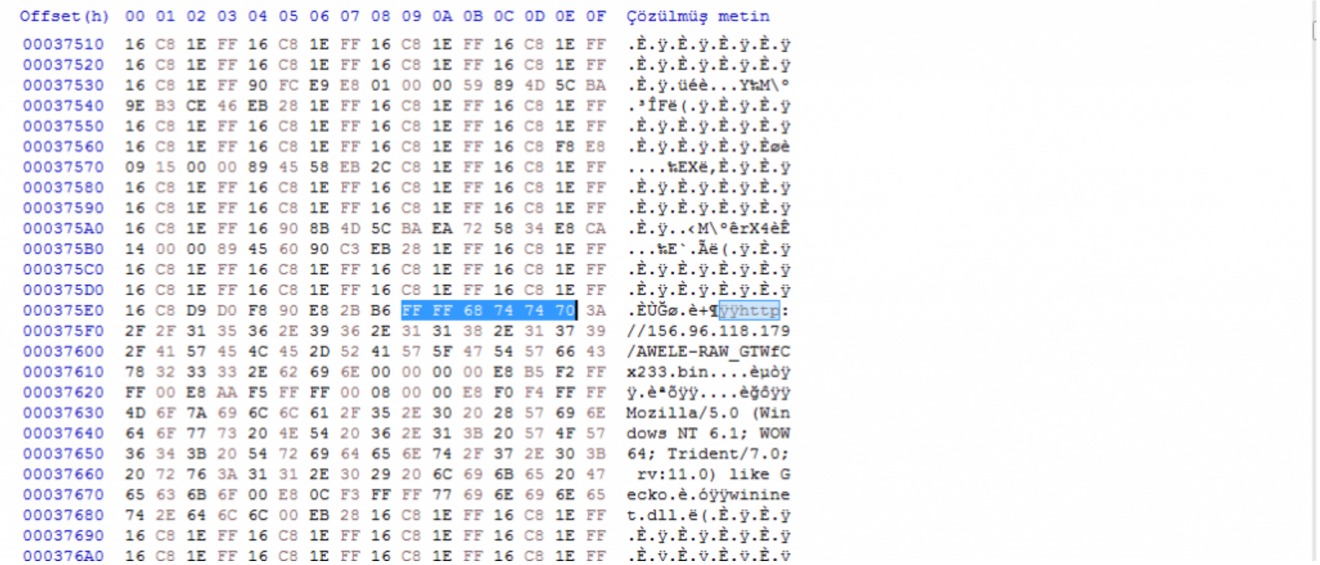
Malware Config Extraction Diaries #1 - GuLoader

malwation.com/malware-config-extraction-diaries-1-guloader/

malwation 08/09/2020



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Çözülmüş metin

00037510  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037520  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037530  16 C8 1E FF 90 FC E9 E8 01 00 00 59 89 4D 5C BA  .È.ÿ.üéè...Y‰M\º
00037540  9E B3 CE 46 EB 28 1E FF 16 C8 1E FF 16 C8 1E FF  .³ÎFë(.ÿ.È.ÿ.È.ÿ
00037550  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037560  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 F8 E8  .È.ÿ.È.ÿ.È.ÿ.Èøè
00037570  09 15 00 00 89 45 58 EB 2C C8 1E FF 16 C8 1E FF  ....‰EXë,È.ÿ.È.ÿ
00037580  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037590  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000375A0  16 C8 1E FF 16 90 8B 4D 5C BA EA 72 58 34 E8 CA  .È.ÿ..‹M\ºêrX4èÊ
000375B0  14 00 00 89 45 60 90 C3 EB 28 1E FF 16 C8 1E FF  ...‰E`.Ãë(.ÿ.È.ÿ
000375C0  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000375D0  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000375E0  16 C8 D9 D0 F8 90 E8 2B B6 FF FF 68 74 74 70 3A  .ÈÙĞø.è+¶ÿÿhttp:
000375F0  2F 2F 31 35 36 2E 39 36 2E 31 31 38 2E 31 37 39  //156.96.118.179
00037600  2F 41 57 45 4C 45 2D 52 41 57 5F 47 54 57 66 43  /AWELE-RAW_GTWfC
00037610  78 32 33 33 2E 62 69 6E 00 00 00 00 E8 B5 F2 FF  x233.bin....èµòÿ
00037620  FF 00 E8 AA F5 FF FF 00 08 00 00 E8 F0 F4 FF FF  ÿ.èªõÿÿ....èğôÿÿ
00037630  4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 57 69 6E  Mozilla/5.0 (Win
00037640  64 6F 77 73 20 4E 54 20 36 2E 31 3B 20 57 4F 57  dows NT 6.1; WOW
00037650  36 34 3B 20 54 72 69 64 65 6E 74 2F 37 2E 30 3B  64; Trident/7.0;
00037660  20 72 76 3A 31 31 2E 30 29 20 6C 69 6B 65 20 47   rv:11.0) like G
00037670  65 63 6B 6F 00 E8 0C F3 FF FF 77 69 6E 69 6E 65  ecko.è.óÿÿwinine
00037680  74 2E 64 6C 6C 00 EB 28 16 C8 1E FF 16 C8 1E FF  t.dll.ë(.È.ÿ.È.ÿ
00037690  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000376A0  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
```

Malware Config Extraction Diaries #2 – njRAT

03/10/2020

The activities of malware are increasing day by day. There are security solutions such as EDR, anti-virus, anti-malware and sandbox to prevent the activities of malicious software. However, the success rate of sandboxes, one of the most effective malware analysis products, is increasing day by day.

Malwation AIMA extract the configurations of malware families with new updates as well as the extra features it offers, and these configurations are critically important IOCs. Today, we can tell you that AIMA has stably extracted configurations of dozens of malware families. And we show how to extract the configurations of the GuLoader malware among them by writing script in Python and we present it to the open source world.

What is the GuLoader Family?

GuLoader (also known as CloudEye) is a Loader type malware written in the Visual Basic language. It downloads and runs RAT and Stealer type malwares such as AgentTesla, NetWire, Formbook from the remote server to the victim's system. Malwares that are downloaded and run from the remote server usually located on Google Drive and OneDrive.

Sophisticated malwares that continue to operate today often resort to many obfuscate and packaging processes in order to avoid security products and complicate the analysis process of malware analysts. As such, we can do the configuration extraction process from the healthiest memory dump.

First Part: Robust Analysis and Detection

If you want to extract configurations of a malware family, the most important thing to do is to continue the analysis stage very well and dump memory on several instances of the malware family that have identical versions. If you work on different versions, the scripts you have written will only be working on the sample you are analyzing, not with the corresponding version of the malware family, which is not a scenario we want.

After obtaining several different samples of the same malware family with the same version, we perform the analysis steps for each. We take note of the configuration data.

As a result of the analysis, the configuration data that can be extracted from this version of the GuLoader family are as follows:

- Remote server where the malicious application is downloaded,
- User-Agent of the request to the remote server,
- Registry path to provide persistence,
- Value and key setting in the registry,
- Dropped malicioud file path and name.

Part Two: Memory Dump

After all the valuable information described above, we are dumping all the malware samples. At this point, asynchronous memory dumps give healthier results instead of synchronous memory dumps. Since the processes on the memory progress very quickly, you may experience data loss depending on time, so it is necessary to dump asynchronously. AIMA's built-in advanced memory dump engine does our job and we get our memory dump in a healthy way.

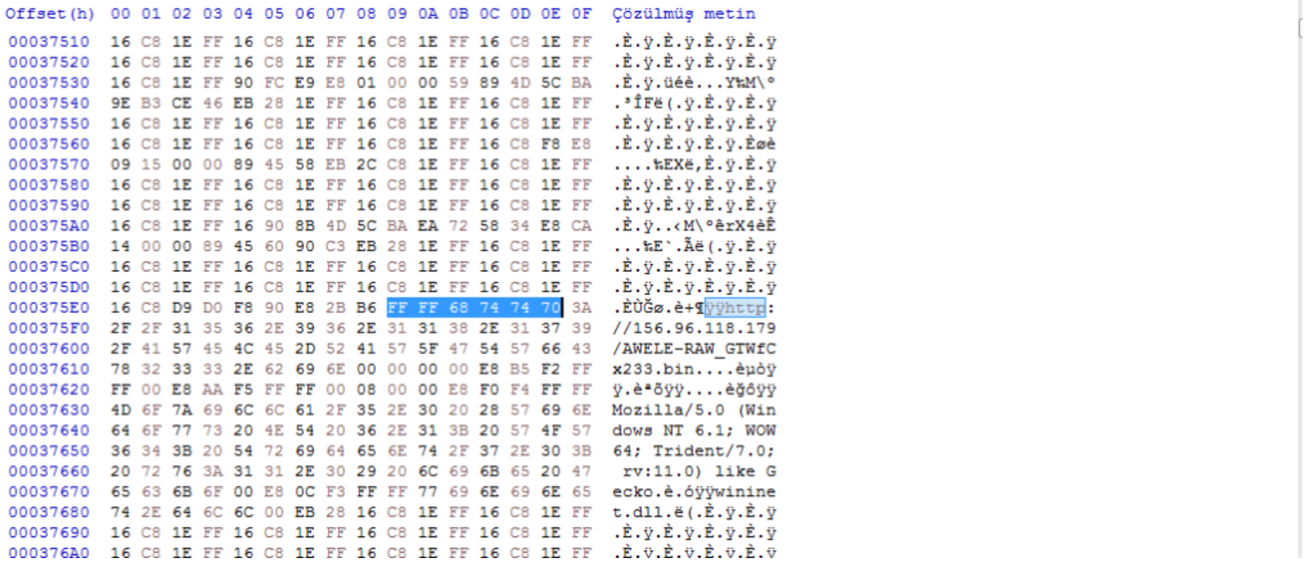
We reached certain configurations as a result of our previous analysis. Now we're drawing our road map.

1. First, detect the configuration items on the memory dump.
2. Compare the detected configuration items for each sample.
3. Find a specific pattern on memory dump for all malware samples.

Our roadmap is as shown above. We first determine the configuration items from the memory dump. Remember! Data must always be dumped into memory.

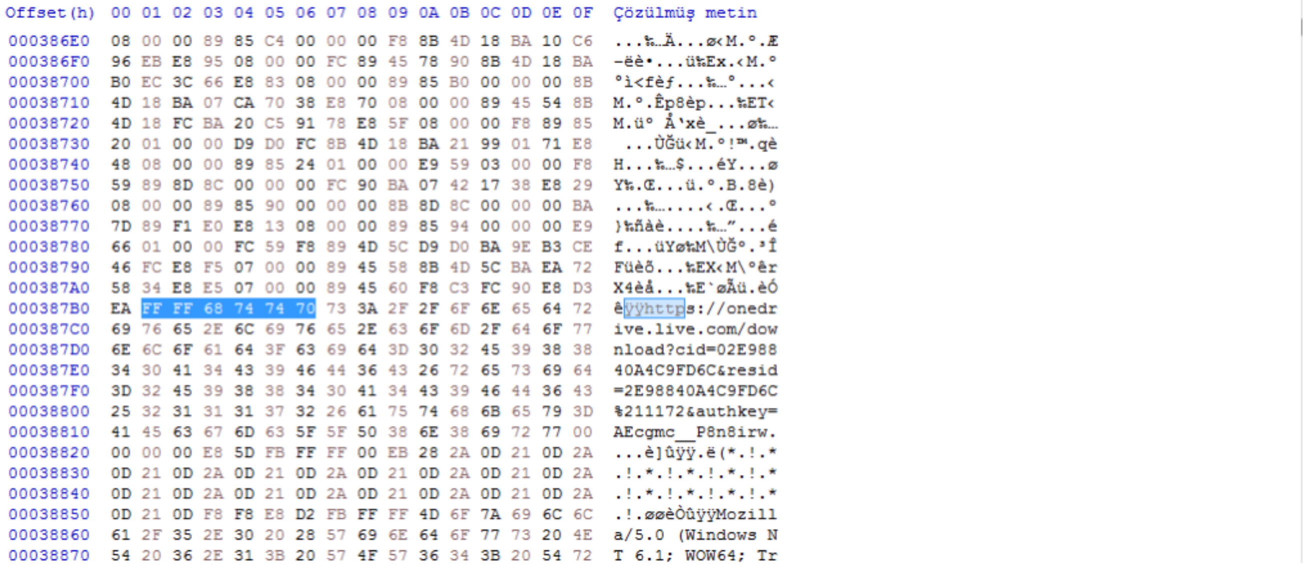
Part Three: Detecting Configurations in Memory Dump

As can be seen in the images below, we have identified the remote server addresses from which two different samples from the GuLoader family with the same version will download. When several different examples were examined, it was understood that the "0xFF 0xFF 0x68 0x74 x74 0x70" pattern could be used in the relevant version of GuLoader.



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Çözülmüş metin

00037510  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037520  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037530  16 C8 1E FF 90 FC E9 E8 01 00 00 59 89 4D 5C BA  .È.ÿ.üéè...Y‰M\°
00037540  9E B3 CE 46 EB 28 1E FF 16 C8 1E FF 16 C8 1E FF  .³ÎFë(.ÿ.È.ÿ.È.ÿ
00037550  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037560  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 F8 E8  .È.ÿ.È.ÿ.È.ÿ.Èøè
00037570  09 15 00 00 89 45 58 EB 2C C8 1E FF 16 C8 1E FF  ....‰EXë,È.ÿ.È.ÿ
00037580  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037590  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000375A0  16 C8 1E FF 16 90 8B 4D 5C BA EA 72 58 34 E8 CA  .È.ÿ..‹M\°êrX4èÊ
000375B0  14 00 00 89 45 60 90 C3 EB 28 1E FF 16 C8 1E FF  ...‰E`.Ãë(.ÿ.È.ÿ
000375C0  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000375D0  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000375E0  16 C8 D9 D0 F8 90 E8 2B B6 FF FF 68 74 74 70 3A  .ÈÙĞø.è+¶ÿÿhttp:
000375F0  2F 2F 31 35 36 2E 39 36 2E 31 31 38 2E 31 37 39  //156.96.118.179
00037600  2F 41 57 45 4C 45 2D 52 41 57 5F 47 54 57 66 43  /AWELE-RAW_GTWfC
00037610  78 32 33 33 2E 62 69 6E 00 00 00 00 E8 B5 F2 FF  x233.bin....èµòÿ
00037620  FF 00 E8 AA F5 FF FF 00 08 00 00 E8 F0 F4 FF FF  ÿ.èªõÿÿ....èğôÿÿ
00037630  4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 57 69 6E  Mozilla/5.0 (Win
00037640  64 6F 77 73 20 4E 54 20 36 2E 31 3B 20 57 4F 57  dows NT 6.1; WOW
00037650  36 34 3B 20 54 72 69 64 65 6E 74 2F 37 2E 30 3B  64; Trident/7.0;
00037660  20 72 76 3A 31 31 2E 30 29 20 6C 69 6B 65 20 47   rv:11.0) like G
00037670  65 63 6B 6F 00 E8 0C F3 FF FF 77 69 6E 69 6E 65  ecko.è.óÿÿwinine
00037680  74 2E 64 6C 6C 00 EB 28 16 C8 1E FF 16 C8 1E FF  t.dll.ë(.È.ÿ.È.ÿ
00037690  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000376A0  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
```

Figure 1: Malicious Sample 1



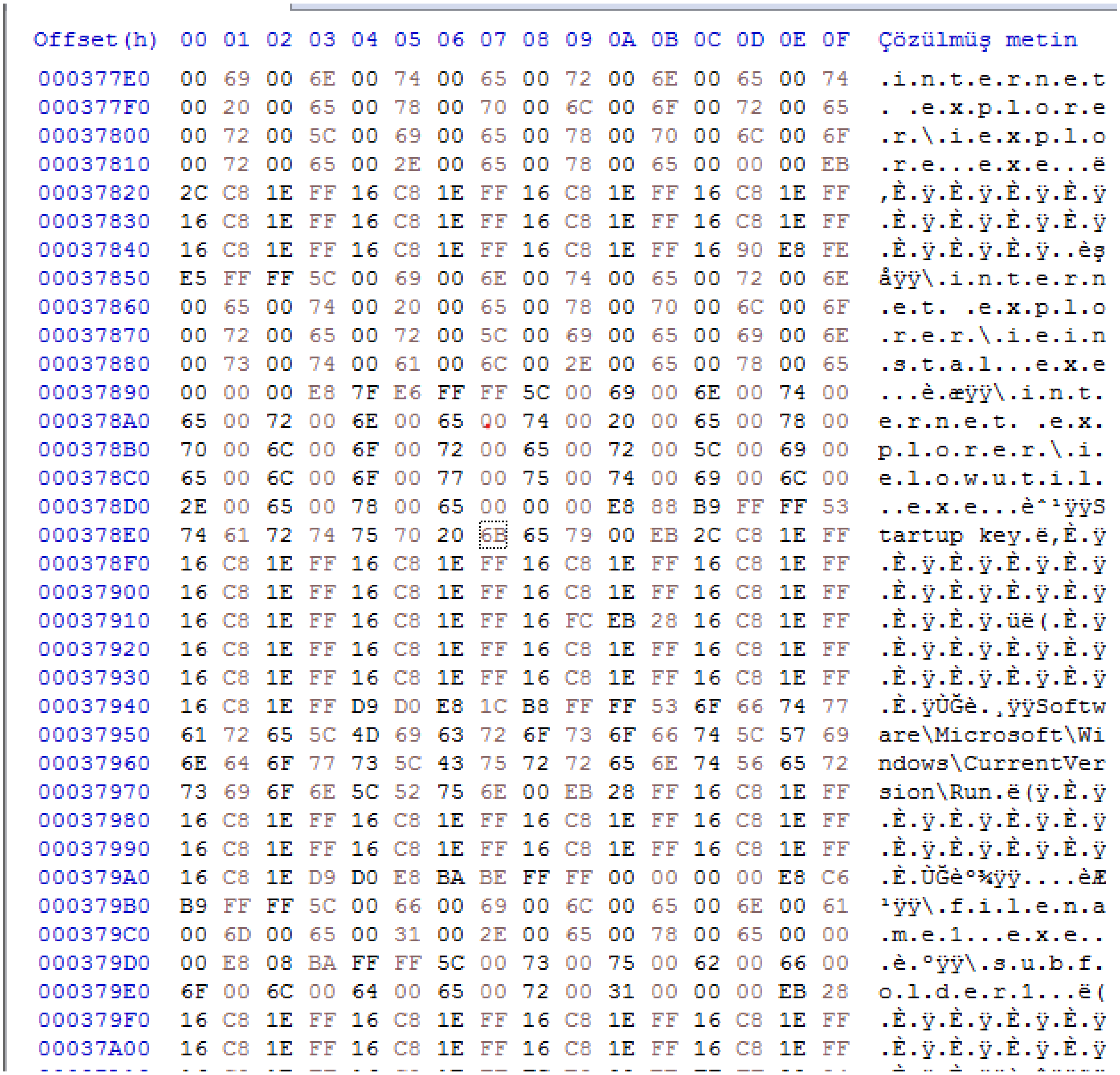
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Çözülmüş metin

000386E0  08 00 00 89 85 C4 00 00 00 F8 8B 4D 18 BA 10 C6  ...‰…Ä...ø‹M.°.Æ
000386F0  96 EB E8 95 08 00 00 FC 89 45 78 90 8B 4D 18 BA  –ëè•...ü‰Ex.‹M.°
00038700  B0 EC 3C 66 E8 83 08 00 00 89 85 B0 00 00 00 8B  °ì<fèƒ...‰…°...‹
00038710  4D 18 BA 07 CA 70 38 E8 70 08 00 00 89 45 54 8B  M.°.Êp8èp...‰ET‹
00038720  4D 18 FC BA 20 C5 91 78 E8 5F 08 00 00 F8 89 85  M.ü° Å'xè_...ø‰…
00038730  20 01 00 00 D9 D0 FC 8B 4D 18 BA 21 99 01 71 E8   ...ÙĞü‹M.°!™.qè
00038740  48 08 00 00 89 85 24 01 00 00 E9 59 03 00 00 F8  H...‰…$...éY...ø
00038750  59 89 8D 8C 00 00 00 FC 90 BA 07 42 17 38 E8 29  Y‰.Œ...ü.°.B.8è)
00038760  08 00 00 89 85 90 00 00 00 8B 8D 8C 00 00 00 BA  ...‰…....‹.Œ...°
00038770  7D 89 F1 E0 E8 13 08 00 00 89 85 94 00 00 00 E9  }‰ñàè....‰…”...é
00038780  66 01 00 00 FC 59 F8 89 4D 5C D9 D0 BA 9E B3 CE  f...üYø‰M\ÙĞ°.³Î
00038790  46 FC E8 F5 07 00 00 89 45 58 8B 4D 5C BA EA 72  Füèõ...‰EX‹M\°êr
000387A0  58 34 E8 E5 07 00 00 89 45 60 F8 C3 FC 90 E8 D3  X4èå...‰E`øÃü.èÓ
000387B0  EA FF FF 68 74 74 70 73 3A 2F 2F 6F 6E 65 64 72  êÿÿhttps://onedr
000387C0  69 76 65 2E 6C 69 76 65 2E 63 6F 6D 2F 64 6F 77  ive.live.com/dow
000387D0  6E 6C 6F 61 64 3F 63 69 64 3D 30 32 45 39 38 38  nload?cid=02E988
000387E0  34 30 41 34 43 39 46 44 36 43 26 72 65 73 69 64  40A4C9FD6C&resid
000387F0  3D 32 45 39 38 38 34 30 41 34 43 39 46 44 36 43  =2E98840A4C9FD6C
00038800  25 32 31 31 31 37 32 26 61 75 74 68 6B 65 79 3D  %211172&authkey=
00038810  41 45 63 67 6D 63 5F 5F 50 38 6E 38 69 72 77 00  AEcgmc__P8n8irw.
00038820  00 00 00 E8 5D FB FF FF 00 EB 28 2A 0D 21 0D 2A  ...è]ûÿÿ.ë(*.!.*
00038830  0D 21 0D 2A 0D 21 0D 2A 0D 21 0D 2A 0D 21 0D 2A  .!.*.!.*.!.*.!.*
00038840  0D 21 0D 2A 0D 21 0D 2A 0D 21 0D 2A 0D 21 0D 2A  .!.*.!.*.!.*.!.*
00038850  0D 21 0D F8 F8 E8 D2 FB FF FF 4D 6F 7A 69 6C 6C  .!.øøèÒûÿÿMozill
00038860  61 2F 35 2E 30 20 28 57 69 6E 64 6F 77 73 20 4E  a/5.0 (Windows N
00038870  54 20 36 2E 31 3B 20 57 4F 57 36 34 3B 20 54 72  T 6.1; WOW64; Tr
```

Figure 2: Malicious Sample 2

If we were based only on the “0x68 0x74 x74 0x70” pattern, we would detect all strings that start with “http” as a remote server, which would significantly increase our false-positive rate.

We have reached the largest and perhaps the only configuration of the GuLoader family, but as a result of the analysis, we have also determined that this version of the malware contains different configurations. This configurations;

- The registry path targeted to ensure persistence on the system,
- The value of set in the targeted registry,
- In which directory of the system and with which name the malware downloaded from the remote server.



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Çözülmüş metin
000377E0  00 69 00 6E 00 74 00 65 00 72 00 6E 00 65 00 74  .i.n.t.e.r.n.e.t
000377F0  00 20 00 65 00 78 00 70 00 6C 00 6F 00 72 00 65  . .e.x.p.l.o.r.e
00037800  00 72 00 5C 00 69 00 65 00 78 00 70 00 6C 00 6F  .r.\.i.e.x.p.l.o
00037810  00 72 00 65 00 2E 00 65 00 78 00 65 00 00 00 EB  .r.e...e.x.e...ë
00037820  2C C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  ,È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037830  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037840  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 90 E8 FE  .È.ÿ.È.ÿ.È.ÿ..èş
00037850  E5 FF FF 5C 00 69 00 6E 00 74 00 65 00 72 00 6E  åÿÿ\.i.n.t.e.r.n
00037860  00 65 00 74 00 20 00 65 00 78 00 70 00 6C 00 6F  .e.t. .e.x.p.l.o
00037870  00 72 00 65 00 72 00 5C 00 69 00 65 00 69 00 6E  .r.e.r.\.i.e.i.n
00037880  00 73 00 74 00 61 00 6C 00 2E 00 65 00 78 00 65  .s.t.a.l...e.x.e
00037890  00 00 00 E8 7F E6 FF FF 5C 00 69 00 6E 00 74 00  ...è.æÿÿ\.i.n.t.
000378A0  65 00 72 00 6E 00 65 00 74 00 20 00 65 00 78 00  e.r.n.e.t. .e.x.
000378B0  70 00 6C 00 6F 00 72 00 65 00 72 00 5C 00 69 00  p.l.o.r.e.r.\.i.
000378C0  65 00 6C 00 6F 00 77 00 75 00 74 00 69 00 6C 00  e.l.o.w.u.t.i.l.
000378D0  2E 00 65 00 78 00 65 00 00 00 E8 88 B9 FF FF 53  ..e.x.e...èˆ¹ÿÿS
000378E0  74 61 72 74 75 70 20 6B 65 79 00 EB 2C C8 1E FF  tartup key.ë,È.ÿ
000378F0  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037900  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037910  16 C8 1E FF 16 C8 1E FF 16 FC EB 28 16 C8 1E FF  .È.ÿ.È.ÿ.üë(.È.ÿ
00037920  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037930  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037940  16 C8 1E FF D9 D0 E8 1C B8 FF FF 53 6F 66 74 77  .È.ÿÙĞè.¸ÿÿSoftw
00037950  61 72 65 5C 4D 69 63 72 6F 73 6F 66 74 5C 57 69  are\Microsoft\Wi
00037960  6E 64 6F 77 73 5C 43 75 72 72 65 6E 74 56 65 72  ndows\CurrentVer
00037970  73 69 6F 6E 5C 52 75 6E 00 EB 28 FF 16 C8 1E FF  sion\Run.ë(ÿ.È.ÿ
00037980  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037990  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
000379A0  16 C8 1E D9 D0 E8 BA BE FF FF 00 00 00 00 E8 C6  .È.ÙĞèº¾ÿÿ....èÆ
000379B0  B9 FF FF 5C 00 66 00 69 00 6C 00 65 00 6E 00 61  ¹ÿÿ\.f.i.l.e.n.a
000379C0  00 6D 00 65 00 31 00 2E 00 65 00 78 00 65 00 00  .m.e.1...e.x.e..
000379D0  00 E8 08 BA FF FF 5C 00 73 00 75 00 62 00 66 00  .è.ºÿÿ\.s.u.b.f.
000379E0  6F 00 6C 00 64 00 65 00 72 00 31 00 00 00 EB 28  o.l.d.e.r.1...ë(
000379F0  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
00037A00  16 C8 1E FF 16 C8 1E FF 16 C8 1E FF 16 C8 1E FF  .È.ÿ.È.ÿ.È.ÿ.È.ÿ
```

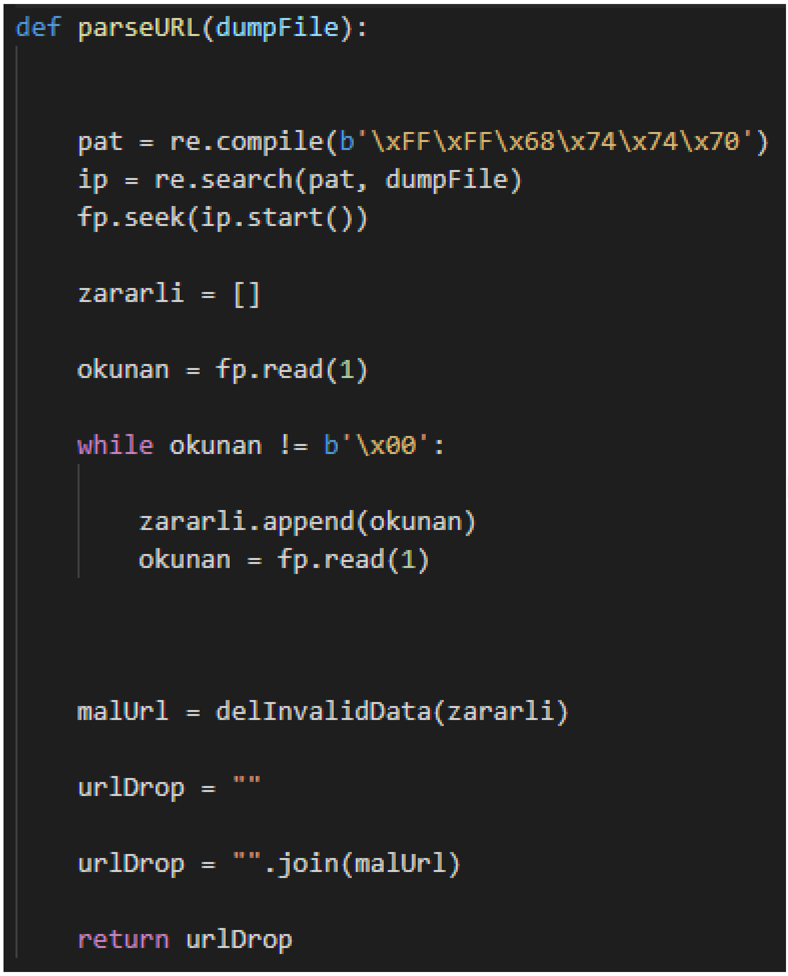
Figure 3: Other Configurations

As a result of the analysis, the permanence mechanism is divided into two in this version of the GuLoader family. The first type of GuLoader instance drops the VBA script to the systems TEMP directory, whose only job is to run a copy of itself. The second type of GuLoader example drops a copy of itself into the system's user directory and runs it through the registry. We need a good concept of these two differences because we will write our Python script accordingly.

Part Four: Writing the Extractor

After all the valuable configurations we find and the roadmap we have created, we can now automate things.

At this point, we first need to write a function that parses the remote server URL, which is the configuration critical to us. Then, the function extracting the User-Agent, which will be included in the HTTP header to connect with the remote server, the function that extracts the path to the targeted registry, the function that extracts the data set in the registry, detects whether the malware is Type 1 or Type 2. According to the function and the type of the malware, we have drawn our roadmap before writing the Python script, including the path and the name of the system directory to which it drops itself.

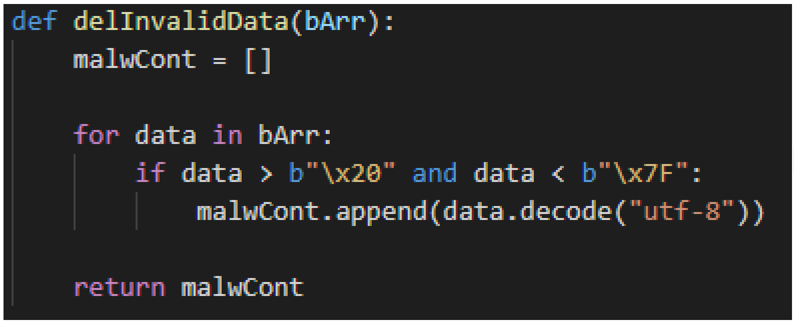


```python
def parseURL(dumpFile):


    pat = re.compile(b'\xFF\xFF\x68\x74\x74\x70')
    ip = re.search(pat, dumpFile)
    fp.seek(ip.start())

    zararli = []

    okunan = fp.read(1)

    while okunan != b'\x00':

        zararli.append(okunan)
        okunan = fp.read(1)



    malUrl = delInvalidData(zararli)

    urlDrop = ""

    urlDrop = "".join(malUrl)

    return urlDrop
```

Figure 4: parseURL Function

In the above parseURL function, we search the memory dump for the pattern that we have extracted by examining the memory dump. Then we move to the starting point of the pattern with File Pointer. (This is the starting offset of the remote server).

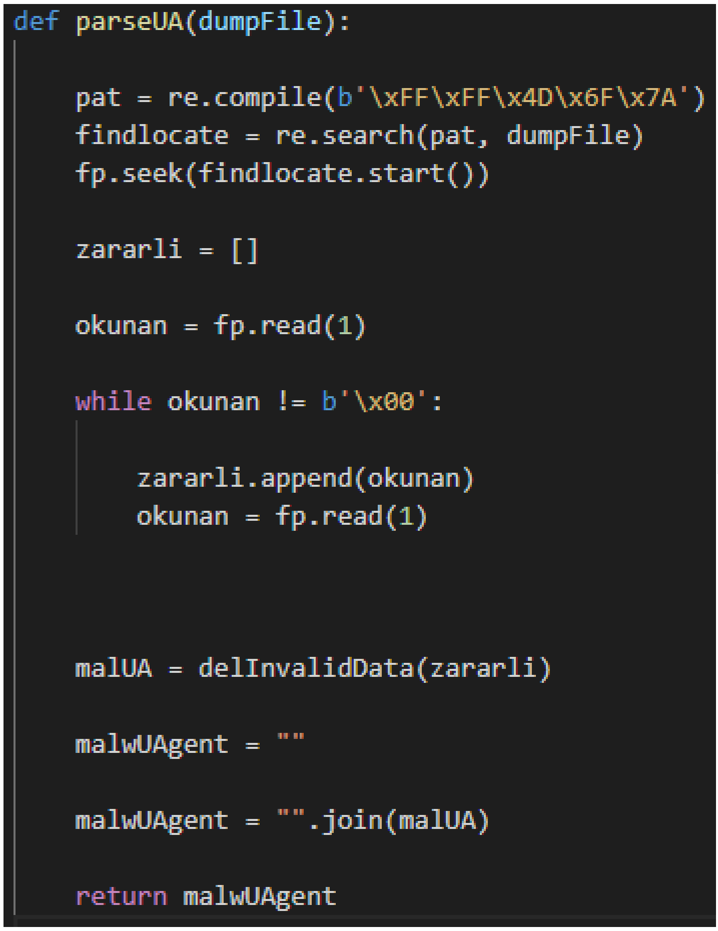
We read a character from the offset we are in and append every character we read to the series called “zararli”. When our loop reads the “0x00” byte, it stops and we come to the end of the remote server address. Then we convert the remote server address, which is one character in the array, into a string and return it to our main function.



```python
def delInvalidData(bArr):
    malwCont = []

    for data in bArr:
        if data > b"\x20" and data < b"\x7F":
            malwCont.append(data.decode("utf-8"))

    return malwCont
```

Figure 5: delInvalidData Function

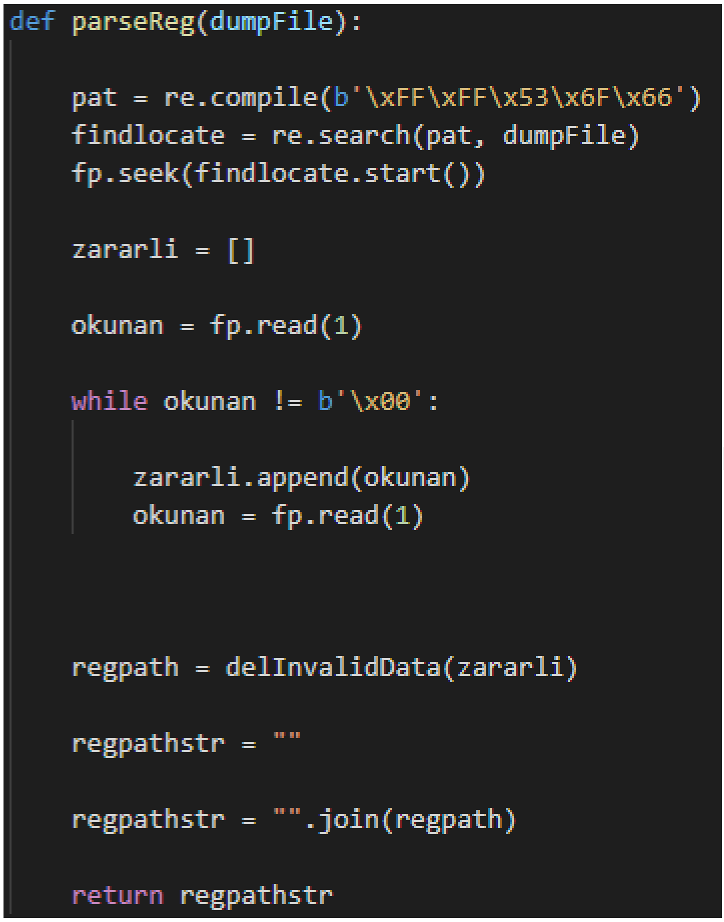
Don’t be confused by the delInvalidData function here. It only deletes characters that are interfering and not found in the ASCII table. You can do the same by passing the errors = “ignore” parameter to the decode () function in Python, but we try to write the script in a structure close to C language and try not to skip the details.



```
def parseUA(dumpFile):

    pat = re.compile(b'\xFF\xFF\x4D\x6F\x7A')
    findlocate = re.search(pat, dumpFile)
    fp.seek(findlocate.start())

    zararli = []

    okunan = fp.read(1)

    while okunan != b'\x00':

        zararli.append(okunan)
        okunan = fp.read(1)



    malUA = delInvalidData(zararli)

    malwUAgent = ""

    malwUAgent = "".join(malUA)

    return malwUAgent
```

Figure 6: parseUA Function

We use the same operations we do in our function that parses the remote server while parsing the User-Agent. Naturally, this function has a separate pattern.

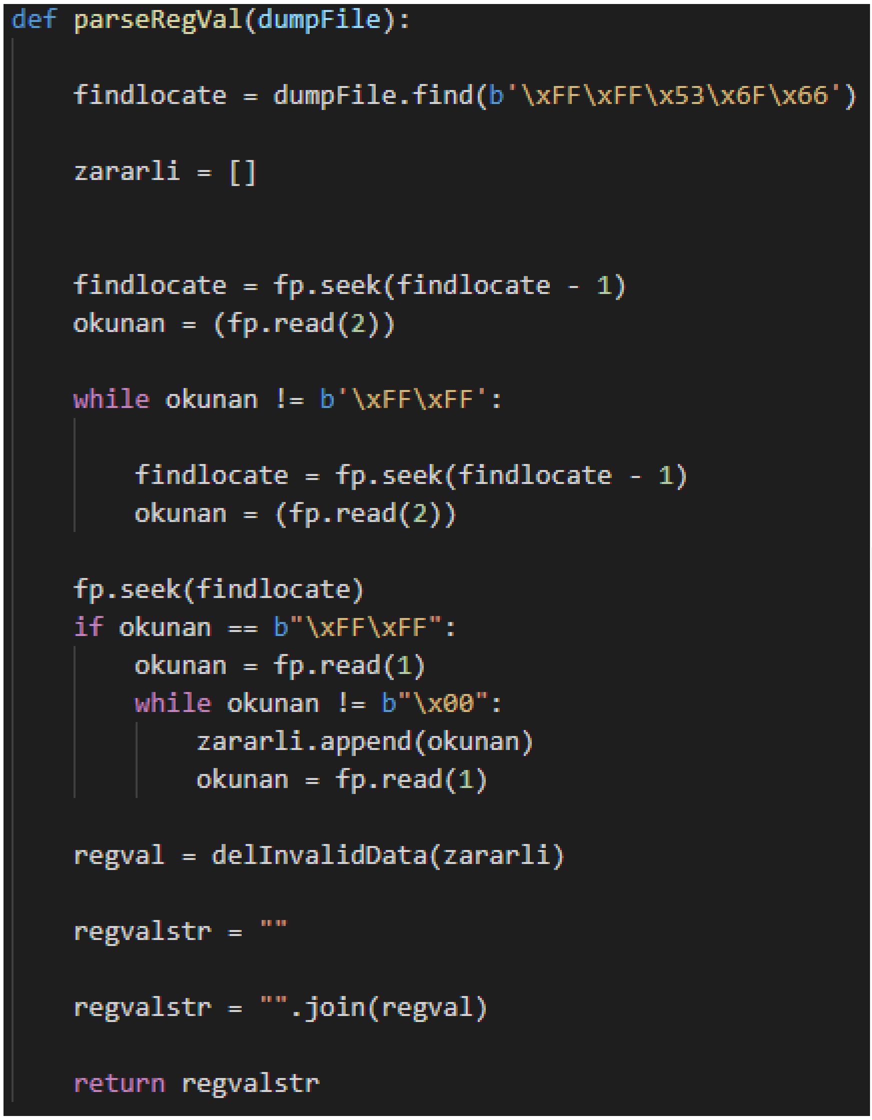


```
def parseReg(dumpFile):

    pat = re.compile(b'\xFF\xFF\x53\x6F\x66')
    findlocate = re.search(pat, dumpFile)
    fp.seek(findlocate.start())

    zararli = []

    okunan = fp.read(1)

    while okunan != b'\x00':

        zararli.append(okunan)
        okunan = fp.read(1)



    regpath = delInvalidData(zararli)

    regpathstr = ""

    regpathstr = "".join(regpath)

    return regpathstr
```

Figure 7: parseReg Function

One of the configurations was the target registry path to provide persistence. We repeat the same processes with the pattern we analyze and extract from the memory dump. This function also shows us the targeted registry path.

Notice we used Python’s re library to find the pattern compile and matching data. You can use the find () function directly, but using regular expressions will be advantageous in many places.



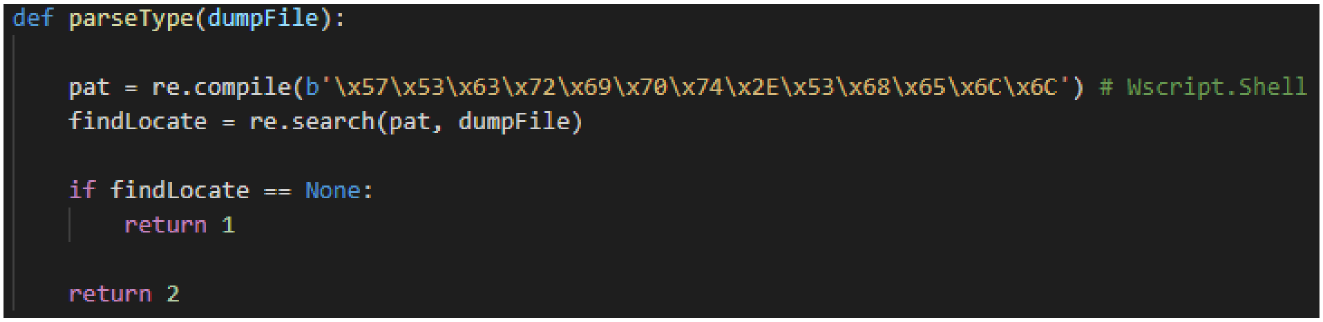
```
def parseRegVal(dumpFile):

    findlocate = dumpFile.find(b'\xFF\xFF\x53\x6F\x66')

    zararli = []


    findlocate = fp.seek(findlocate - 1)
    okunan = (fp.read(2))

    while okunan != b'\xFF\xFF':

        findlocate = fp.seek(findlocate - 1)
        okunan = (fp.read(2))

    fp.seek(findlocate)
    if okunan == b"\xFF\xFF":
        okunan = fp.read(1)
        while okunan != b"\x00":
            zararli.append(okunan)
            okunan = fp.read(1)

    regval = delInvalidData(zararli)

    regvalstr = ""

    regvalstr = "".join(regval)

    return regvalstr
```

Figure 8: parseRegVal Function

After finding the registry path, we need to parse the entered key in the targeted registry. If you remember, the configurations we aimed to remove included the registry key.

This time we show you how to extract the registry key using the find () function to show the difference between re and find (). This time, we understand that we have come to the beginning of the configuration with the bytes “0xFF 0xFF”. That’s why we are doing two byte reads, and we are doing a backward reading by removing the File pointer by 1. Then we read up to the “0x00” byte in a classical way, delete non-ASCII characters and return the parsed registry key to our main function.

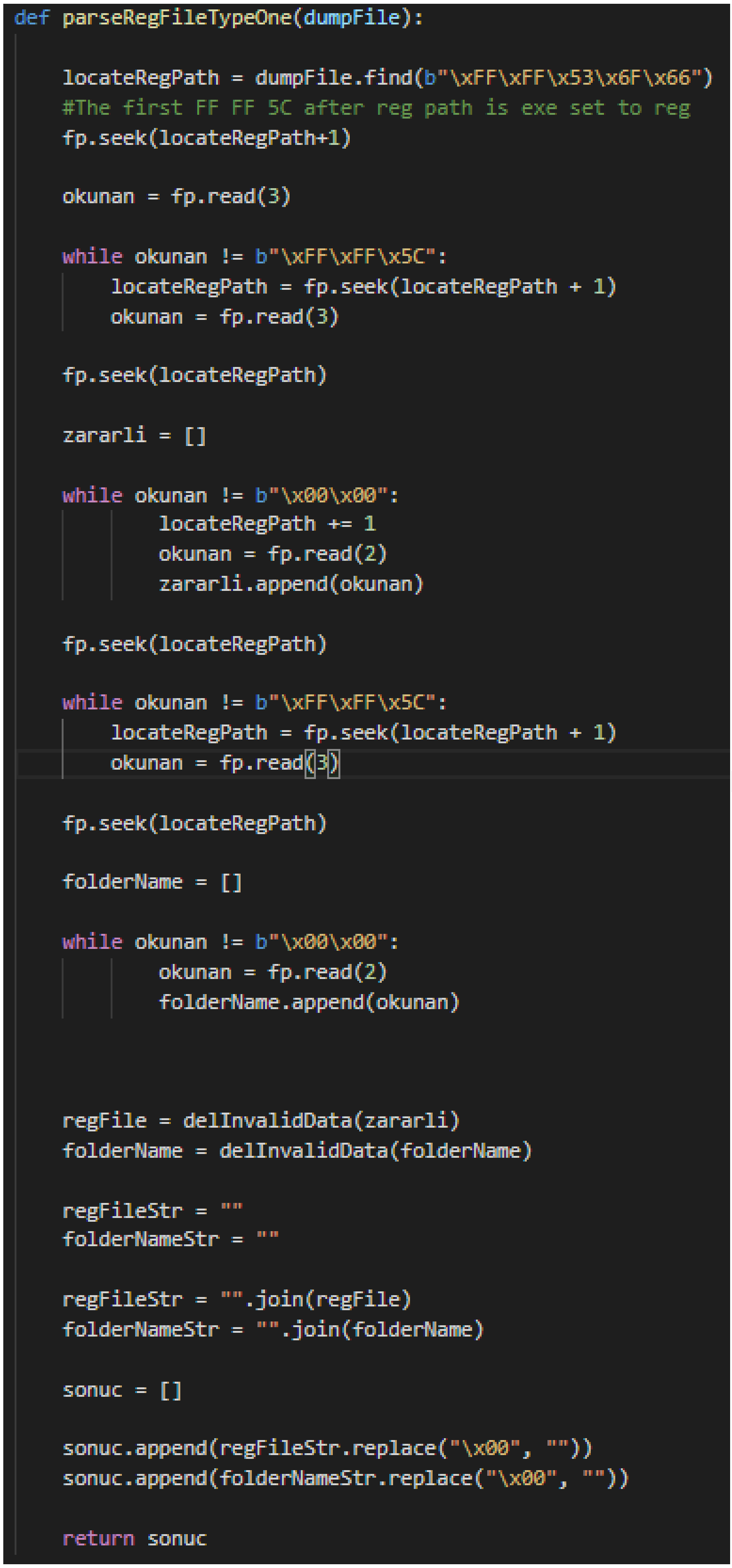
Now all that remains is to learn the persistence type of the malware. After that, we will parse the name in which folder according to its type.



```
def parseType(dumpFile):

    pat = re.compile(b'\x57\x53\x63\x72\x69\x70\x74\x2E\x53\x68\x65\x6C\x6C') # Wscript.Shell
    findLocate = re.search(pat, dumpFile)

    if findLocate == None:
        return 1

    return 2
```

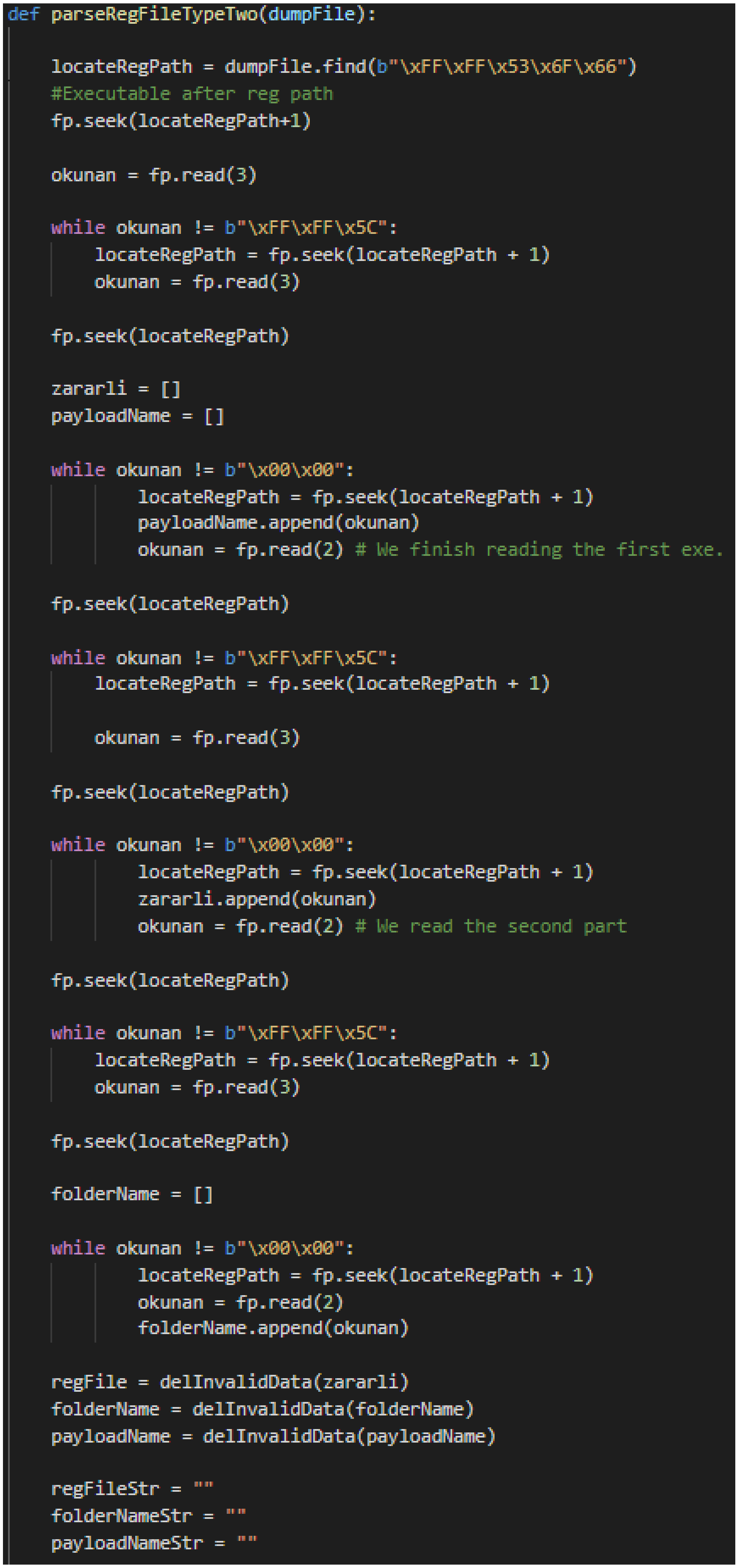
Figure 9: parseType Function

As you can see in the image above, if the malware has the relevant pattern, it is Type 2, if not, it is Type 1. Now, we will write the functions that parse both the created folder name and the name of the malware from the memory dump according to Type 1 and Type 2.



```python
def parseRegFileTypeOne(dumpFile):

    locateRegPath = dumpFile.find(b"\xFF\xFF\x53\x6F\x66")
    #The first FF FF 5C after reg path is exe set to reg
    fp.seek(locateRegPath+1)

    okunan = fp.read(3)

    while okunan != b"\xFF\xFF\x5C":
        locateRegPath = fp.seek(locateRegPath + 1)
        okunan = fp.read(3)

    fp.seek(locateRegPath)

    zararli = []

    while okunan != b"\x00\x00":
            locateRegPath += 1
            okunan = fp.read(2)
            zararli.append(okunan)

    fp.seek(locateRegPath)

    while okunan != b"\xFF\xFF\x5C":
        locateRegPath = fp.seek(locateRegPath + 1)
        okunan = fp.read(3)

    fp.seek(locateRegPath)

    folderName = []

    while okunan != b"\x00\x00":
            okunan = fp.read(2)
            folderName.append(okunan)



    regFile = delInvalidData(zararli)
    folderName = delInvalidData(folderName)

    regFileStr = ""
    folderNameStr = ""

    regFileStr = "".join(regFile)
    folderNameStr = "".join(folderName)

    sonuc = []

    sonuc.append(regFileStr.replace("\x00", ""))
    sonuc.append(folderNameStr.replace("\x00", ""))

    return sonuc
```

Figure 10: parseregFileTypeOne Function



```
def parseRegFileTypeTwo(dumpFile):

    locateRegPath = dumpFile.find(b"\xFF\xFF\x53\x6F\x66")
    #Executable after reg path
    fp.seek(locateRegPath+1)

    okunan = fp.read(3)

    while okunan != b"\xFF\xFF\x5C":
        locateRegPath = fp.seek(locateRegPath + 1)
        okunan = fp.read(3)

    fp.seek(locateRegPath)

    zararli = []
    payloadName = []

    while okunan != b"\x00\x00":
            locateRegPath = fp.seek(locateRegPath + 1)
            payloadName.append(okunan)
            okunan = fp.read(2) # We finish reading the first exe.

    fp.seek(locateRegPath)

    while okunan != b"\xFF\xFF\x5C":
        locateRegPath = fp.seek(locateRegPath + 1)

        okunan = fp.read(3)

    fp.seek(locateRegPath)

    while okunan != b"\x00\x00":
            locateRegPath = fp.seek(locateRegPath + 1)
            zararli.append(okunan)
            okunan = fp.read(2) # We read the second part

    fp.seek(locateRegPath)

    while okunan != b"\xFF\xFF\x5C":
        locateRegPath = fp.seek(locateRegPath + 1)
        okunan = fp.read(3)

    fp.seek(locateRegPath)

    folderName = []

    while okunan != b"\x00\x00":
            locateRegPath = fp.seek(locateRegPath + 1)
            okunan = fp.read(2)
            folderName.append(okunan)

    regFile = delInvalidData(zararli)
    folderName = delInvalidData(folderName)
    payloadName = delInvalidData(payloadName)

    regFileStr = ""
    folderNameStr = ""
    payloadNameStr = ""
```

Figure 11:

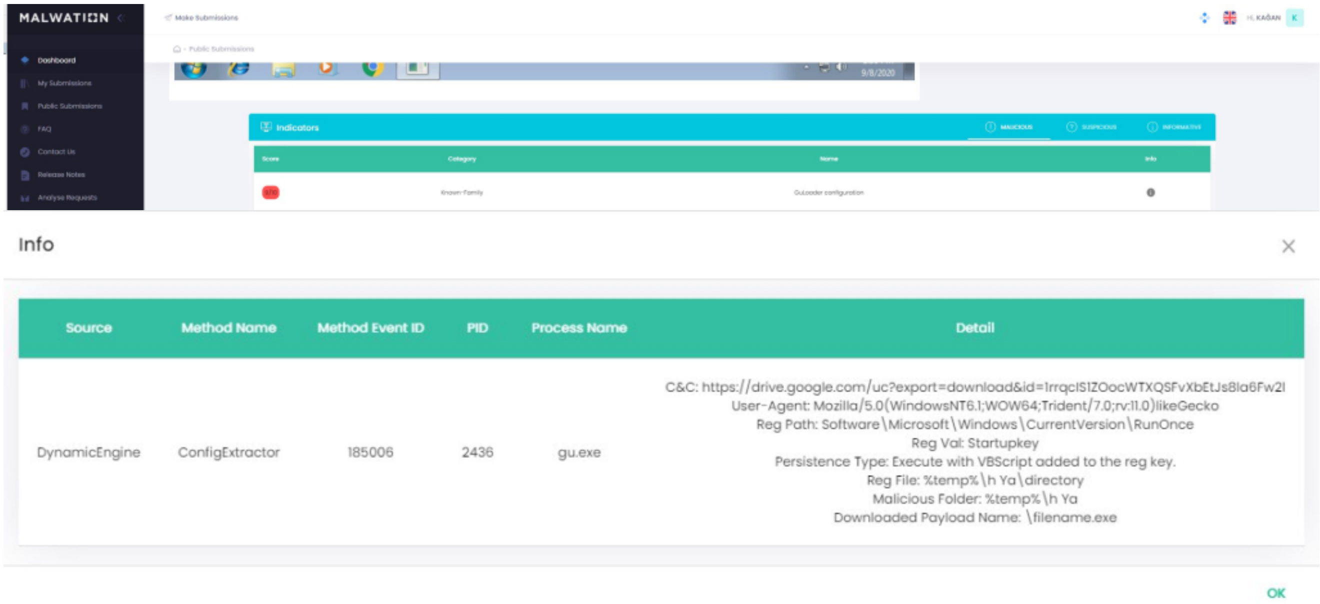
parseRegFileTypeTwo Function

As can be seen in the figures above, there is no secondary VBA script because the malware with Type 1 provides persistence over the registry. The path of the malware is written directly in the registry. However, the malware with Type 2 gives the path of the VBA script to the registry. And VBA script is running at system startup. VBA script also runs the malware with its payload.

In the Type 1 malware, the name of the executable and the name of the folder in which it is located are included in the bytes under the registry configurations in the memory dump, respectively.

In the Type 2 malware, the payload, the name of the executable and the name of the folder in which it is located are included in the bytes under the registry configurations in the memory dump, respectively.

Although the patterns of both types are the same, we just write a few additional code snippets and extract the necessary configurations. In the image below, you can see the output of AIMA's integrated Config Extractor module.



Score	Category	Name	Info
[illegible]	Known-Family	GuLoader configuration	●

Info

Source	Method Name	Method Event ID	PID	Process Name	Detail
DynamicEngine	ConfigExtractor	185006	2436	gu.exe	C&C: https://drive.google.com/uc?export=download&id=1rrqclS1ZOocWTXQSFvXbEtJs8la6Fw2I User-Agent: Mozilla/5.0(WindowsNT6.1;WOW64;Trident/7.0;rv:11.0)likeGecko Reg Path: Software\Microsoft\Windows\CurrentVersion\RunOnce Reg Val: Startupkey Persistence Type: Execute with VBScript added to the reg key. Reg File: %temp%\h Ya\directory Malicious Folder: %temp%\h Ya Downloaded Payload Name: \filename.exe

We are waiting for your feedback and see you in our next Extraction article, we say goodbye.

malwation