APT REPORTS

MontysThree: Industrial espionage with steganography and a Russian accent on both sides

By Denis Legezo on October 8, 2020. 10:00 am

CONTENTS

In summer 2020 we uncovered a previously unknown multi-module C++ toolset used in highly targeted industrial espionage attacks dating back to 2018. Initially the reason for our interest in this malware was its rarity, the obviously targeted nature of the campaign and the fact that there are no obvious similarities with already known campaigns at the level of code, infrastructure or TTPs. To date, we consider this toolset and the actor behind it to be new. The malware authors named the toolset “MT3”; following this abbreviation we have named the toolset “MontysThree”.

IN THE SAME CATEGORY

MosaicRegressor: Lurking in the Shadows of UEFI

Transparent Tribe: Evolution analysis, part 2

Lifting the veil on DeathStalker, a mercenary triumvirate

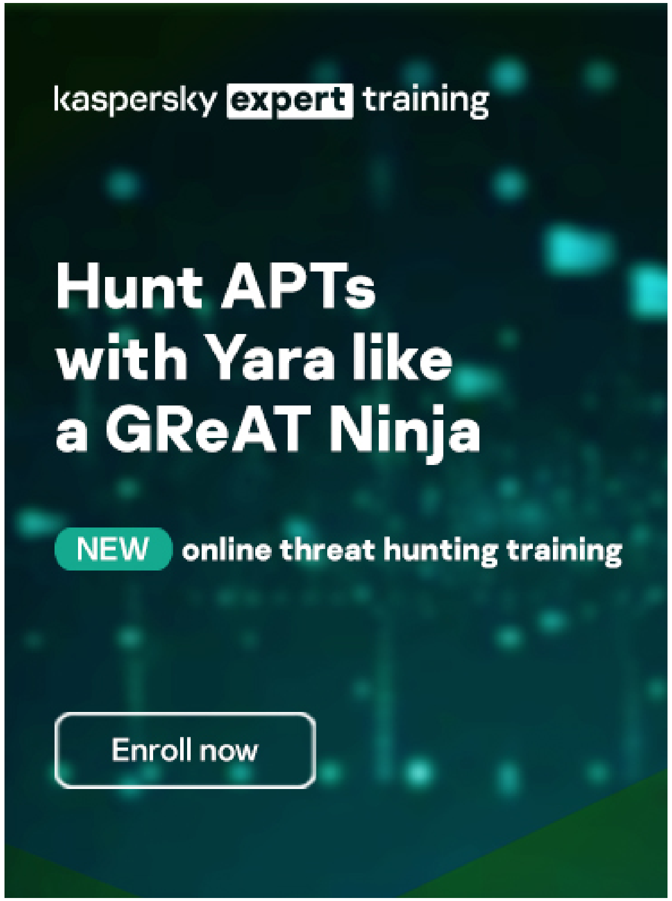
Transparent Tribe: Evolution analysis, part 1

CactusPete APT group’s updated Bisonal backdoor

Following the MT3 abbreviation we named the toolset MontysThree

The malware includes a set of C++ modules used for persistence, obtaining data from a bitmap with steganography, decryption of configuration tasks (making screenshots, fingerprinting the target, getting the file, etc.) and their execution, and network communications with major legitimate public cloud services such as Google, Microsoft and Dropbox. MontysThree is configured to search for specific Microsoft Office and Adobe Acrobat documents stored in current documents directories and on removable media. The malware uses custom steganography and several encryption schemes: besides custom XOR-based encryption, the modules rely on 3DES and RSA algorithms for configuration decryption and communications.

MontysThree contains natural language artifacts of proper Russian language and configuration that seek directories that exist only on Cyrilic localised Windows versions. While most external public cloud communications use token-based authorisation, some samples contain email-based accounts for them, which pretend to be a Chinese lookalike. We consider these names to be false flags. Many more artifacts suggest that the malware was developed by a Russian-speaking actor and is targeting Cyrillic Windows versions.



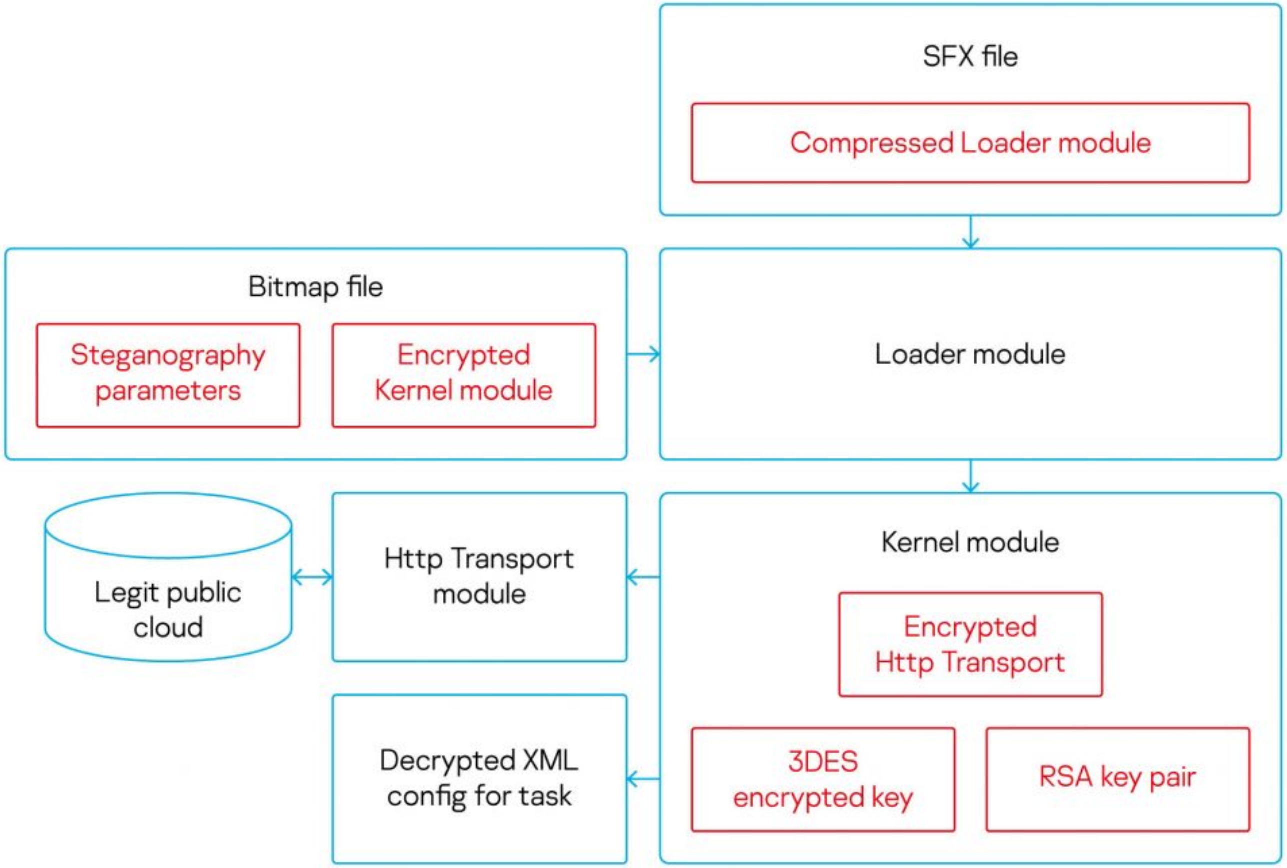
How the malware spreads

The initial loader module is spread inside RAR self-extracting archives (SFX) with names related to employees' phones list, technical documentation and medical test results. There are no lures, only PE files (masquerading a .pdf or .doc file), but such titles are now a typical trick used in spear-phishing – "corporate info update" or "medical analysis results". One of the loaders (MD5 da49fea229dd2dedab2b909f24fb24ab) has the name "Список телефонов сотрудников 2019.doc" ("Employee phone list", in Russian). Other loaders have the names "Tech task.pdf" and "invitro-106650152-1.pdf". The latter is the name of a medical laboratory in Russia. All of them seem like typical spear-phishing tricks. The SFX script is as follows:

```
Path=%TEMP%\
SavePath
Setup=rundll32.exe "invitro-106650152-1.pdf",Open
Silent=1
Overwrite=1
Update=U
Delete=invitro-106650152-1.pdf
```

On execution, the SFX script calls the Open() function (we'll return to this exported name) of the decompressed loader executable in the %TEMP% directory and deletes it. Judging by the filename, it most likely imitates medical analysis results, given that "Invitro" is a prominent medical laboratory in Russia. This initial PE32 is the first loader module.

How modules work and communicate



Execution flow of MontysThree's modules

The diagram above shows the overall execution flow of the MontysThree modules. Four modules and their features are listed in the table below. The modules share common communication conventions. When dealing with shared data, such as the configuration and detailed execution log, the malware initializes the structure in thread local storage (TLS), which in its turn refers to heap structures. Interestingly, besides RAM, the execution log is stored on disk in a file, encrypted with a one-byte XOR.

The entry point DllEntryPoint() works just like a construtor, which allocates the structure with TlsAlloc() and saves it in a global variable. Modules must export a function named Open(), which takes no parameters (but could parse the command line) and returns a four-byte error code.

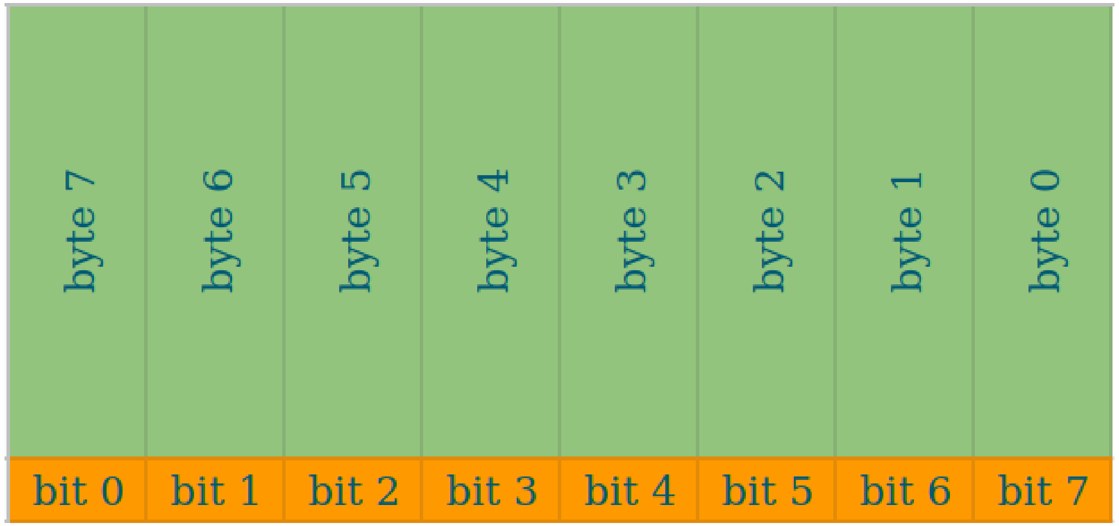
Module name	Features
Loader	This anti-detection module is in charge of custom steganography, kernel module decryption.
Kernel	This kernel (main) module is in charge of decrypting the config XML, then parsing and executing the corresponding tasks in it.
HttpTransport	Network module to communicate with Google, Microsoft, Dropbox legitimate public cloud services, as well as with WebDAV sources. The module is able to make requests through RDP and Citrix in a naive way using legitimate clients.
LinkUpdate	Persistence module is a Windows Quick Launch .lnk modifier. With this naive persistence method users would run the Loader module by themselves every time along with the browsers from the Windows Quick Launch toolbar.

Now let’s take a look how the developers mixed strong modern cryptography standards with custom XOR-based ones.

Task	Encryption in use
Steganography	To decrypt the kernel module the initial loader uses a custom algorithm.
Logs encryption	The malware logs exist in memory as well as in encrypted files on disk at the same time. In RAM the developers store the logs in plaintext, on disk they use one-byte XOR.
Config encryption	Kernel module uses strong encryption algorithms. Configuration data is encrypted with 3DES and the key is encrypted using RSA. All the keys – RSA public/private as well as encrypted 3DES – are stored inside the module’s .data section.
Network module encryption	Initially encrypted HttpTransport is made of four binary blobs stored in the kernel module. The kernel concatenates them and decrypts them with a custom XOR-based algorithm. A round key of four bytes length is used
Communications encryption	The encryption algorithm is RSA using the same public and private keys stored inside the kernel module .data section.

Loader module: Bitmap decryptor and next stage launcher

If the filename of the bitmap containing the steganography-encrypted data is provided to the loader as an argument, the loader decrypts the next stager from the pixel array. In the first iteration, it extracts the steganography parameter data. To do so, the algorithm takes the last bits of the bytes.



The IID, IParam and ISize parameters are kept in the first 384 bytes of the pixel array, meaning that only the last bit of every pixel array's byte is needed. As a result, the module gathers 48 bytes of steganography configuration structure with the fields, determining the next decryption stages.

Field	Offset	Features
IID	0x00	Determines one or two decryption layers would apply to the following pixel array.
IParam	0x04	Determines which bits from pixel arrays bytes would form the next kernel module.
ISize	0x28	The decrypted kernel module's resulting size.

After extracting the steganography parameters, the next stager is decrypted using a two-step algorithm. Firstly, the IParam algorithm chooses the bits from the pixel array's bytes. Then, if IID equals 2, a custom dexoring operation using a four-byte round key is applied on the gathered bytes. The initial key for the first four-byte decryption has the hardcoded value 0x23041920. Then the formula for the round XOR key for the next bytes is:

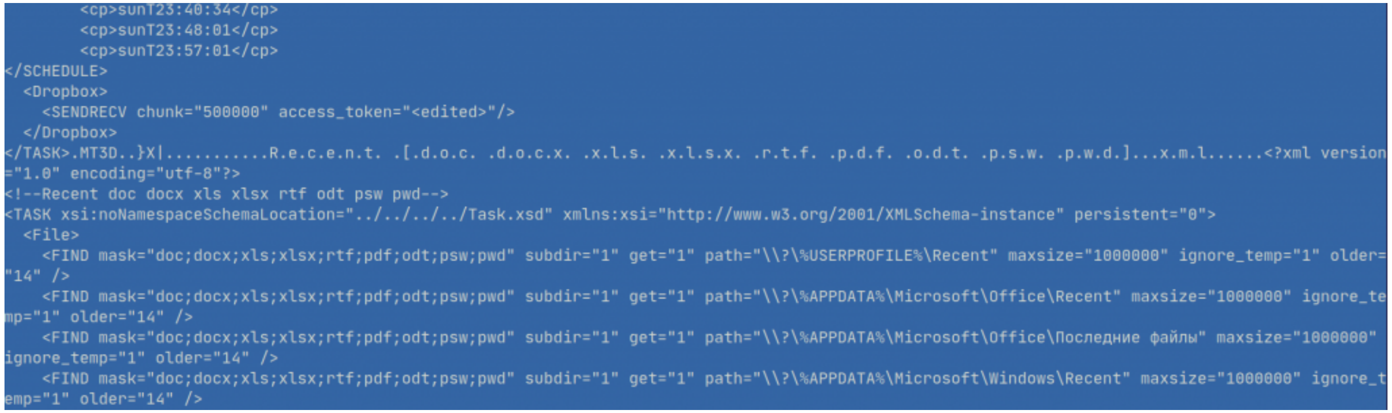
```
key ^= 8 * (key ^ (key << 20))
```

We consider this steganography algorithm to be custom made and not taken from some open source third-party repository. Surprisingly, the decryption result is not injected into some process's memory, but dropped to disk as a file named msgslang32.dll. The loader then simply uses the Windows API functions LoadLibraryW() and GetProcAddress() to run the next stager's Open() function, as we previously saw with the loader module.

Kernel module: Config decryptor and tasks dispatcher

The kernel module contains three encryption keys used for configuration decryption and C2 communications. Public and private RSA keys are stored in the .data section as PUBLICKEYBLOB and PRIVATEKEYBLOB respectively. These are used to encrypt C2 communications and to decrypt the 3DES key as well. The third 3DES key is also stored in the .data section in its encrypted form; this key is used to decrypt an embedded .cab archive containing the XML config. To decompress the .cab archive the module uses Window’s standard system utility, “expand.exe”. We’ll see another common software usage in the HttpTransport module.

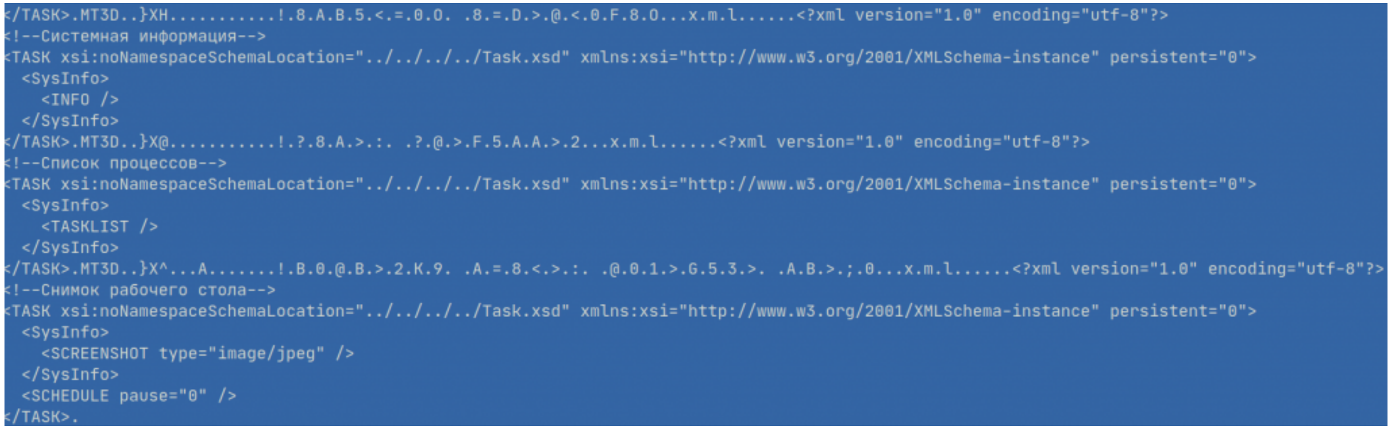
The XML configuration contains valuable data that helps us understand the campaign operator’s interest. It is structured using various “tasks” for the malware, such as fingerprinting the target using its OS version, process list and capturing screenshots; but also grabs the list of users’ recent documents with any of the extensions .doc, .docx, .xls, .xlsx, .rtf, .pdf, .odt, .psw, .pwd from the several recent documents directories in %USERPROFILE% and %APPDATA%, including %APPDATA%\Microsoft\Office\Последние файлы. This folder name translates to “Recent files” in Russian, suggesting that the malware is aimed at Cyrillic localised Windows versions.



```
        <cp>sunT23:40:34</cp>
        <cp>sunT23:48:01</cp>
        <cp>sunT23:57:01</cp>
</SCHEDULE>
  <Dropbox>
    <SENDRECV chunk="500000" access_token="<edited>"/>
  </Dropbox>
</TASK>.MT3D..}X|...........R.e.c.e.n.t. .[.d.o.c. .d.o.c.x. .x.l.s. .x.l.s.x. .r.t.f. .p.d.f. .o.d.t. .p.s.w. .p.w.d.]...x.m.l......<?xml version
="1.0" encoding="utf-8"?>
<!--Recent doc docx xls xlsx rtf odt psw pwd-->
<TASK xsi:noNamespaceSchemaLocation="../../../../Task.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" persistent="0">
  <File>
    <FIND mask="doc;docx;xls;xlsx;rtf;pdf;odt;psw;pwd" subdir="1" get="1" path="\\?\%USERPROFILE%\Recent" maxsize="1000000" ignore_temp="1" older=
"14" />
    <FIND mask="doc;docx;xls;xlsx;rtf;pdf;odt;psw;pwd" subdir="1" get="1" path="\\?\%APPDATA%\Microsoft\Office\Recent" maxsize="1000000" ignore_te
mp="1" older="14" />
    <FIND mask="doc;docx;xls;xlsx;rtf;pdf;odt;psw;pwd" subdir="1" get="1" path="\\?\%APPDATA%\Microsoft\Office\Последние файлы" maxsize="1000000"
ignore_temp="1" older="14" />
    <FIND mask="doc;docx;xls;xlsx;rtf;pdf;odt;psw;pwd" subdir="1" get="1" path="\\?\%APPDATA%\Microsoft\Windows\Recent" maxsize="1000000" ignore_t
emp="1" older="14" />
```

Config holds the tasks scheduling (screenshot top), access tokens (here Dropbox, redacted), directories and extensions of interest. One directory exists only on Cyrillic Windows localized versions

We observed several Cyrillic text strings such as “Снимок рабочего стола” (desktop snapshot), “Системная информация” (system information), “Время выхода” (exit time).



```
</TASK>.MT3D..}XH...........!.8.A.B.5.<.=.0.0. .8.=.D.>.@.<.0.F.8.O...x.m.l......<?xml version="1.0" encoding="utf-8"?>
<!--Системная информация-->
<TASK xsi:noNamespaceSchemaLocation="../../../../Task.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" persistent="0">
  <SysInfo>
    <INFO />
  </SysInfo>
</TASK>.MT3D..}X@...........!.?.8.A.>.:. .?.@.>.F.5.A.A.>.2...x.m.l......<?xml version="1.0" encoding="utf-8"?>
<!--Список процессов-->
<TASK xsi:noNamespaceSchemaLocation="../../../../Task.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" persistent="0">
  <SysInfo>
    <TASKLIST />
  </SysInfo>
</TASK>.MT3D..}X^...A.......!.B.0.@.B.>.2.K.9. .A.=.8.<.>.:. .@.0.1.>.G.5.3.>. .A.B.>.;.0...x.m.l......<?xml version="1.0" encoding="utf-8"?>
<!--Снимок рабочего стола-->
<TASK xsi:noNamespaceSchemaLocation="../../../../Task.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" persistent="0">
  <SysInfo>
    <SCREENSHOT type="image/jpeg" />
  </SysInfo>
  <SCHEDULE pause="0" />
</TASK>.
```

Config tasks description starts with MT3D and contains proper short phrases in Russian

The decrypted config structure is as follows:

Field	Size	Content
Magic	4 bytes	MT3D. All parsed files must have this as a prefix to be valid
Creation time	4 bytes	Timestamp, task config creation time stored as Epoch time
Header size	4 bytes	Header size has to be greater than 18. Observed value is e.g. 0x7E
XML size	4 bytes	XML task description has to be greater than zero. Observed value is e.g. 0x662D
XML body	XML size	The task's description and schedule in XML format

While the samples we looked at didn't contain RTTI information, the execution logs allowed us to recover the C++ class names. After the kernel module parses the tasks from the configuration into memory, the main class that processes the instruction is CTask. CTask's IoControl() method is in charge of handling the corresponding tasks and in turn runs the following methods:

CTask method	Features
MainIoControl()	Handler of "Main" task in XML. In case of a RESET command the file, serving as a "pipe", will be deleted. Any other command here will be logged, but not executed
FileIoControl()	Handler of "File" task with PUT, DEL, FIND, WATCH, WATCH_REMOVABLE, RUN and LOGS subcommands
SysInfoIoControl()	Handler of "SysInfo" task with SCREENSHOT, INFO and TASKLIST subcommands
HttpIoControl()	Handler of "Http" task with SENDRECV subcommand

GDriveIoControl()	Handler of “GDrive” task with SENDRECV subcommand
DropboxIoControl()	Handler of “Dropbox” task with SENDRECV subcommand

All methods used for external communications first decrypt the HttpTransport module and use it to transmit the corresponding data RSA-encrypted. The RSA keys in use are the same aforementioned keys used to decrypt the 3DES config key. In a separate Window procedure, the malware monitors if a USB device is plugged in, searching for files of interest.

HttpTransport module: network tasks

The HttpTransport module exists as four encrypted chunks of data inside the .text section of the kernel module. When the kernel needs to communicate, it decrypts this module and, as usual for MontysThree, runs the Open() function, passing command line arguments.

Depending on the arguments transmitted from the kernel module, the module may upload or download content using RDP, WebDAV, Citrix and HTTP protocols. Downloading data from Google and Dropbox public services using user tokens is implemented in HttpTransport as well. In case of HTTP GET/POST requests, the malware would receive a steganography bitmap picture using Windows API HTTP-related functions from a corresponding URL.

The aforementioned communication protocols themselves aren’t implemented inside the module. The malware authors make use of legitimate Windows programs like RDP, Citrix clients and Internet Explorer already installed on the target’s machine. For example, the module executes a task to send some data to a URL and receive the reply through an RDP connection as follows: edit the .rdp file to silently run Internet Explorer on the remote machine; paste the URL to the browser via the clipboard; wait and paste the contents to the opened web page via the clipboard as well; wait and receive the result through the clipboard again.

To copy data, the malware literally sends Ctrl+C, Ctrl+V and Ctrl+A. Perhaps it’s the first time we have seen such a method of “RDP communication”. The Citrix communication is done using a similar procedure: the malware doesn’t implement the protocol but rather searches for Windows Quick Launch .lnk for XenApp pnagent.exe, runs Internet Explorer remotely and communicates with it through the clipboard using special keyboard shortcuts.

Dropbox and Google data upload and download relies on another principle: its implementation uses the custom class CSimpleHttp to authenticate and send HTTP requests. For WebDAV communication, the developers simply use the “net use” Windows command.

LinkUpdate

This auxiliary module is in charge of achieving persistence on the host. It changes the .lnk files in the Windows Quick Launch panel to run the loader along with legitimate applications such as browsers when the user executes them using the modified link.

Who is behind this malware

As we mentioned at the beginning, to date we have observed no similarities or overlaps with known campaigns in terms of TTPs, infrastructure or malware code. So far, we attribute this activity and the use of MontysThree to a new actor. Some samples contain account details used for communicating with public cloud services, which pretend to be of Chinese origin. Taking into consideration all the aforementioned Cyrilic artefacts, we consider these account names to be false flags.

We assume that the actor behind MontysThree is both Russian-speaking and is going after Russian-speaking targets. Some of the filenames of the RAR SFX archives used for spreading the malware were written in Russian and referenced a Russian medical laboratory, used to entice the user to open the file. The XML configuration showcased data fields and Windows titles written in Russian, as well as specific folder paths that exist on Cyrilic localised versions of Windows. We also saw some grammatical errors in the malware's English log message strings.

Let's sum up

Typically we see targeted malware that is mostly going after governmental entities, diplomats and telecom operators, which are fruitful for state-sponsored actors. Industrial espionage cases like MontysThree are far more rare.

The overall campaign sophistication doesn't compare to top notch APT actors in terms of spreading, persistence method. And some aspects of the malware – logging in RAM and files at the same time, keeping the encryption keys in the same file, running an invisible browser on the remote RDP host – seem immature and amateurish in terms of malware development.

On the other hand, the amount of code and therefore effort invested, in MontysThree is significant. The toolset demonstrates some tech-savvy decisions: Storing 3DES key under RSA encryption, custom steganography to avoid IDS and the use of legitimate cloud storage providers to hide the C2 traffic.

File Hashes

Loader
1B0EE014DD2D29476DF31BA078A3FF48
0976C442A06D2D8A34E9B6D38D45AE42
A2AA414B30934893864A961B71F91D98

Kernel
A221671ED8C3956E0B9AF2A5E04BDEE3
3A885062DAA36AE3227F16718A5B2BDB
3AFA43E1BC578460BE002EB58FA7C2DE

HttpTransport
017539B3D744F7B6C62C94CE4BCA444F
501E91BA1CE1532D9790FCD1229CBBDA
D6FB78D16DFE73E6DD416483A32E1D72

Domains and IPs

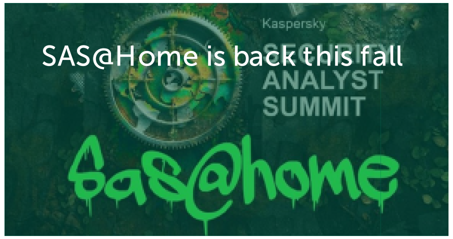
autosport-club.tekcities[.]com
dl10-web-stock[.]ru
dl16-web-eticket[.]ru
dl166-web-eticket[.]ru
dl55-web-yachtbooking[.]xyz

INDUSTRIAL THREATS MALWARE DESCRIPTIONS RUSSIAN-SPEAKING CYBERCRIME TARGETED ATTACKS

Share post on:

Related Posts

MosaicRegressor: Lurking in the Shadows of UEFI


SAS@Home is back this fall

An overview of targeted attacks and APTs on Linux

THERE IS 1 COMMENT

Hiep

Posted on October 9, 2020. 3:14 am

https://otx.alienvault.com/pulse/5f7f1fbb47295b54fbf69b6a/?utm_userid=Minhtri15&utm_medium=InProduct&utm_source=OTX&utm_content=Email&utm_campaign=new_pulse_from_subscribed

REPLY

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Enter your comment here

Name *

Email *

□

Save my name, email, and website in this browser for the next time I comment.

□ Notify me when new comments are added.

SUBMIT

□ I'm not a robot

reCAPTCHA
Privacy - Terms

kaspersky

Contact us | Privacy Policy | License Agreement

Email

□ I'm not a robot

reCAPTCHA
Privacy - Terms

□ I agree to provide my email address to “AO Kaspersky Lab” to receive information about new posts on the site. I understand that I can withdraw this consent at any time via e-mail by clicking the “unsubscribe” link that I find at the bottom of any e-mail sent to me for the purposes mentioned above.