Dissecting Emotet - Part 2

telekom.com/en/blog/group/article/cybersecurity-dissecting-emotet-part-two-596128

Blog.Telekom

03-06-2020
Thomas Barabosch

2 Comments

- Share Share
  Two clicks for more data privacy: click here to activate the button and send your recommendation. Data will be transfered as soon as the activation occurs.

- Print
- Read out

Ready to follow me further into the details of Emotet? That's right, it's about one of the biggest malware threats in the world. Emotet is mainly spread through spam campaigns. In the first part of this blog post I introduced the modular structure and individual elements in more detail. It makes sense to read this classification first and then continue at this point.

Emotet became world famous in 2019 and is still very active

First of all, I would like to show this time which tricks this malware uses to protect itself from being detected via so-called hash values. And before we finally turn to the evolution of the modules, I will talk about the consequences for us as defenders.

Ready? Let's do it!

"Rehashing" of Modules

Even though there are only a couple of unique modules, there are many versions with different hashes. This makes a hash-based detection of them not very efficient. In order to detect them, we need the latest version of a module. However, since these modules are only memory resident, it may take some time for them to get to AV companies or to public platforms like VirusTotal.

Let us have a look at the "rehashing" feature of Emotet's Command and Control (C2) servers. There are two classes of modules: those that change their hash value every couple of hours and those that do not change their hash value for months. So, the interesting question is: why does this happen?

At first, I consider the modules that continuously rehash. When byte comparing the outlook mail harvester modules that I fetched from Epoch 3 on 2019-11-04 and 2019-11-05, then it becomes clear that only a handful of bytes are different.

The area, where those two binaries differ, holds the set of C2 addresses. Only three of the four addresses are different. The first address (96 4E 3F B2 90 1F) stayed the same. Furthermore, both binaries comprise the same compilation timestamp (2019-10-15 04:31:32). Replacing the C2 addresses does not require any recompilation. It is probable that the C2 server does this fully automatically by directly writing to a fixed file offset (e.g. in this case 0xD300). From the operator’s point of view, the good thing is that these modules continuously rehash so that hash-only based solutions fail to detect them.

Now I regard two examples of the second case: the modules that do not change their hash values in months. I was able to obtain the administrative shares spreader module by all three epochs on several consecutive days (2019-11-05 - 2019-11-08). The modules always had the same hash (f8dd847ab1565aa460875c782f44a003a5b2c20b0e76a6672cfe3cd952a38727) and they all had a creation timestamp of 2019-03-31 10:59:43, which I believe to be correct. Its first submission to VirusTotal was on 2019-05-14 07:35:47. Under the assumption that the timestamps are correct then this would be six weeks after compilation and after their subsequent distribution within the botnet. I observed the same for the network password bruteforcer (d714eca8a485b86cfa40dacfdedcd164877bf9d0e0d704ea325718b14498ba02): its creation time is 2019-04-24 17:53:50 and its first submission to VirusTotal was on 2019-05-17 19:44:25.
The reason that those modules do not rehash is that they do not depend on fresh C2 addresses. They just spread the Emotet loader laterally. Since the loader has a set of C2 addresses, these modules can stay as they are and the Emotet operators do not have to bother to change these modules. For us as defenders this means that we can detect them either with Yara signatures or by using fuzzy hashes that are not affected by a couple of bit flips.

Evolution of Modules

In the previous section, I stated that the timestamps found in the modules' PE headers are reasonable. It is very likely that the Emotet operators did not tamper with them. Therefore, I utilized them to cluster the modules.
D00rt published a corpus of Emotet modules along with his blog post on Emotet's network protocol. Even though the corpus was collected within a limited time frame, i.e. around one month, there are more than 600 DLLs included. This is a wonderful opportunity to dig a little bit deeper in the recent evolution of Emotet's modules.

I clustered the over 600 DLLs of D00rt's corpus by timestamp and got 15 clusters. We can see that there are five modules that the Emotet operators compiled on 2019-03-31. Another batch of modules was compiled in April 2019. Finally, there are some modules that were compiled in October 2019. The following table summarizes the clusters. It holds the cluster's timestamp, the module that I associate with this cluster, the number of modules in this cluster, and a cluster representative for future research.

Timestamp	Module	Cluster Size	Cluster Representative
2019-03-31 10:48:35	wrapper WebBrowserPassView	102	1334b5c6025151c92abaa08adff7a0f04bf6e5d9a2b04d55e6c73bbe8a6a5983
2019-03-31 10:48:40	C2 traffic proxy	102	d45eb10428eb2b262a6bedd275e070618f36288dc8a872445456834e6c938d47
2019-03-31 10:48:52	wrapper MailPassView	102	4a23a1a24421b50570d840e1aade7d6304674707d81155be0818f8ac2f8bef1e
2019-03-31 10:48:55	Outlook contact harvester	117	49bce46b7931a52c655f9d9ba83ce47be6f07263b8053bb009f9ea6a7216ca91

2019-03-31 10:49:11	Outlook mail harvester	117	e9a077d388e98316af074d109e8d8f93cd07fe67073a585027fef0d640eff1f6
2019-03-31 10:59:43	administrative share spreader	30	f8dd847ab1565aa460875c782f44a003a5b2c20b0e76a6672cfe3cd952a38727
2019-04-08 16:38:38	spammer	23	b7ff07953428feb7d629b407158ce1fa896169ffd108d2a7a6c45e541167d4eb
2019-04-24 17:53:50	network password bruteforcer	39	d714eca8a485b86cfa40dacfdedcd164877bf9d0e0d704ea325718b14498ba02
2019-04-24 18:17:09	spammer	23	77c04e8a66efe29b56e9ef7c615776c0559872ca99c3db7fc80ef5a540c43b0a
2019-10-02 06:38:18	spammer	8	cbebc09364704b9078f6463eed723c31c384920f5f65ff1caf702f136d2ea93a
2019-10-04 23:19:53	Outlook contact harvester	15	918e4e36ced6d131ebd30fbcc2206823d14b702f448780afcbd06d60aae375f8
2019-10-04 23:20:39	Outlook mail harvester	15	3706c6ce387f803517464754bc6e04ddd3711401b07535cc9ca25008f321ff36
2019-10-14 20:45:29	spammer	2	54be4115e54b6dcb2986ff953b271202de0b3c64b4b18714f49e0febddba3c0c
2019-10-15 04:31:43	Outlook contact harvester	9	2f360761d5bc5505fa4ebf80281ba724e8027b923300aeb6da2799d21035720c
2019-10-15 04:31:32	Outlook mail harvester	9	5638f3f14cb916ac6f3debf1838822f789fbd760ea63981d7e3fb454b8afb95e

There are some interesting points to note. First, the Emotet operators compiled at least six modules on 2019-03-31. Since they were all compiled within two minutes, I assume that they belong to one (Visual Studio) solution, which got built on this date.

Second, there was some activity in April, when two versions of the spammer module and one version of the network password bruteforcer got compiled. In the case of the network password bruteforcer I assume that it got recompiled because the bruteforce dictionary was exchanged. It seems that there were no modules compiled in Summer 2019, which coincides with the reports on Emotet's summer break from June until September 2019.

Third, tracking these modules is (still) easy given several facts that I showed in this blog post: the timestamps seem to be reliable, there is no real packing of the modules, and only a small number of bytes of each module may change due to binary patching. Therefore, it is enough to match new modules against the fuzzy hash (e.g. ssdeep) of a cluster representative.

Conclusion

This blog post concludes my two-part series on the internals of Emotet's modules. I have described the "rehashing" of some of Emotet's modules, which is just a byproduct of binary patching new C2 addresses into already compiled modules. Next, I have analyzed D00rt's corpus of Emotet modules by clustering them by their timestamp. This has yielded only a couple of clusters and gives a good insight in the development of Emotet modules within the last months. Furthermore, I have discussed the possibility to detect modules by utilizing fuzzy hashes.