point of a 32-bit update (7d3f067f4b135a4a4d4b717bc7f7f4dd8e3a7ff8).

The updated 64-bit DLLs retain this specific loading mechanism, but they now correctly initialize only when a different value, i.e., 100, is given as input (see Figure 5); they also remove the branch called when fdwReason=32, which was originally leading to an empty function. While these little changes can still be considered merely cosmetic, they are able to break any custom loader built by security analysts to dynamically analyze these update binaries; hopefully this is not the start of yet another cat-and-mouse game between security analysts and Emotet developers.

```
BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  if ( fdwReason == 100 )
    sub_18006DBC4(lpReserved, 1047766i64, 678436i64);
  return 1;
}
```

Figure 5: Entry point of a 64-bit update (3c729151d9d2d326a4a3772ee18a1c0ca5db55ce).

Conclusions

As Emotet is a moving target, security researchers have no choice but to keep investigating how the tactics, techniques, and procedures (TTPs) used by Emotet's authors are evolving over time, and how to effectively detect this threat.

The move to 64-bit binaries seems to prepare the groundwork for further modifications yet to come, and while the differences are not functionally relevant, they already pose potential challenges to dynamic analysis systems that analyze payloads in sandboxing environments. Analyzing a threat by focusing on its full attack chain (as done by modern NDR solutions) is the only way to keep up with an ever-evolving threat landscape.

Appendix: IoCs

Indicators of compromise identified from this report can be found on VMware TAU's GitHub IoCs repository.

Source: https://blogs.vmware.com/security/2022/05/emotet-moves-to-64-bit-and-updates-its-loader.html