Dissecting Emotet – Part 1

By Deutsche Telekom AG

Published: 2020-02-03 · Archived: 2026-04-02 10:46:55 UTC

I bet you've heard of Emotet, a modular botnet that is active at least since 2014. Since many years it is considered one of the most costly and destructive malware family affecting the public and private sectors as well as individuals. For those who do not know the background to Emotet yet: It started off as a banking Trojan but changed its modus operandi in 2017 to the distribution of third party malicious payloads. Recently, researchers have observed Emotet to deliver TrickBot and the ransomware Ryuk as secondary payloads.

> Thomas' path led from research to business. He regularly shares his findings in this blog.
>

This botnet comprises its own spreading mechanism. Most infections are achieved by sending out spam with links to malicious documents. If the target actively interacts with such a document, Emotet's loader is installed. The loader is very limited in its capabilities from my point of view. Its main purpose is to persist Emotet on a target system, contact its Command & Control Servers, which steer the attack (C2 servers), and download as well as execute further payloads. These payloads are either Emotet modules or further secondary payloads like TrickBot. The interesting functionality of Emotet is implemented in its modules such as spamming, lateral movement, or credential theft. This modular structure makes Emotet very flexible and effective.

Emotet's modules do not persist on a target system. This has several consequences. First, it is forensically hard to tell what kind of data an Emotet infection stole. Since modules are distributed to a target system based on an unknown selection algorithm, the number of modules that are deployed on a target system varies. Second, there tends to be a delay of up to a couple of weeks in the emergence of such modules on public online scan engines like VirusTotal. This results in a short period of time where these modules fly under the radar.

In this multi part blog post, I will now dive deep into Emotet's modules internals. In the first part, I will give an overview of the currently distributed modules and the way how some of these modules encrypt their payloads. Then I will describe how and why some of these modules are constantly changing their hash values and I will discuss the evolution of Emotet's modules in the last couple of months.

Emotet - from the big picture to the detail

The Emotet loader downloads modules from its C2 server and executes them. These modules are implemented as Windows libraries (PE DLLs). They do not export any functions besides a DLL entry point. They may directly import additional functions from Windows libraries like 'kernel32.dll'. The Emotet loader executes a module in a new thread within its process space. In order to do so, it resolves any imported functions and applies relocations. Subsequently, it starts a new thread at the DLL's entry point. I found that modules work independently of the main module and hence they are easy to debug in isolation.

The obfuscation techniques of Emotet's modules are equal to its loader: they dynamically resolve most of the Windows API addresses via API hashing and they utilize a simple string encryption. Cafe Babe described Emotet's API hashing and string encryption in detail last year, which has not significantly changed since then. Note that Emotet's modules do not come in packed form. However, many of them are just wrappers around another payload. For instance, there is a wrapper module around NirSoft's WebBrowserPassView, which is a recovery tool for passwords stored in web browsers. This module decrypts WebBrowserPassView first and then it starts it in its own process.

I observed that some of these wrappers carry a 32 bit and a 64 bit payload. Depending on the system's architecture, the corresponding module gets decrypted and executed. The payloads are encrypted using a simple eXclusive OR cipher (XOR) with a hard-coded key, which I will explain later on.

Table 1 lists the payloads that I observed recently while tracking Emotet's infrastructure. Our findings are congruent with other public sources like Lucy Nagy's VirusBulletin 2019 paper.

Module	Is Wrapper Module?	Representative Hash
spammer	No	54be4115e54b6dcb2986ff953b271202de0b3c64b4b18714f49e0febddba3c0c
administrative hare spreader	No	f8dd847ab1565aa460875c782f44a003a5b2c20b0e76a6672cfe3cd952a38727
network password bruteforcer	No	d714eca8a485b86cfa40dacfdedcd164877bf9d0e0d704ea325718b14498ba02
WebBrowser-PassView	Yes	7f11dabe46bf0af8973ce849194a587bd0ba1452e165faf028983f85b2b624c2
MailPassView	Yes	400b411a9bffd687c5e74f51d43b7dc92cdb8d5ca9f674456b75a5d37587d342
Outlook contact harvester	Yes	fb541624735a03a1a72aebf80847403aa68d7278f6fea101d2ea9afc568b3b46

Outlook mail harvester	Yes	404652e82de15d9c6436d49cd9a2b46bc8b3a66f0a6530574e50ae165a5619e1
C2 traffic proxy	No	902bac124f2926e6a345f393f292136a02b3cf523f9c156f40223a0343f0869c

These modules keep the Emotet machinery running. The first three modules (spammer, administrative share spreader, and network password bruteforcer) spread Emotet to other machines. The spammer module carries out the spreading via email on a global scale. The administrative share spreader and the network password bruteforcer help to spread Emotet laterally within an infected organization. The administrative share spreader tries to spread Emotet to network shares that are accessible to the local administrator. The network password bruteforcer tries to brute force accounts of local network resources and subsequently execute the Emotet loader. Researchers observed this module to use a list of 10.000 passwords earlier this year. However, I found out that the latest known version from 2019-04-24 17:53:50 utilizes a brute force dictionary similar to the one reported to be used by QuackBot.

The modules for credential harvesting (WebBrowserPassView and MailPassView) allow for monitarization of stolen account data. Furthermore, they feed the spam module with new mail accounts. The Outlook contact harvester finds fresh target email addresses to be used by the spammer module. The Outlook mail harvester is a targeted way of spreading Emotet by highjacking email threads. Finally, the C2 traffic proxy module ensures resilience of the botnet. It accepts C2 traffic of other infected machines and sends it to the C2 server.

Wrapper Module Payload Decryption

Earlier I stated that there are some modules that are wrappers around another payload. For instance, there is a wrapper that contains the Nirsoft tool WebBrowserPassView as payload. The wrapper decrypts this payload and injects it into another process in order to harvest credentials stored by local web browsers.

The payload decryption algorithm is actually very similar to the string decryption algorithm that Cafe Babe described. In a nutshell, the decrypted payload is stored in a binary blob. Logically, this blob consists of four byte blocks. The blocks are encrypted with an eXclusive OR cipher (XOR). The XOR key is hardcoded in the binary. The first block contains the payload size. Therefore, this block must be decrypted first. Afterwards, the following blocks can be sequentially decrypted.

A major difference between the payload decryption and string decryption as described by Cafe Babe is that the payload decryption algorithm utilizes Intel's Streaming SIMD Extensions (SSE) to decrypt the blocks. By doing so, Emotet can decrypt four blocks at a time, i.e. 128 bits or 16 bytes. First, it loads the key into the XMM register 'xmm1'. The key repeats the same DWORD four times.

Next, it repeatedly utilizes the instruction 'pxor' to decrypt four blocks at a time until there are less than four blocks left.

Emotet can decrypt four blocks at a time. © Deutsche Telekom AG

It decrypts these remainder blocks with a sequential xor. Whether the usage of SSE is done due to emulation evasion or due to optimization considerations is unclear. However, I tend to believe that this should yield performance improvement.

The following algorithm implements the payload decryption as pseudo code. Note that I do not need to implement it using SSE because decrypting the blocks sequentially or in parallel is semantically equivalent.

That's how the payload decryption is implemented as pseudo code. © Deutsche Telekom AG

Some wrapper modules like the WebBrowserPassView wrapper contain only one payload. But there are some wrapper modules that contain two payloads: a 32 bit and 64 bit payload. Depending on the target system's architecture the corresponding payload gets decrypted. An example for such a wrapper module is the Outlook mail harvester wrapper **(**ebb9f0920f08ffd6d7a05da35ecf57a13d15fe9c79f1b39d48a53099794f4274**)**. The first payload is the 32 bit version of the Outlook mail harvester **(**404652e82de15d9c6436d49cd9a2b46bc8b3a66f0a6530574e50ae165a5619e1**)** and the second payload is the 64 bit version of the Outlook mail harvester **(**b074d13f218633803e419f4e72d134a129b15d5c21a9474d2f5665a3be194c9c**)**.

Conclusion

In this first blog post, I have discussed the heart of Emotet: its modular structure and its modules. I have given an overview of the eight modules that are currently distributed. These modules fall into three categories: spreading (spammer, password bruteforcer, network spreader), data exfiltration (outlook contact harvester, outlook mail harvester, wrapper WebBrowserPassView, wrapper MailPassView), and botnet resilience (C2 traffic proxy).

The modules are implemented as PE DLLs and they are executed first in a new thread within the same process space as the Emotet loader. Some modules are just wrapper around another payload like Nirsoft's MailPassView. These wrappers may inject their payload into another process in order to execute them. Emotet's modules live only in memory and they are not persisted on a target system. Consequently, this delays their appearance on public platforms like VirusTotal by up to a couple of weeks.

From a botnet author's point of view, this modular structure makes sense: it makes its analysis harder, the botnet more flexible and testable, and in the end, it also increases its efficiency. In our next blog post, I will investigate the rehashing of modules and the evolution of Emotet's modules within the last couple of months.
