The ‘Madi’ infostealers - a detailed analysis

By Nicolas Brulez

Published: 2012-07-27 · Archived: 2026-04-05 17:55:21 UTC

On 17 July, we published a blog about Madi and the ongoing campaign used to infiltrate computer systems throughout the Middle East that has targeted users in Iran, Israel, Afghanistan and other individuals scattered across the globe. Here is the follow up with a detailed analysis of the infostealer used in the campaign.

Installation

The infostealer is installed by one of the various downloaders used in the attacks, which can be separated into two categories:

- Downloaders using the social engineering techniques described in our first blog post (displaying pictures, movies, documents etc.) to trick the user
- Downloaders that simply download and install the infostealer

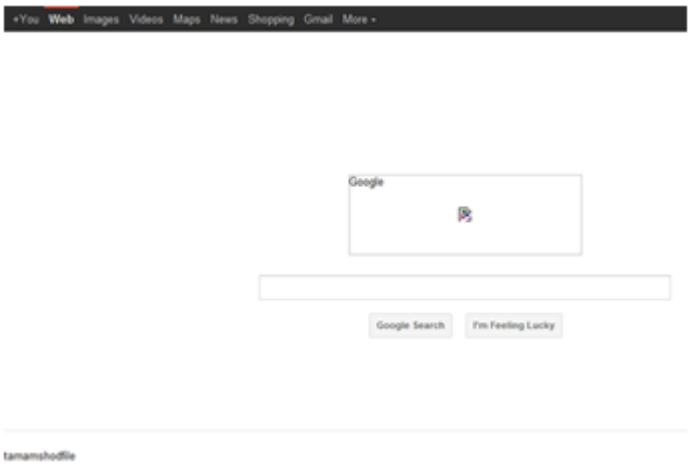
Both types of downloaders copy themselves as “UpdateOffice.exe” into the “Printhood” directory, e.g.: “C:Documents and Settings%USER%PrintHoodUpdateOffice.exe” where they start executing.

Both the infostealer and downloaders create fake files with random names in their respective folders. The downloaders also drop some files which assist the malware (see our first blog for details).

Only one file will be used by the infostealer: nam.dll. This file is created by the downloader in the “Templates” directory (e.g.: “C:Documents and Settings%USER%Templatesnam.dll”) and contains a BOT prefix/build that will be used by the infostealer when connecting to the command and control server (C&C). In order to download and install the infostealer, the downloaders connect to the C&C server to request an HTM page.

Older variants use http://[C&C address]/ASLK/khaki/Abi/UUUU.htm, whereas more recent ones use “http://[C&C address]/ASLK/asgari/mah/UeUeUeUe.htm”.

The HTM page is a copy of Google index, with a double BASE64 encoded executable embedded in the page:



The keyword “tamamshodfile” at the bottom will be explained in the ‘Infostealer analysis’ section below.

The downloaders simply parse the HTM file, and decode the Base64 payload twice and save the resulting PE file as “iexplore.exe” in the “Templates” directory. Once downloaded, the infostealer is executed.

Infostealer analysis: Iexplore.exe

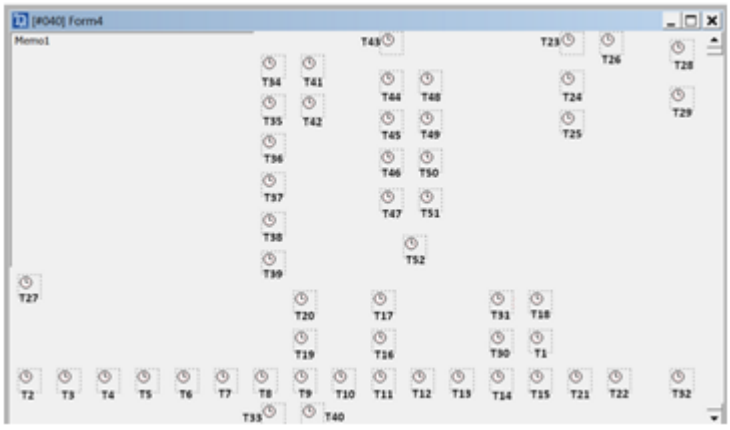
All the versions of the infostealer have an Internet Explorer icon, and were written in Delphi.

The version used in this article, which appears to have been compiled on 10 June 2012, is packed using UPX 3.08.

The file is rather big: 415 KB packed, and 1.14 MB once unpacked.

One peculiarity of the infostealer used in the Madi campaign is the heavy use of Delphi Timers. There are 52 of them as you can see on the screenshot below:

.



Numerous bugs were discovered during the analysis of the infostealer. Some of them won’t be discussed here as we don’t want to help the authors improve their malware.

TForm4.FormCreate:

Upon execution, the first activity of interest performed by the infostealer happens inside TForm4.FormCreate.

It starts with the setup of a keylogger. In order to do so, Madi infostealer uses the Windows function "SetWindowsHookEx" with the "WH_KEYBOARD_LL" Id_Hook.

Once the keylogger has been installed, the infostealer reads the "nam.dll" file (dropped by the downloader) to get the BOT prefix and concatenates it with the computer name. Hereafter this will be referred to as "BOTID_TMP". The final BOTID contains some numbers derived from the "C:" Volume Serial Number, as we will see later on.

The following timers are then disabled in this specific order:

Timer1, Timer16, Timer18, Timer17, Timer20, Timer19, Timer24, Timer8, Timer30, Timer31, Timer33, Timer34, Timer36, Timer37, Timer38, Timer39, Timer40, Timer41, Timer44, Timer45, Timer46, Timer48, Timer49, Timer50.

The malware uses a lot of external files to receive commands, which is another indicator of poor programming skills. Those files are used to inform the malware about the infection status. In order to avoid confusion, hereafter, when referring to a file, it is in the malware directory ("Templates" directory), unless stated otherwise.

The infostealer looks for the following files:

"fsdiskget.dll": If found, it enables Timer 23 – otherwise, disables it.

"nrbindek.dll" : If found, it enables Timer 28 – otherwise, disables it.

"specialfile.dll": If found, it deletes it.

"filesend.xls": Doesn't actually look for it; just tries to delete it.

"begirnagir.htp" : If NOT found, it disables Timer3

"filebind.xls": If found, it enables Timer29 – otherwise, disables it.

Next, Timer14 and Timer13 are both disabled.

The Trojan looks for "First.dll", which is created the first time the malware is executed.

If already present, the code returns from TForm4.FormCreate. Otherwise, the following happens.

It creates first.dll with a hardcoded stream of bytes (not a real .dll, like the .dll mentioned above, as we will see later on when we analyze the timers more closely).

Like the downloaders, the infostealer also generates fake files with random names. Before returning from TForm4.FormCreate, 6 loops will be executed:

XLS: 51 fake XLS files with random names (7 characters) are generated using a hardcoded stream of bytes.

EXE: 51 fake EXE files with random names (6 characters) are generated using a hardcoded stream of bytes.

DLL: 201 fake DLL files with random names (9 characters) are generated using a hardcoded stream of bytes.

TXT: 51 fake TXT files with random names (4 characters) are generated using a hardcoded stream of bytes.

XML: 51 fake XML files with random names (8 characters) are generated using a hardcoded stream of bytes.

HTM: 51 fake HTM files with random names (8 characters) are generated using a hardcoded stream of bytes.

Keylogger analysis:

As mentioned before, the keylogger setup is done in the TForm4.FormCreate. It uses “SetWindowsHookEx” with the “WH_KEYBOARD_LL” Id_hook to intercept keystrokes.

The hook function is rather rudimentary. For instance, it uses the GetAsyncKeyState, with the “VK_BACK” to find out if the victim used backspace.

```
                        ; CODE XREF: Keylogger+271B↓j
movzx   esi, bl
push    esi             ; vKey
call    user32_GetAsyncKeyState
cmp     ax, 8001h
jnz     loc_4D5C4B      ; jumptable 004D358F default case
movzx   eax, bl
add     eax, 0FFFFFFF8h
cmp     eax, 0D6h       ; switch 215 cases
ja      loc_4D5C4B      ; jumptable 004D358F default case
movzx   eax, byte ptr ds:Array_VK[eax]
jmp     dword ptr ds:PRINT_KEY_TYPED_CASE[eax*4] ; switch jump
```

For each typed key, there is a handler to save which key was typed in the keylogger buffer “poki65_pik_log”:

```
                        ; DATA XREF: Keylogger:PRINT_KEY_TYPED_CASE↑o
 mov     eax, offset poki65_pik_log ; jumptable 004D358F case 36
 mov     edx, offset aPrintScreen ; "[Print Screen]"
 call    @UStrCat
 jmp     loc_4D5C4B      ; jumptable 004D358F default case
-------------------------------------------------------------

                        ; CODE XREF: Keylogger+5B↑j
                        ; DATA XREF: Keylogger:PRINT_KEY_TYPED_CASE↑o
 mov     eax, offset poki65_pik_log ; jumptable 004D358F case 37
 mov     edx, offset aInsert_24 ; "[Insert]"
 call    @UStrCat
 jmp     loc_4D5C4B      ; jumptable 004D358F default case
-------------------------------------------------------------

                        ; CODE XREF: Keylogger+5B↑j
                        ; DATA XREF: Keylogger:PRINT_KEY_TYPED_CASE↑o
 mov     eax, offset poki65_pik_log ; jumptable 004D358F case 38
 mov     edx, offset aDel ; "[Del]"
 call    @UStrCat
 jmp     loc_4D5C4B      ; jumptable 004D358F default case
```

It comes as no real surprise that the keylogger is very basic and makes no use of any advanced technologies.

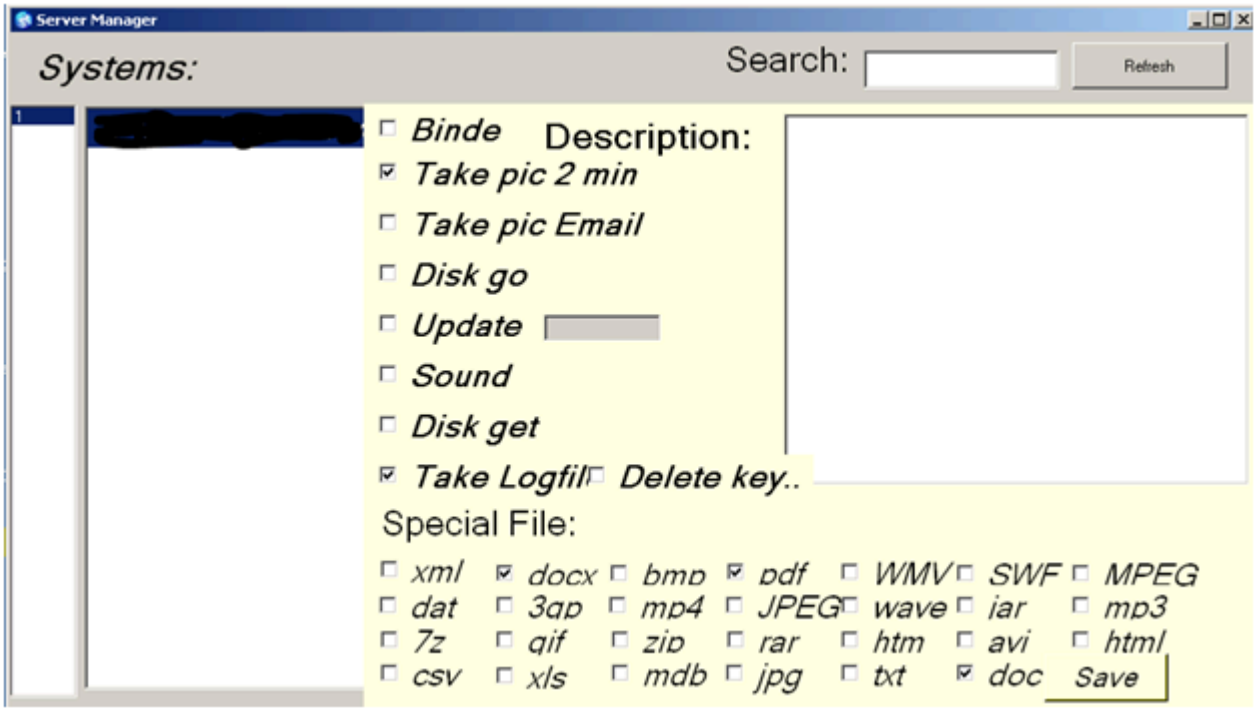
The malware uses 52 timers. Therefore, we will group them by actions, in order to make the overall analysis easier to follow.

Command and control: Protocol

We are now going to cover all the timers responsible for contacting the C&C server and receiving commands to execute on the infected machine, and all the various handlers used to execute actions according to those orders.

Note: In many routines, Madi creates ".bat" files in order to ping the C&C server to see if it is up or not and saves the result in a special file. Each file has a different name. If these files are referenced, we will provide the timer number responsible for its creation.

The server manager looks like this:



The GUI was probably rushed, but it serves its purpose. It can be used to create specific tasks for victims. See Timer 12 to see how each command is handled by the infostealer.

Timer 1: Check-in

Interval: 25 seconds

Before receiving commands, the infostealer connects to the C&C to a special page. I call it the check-in routine. Here is the description:

Timer 1 gets the ApplicationName and concatenates it with ".pkklm" (See Timer 15 description for details on how this file is created). It tries to open that file, looking for the "Reply From" string (when the IP responds to a ping). If it's not found, it disables Timer 1 and returns.

If present, the last part of the BOTID is generated using the “C:” Volume Serial Number.

Basically, the API function GetVolumeInformationW is called to get the Volume Serial Number, which is then concatenated to the BOTID_TMP generated in the TForm4.FormCreate. Now that the final BOTID has been generated, the final URL that is visited is generated as follows:

BOTID|COMPUTERNAME|VolumeSerialNumber/dastor/file.htm
e.g.: abaanu5|MYCOMPUTER-8712422C|6D8704FE/dastor/file.htm

The final URL is visited using Internet Explorer (IE) instrumentation.
(e.g.: http://C&C/abaanu5MYCOMPUTER-8712422C6C7704EF/dastor/file.htm)

Once visited, it enables Timer 18, disables Timer 1 and returns.

This is the checking-in process, which can tell the attackers when a victim computer is ready to receive commands.

Once the attackers have decided to send commands to the infected computer, a “das.htm” will be available in the “/dastor/” folder.

Timer 16: Visit commands page

Interval: 25 seconds

Timer 1 gets the ApplicationName and concatenates it with “.pkxm” (ping results from Timer 11). It tries to open that file, looking for the “Reply From” string (when the IP responds to a ping).

If it’s not found, it disables Timer 16 and returns.

The Final BOTID is computed (see Timer 1 description) to build the URL that is visited in order to receive commands. Before visiting that URL, the “dast.xls” file is deleted (see Timer 17 below).

The URL is visited using IE instrumentation. Timer 17 is enabled, and Timer 16 disabled.

Timer 17: Save the command page as “dast.xls”

Interval: 20 seconds

Note: During the execution of the Madi infostealer, many instances of IE are running.

Timer 17 will go through all the different instances of instrumented IE, looking for pages with “dastor” in their title. Once found, the content of the page (without the title) is saved as “dast.xls”.

If nothing is found, it will go to next IE instance, and repeat the checks until no instances are left. If nothing is found, a clean-up routine is launched.

At the end of the Timer 17, it looks for ” – dastor – Windows Internet Explorer” and different variants (Internet Explorer) and sends a “WM_Close” Message using the “PostMessageW” function in order to close the page.

Among all those captions, it also looks for " – 404 – File or directory not found" and variants of 404 pages, if the page wasn't found.

Once the clean-up is completed, Timer 17 disables itself and returns.
At this point, we have a local file with the commands to execute on the infected machine.

Timer 12: Command dispatcher

This timer is responsible for parsing the command file. In order to make the description a little easier to follow, here is a sample command file:

```
deskpikdiskgosounddiskget$.$*xml*$.$*docx*$.$*bmp*$.$*pdf
*$.$*wmv*$.$*swf*$.$*mpeg*$.$*dat*$.$*3gp*$.$*mp4*$.$
*jpeg*$.$*wave*$.$*jar*$.$*mp3*$.$*7z*$.$*gif*$.$*zip*$.$
*rar*$.$*htm*$.$*avi*$.$*html*$.$*csv*$.$*xls*$.$*mdb*$.$
*jpg*$.$*txt*$.$*doc*
```

When executed, Timer 12 is disabled.

The infostealer Trojan then checks if the file "dast.xls" is present (created by Timer 17, see above).

If it's not present, Timer 12 is re-enabled and returns.

The next stage of the process opens "dast.xls" which searches for commands to execute (see the command file above). Lots of commands can be sent simultaneously, meaning Timer 12 will not stop parsing when one command is found. Here is the full logic of the parsing:

PIK:
If the command file contains the word "pik", it checks if the status of Timer 3 is enabled. (Timer 3 is a webmail, social network and IM screen capture routine.)
If not enabled, Timer 3 is enabled, and screen monitoring begins. Command parsing continues. If the "pik" command is not found, Timer3 is disabled.

DESK:
If the command file contains the word "desk", it checks if the status of Timer 13 is enabled. (Timer 13 is a screen capture routine.)
If not enabled, Timer 13 is enabled, and screen monitoring begins. Command parsing continues. If the "desk" command is not found, Timer13 is disabled.

SOUND:
If the command file contains the word "sound", it checks if the status of Timer 14 is enabled. (Timer 14 is a sound recording routine.)
If not enabled, Timer 14 is enabled, and sound recording begins. Command parsing continues. If the "sound" command is not found, Timer14 is disabled.

If the command file contains the word "newfi", nothing happens. This is probably a leftover from older code.

UPDATE:

If the command file contains the word “update”, it checks to see if it also contains a version number, which must be different from current version (“1.1.6” in the analyzed sample). If neither of those two conditions are valid, it goes to the next command parsing. The checking routine is very simplistic and assumes that the version number will be higher, not lower. It is therefore possible to downgrade the Trojan.

If the required update criteria are met, it will create “Update.dll”. (Update.dll is made from a hardcoded stream of bytes and isn’t a valid DLL.)

The Trojan now locates the “STARTUP” folder where a copy of the “UpdateOffice.exe” (Trojan downloader) is found, and executes it using ShellExecute. (In the first part of the article, we explained how the downloader downloads and installs the infostealer.)

The Trojan downloader is necessary in order for updates to occur. If the downloader has been deleted for some reason, the update won’t be performed.

Once executed, Timer 12 terminates its execution, as the infostealer executable (iexplore.exe) will be overwritten by the Trojan downloader with a newer version and executed.

DELETE:

If the command file contains the word “delete”, it will create “delete.dll”, using exactly the same stream of bytes that is used in “update.dll”.

The Trojan now locates the “STARTUP” folder where a copy of the “UpdateOffice.exe” downloader is located, and deletes it. Once deleted, it then proceeds to terminate itself.

At this point, upon the next reboot, the infection isn’t restarted.

Note: The infostealer doesn’t restart by itself, allowing an automatic update every time the computer reboots.

On the other hand all the other downloader files (non-malicious) are still present in the /printhood/ folder. The full folder of the infostealer is still present, as is the malware.

BIND:
If neither “update” nor “delete” are found, Timer 12 checks if the command file contains the word “bind” and creates “nrbindek.dll” using exactly the same stream of bytes that is used in “update.dll”.

Nothing else happens at this point. However, as we have seen in the Form creation, upon execution, the malware checks whether “nrbindek.dll” is present.

If it is present, Timer 12 will enable Timer 28.

If “bind” isn’t found, the parsing continues with the next command.

DISKGO:
If the command file contains the word “diskgo”, it will create “lbdiskgo.dll”, using exactly the same stream of

bytes that is used in “update.dll”. Parsing continues with next command.

Note: “lbdiskgo.dll” is checked by Timer 42 and Timer 43.

DISKGET:
If the command file contains the word “diskget”, it will create “fskdiskget.dll”, using exactly the same stream of bytes that is used in “update.dll” and enable Timer 23.

Timer 12 then checks whether “specialfile.dll” is present. If NOT, it will look for the file extensions included in the command that was received. The attackers select from a list of 27 extensions that are provided by the C&C server, and which can be selected using a Remote Control Tool (see at the beginning of the Timer 12 description to view the extensions listed in the sample command file).
Each file extension is separated by a special marker “$.$”.

Timer 12 searches for the “$.$” marker. If it’s not present, the parsing stops there.

If the marker is present, it saves those extensions to the “specialfile.dll” and enables Timer 26.

Note: Specialfile.dll is therefore used to tell the malware what file extensions to look for and Timer 26 will handle diskget.

Afterwards, or if specialfile.dll was already present, it will check whether the “logfi.dll”
is present, and stop parsing commands if it is not.

If the file is present, it looks in the command buffer for the word “file” and exits the commands parsing if not found.

If logfil.dll is present, it will search files on fixed hard drives and remote drives. The authors’ poor programming skills are quite noticeable in this part of the code.

It is also interesting to note that it will search for “MHTML” files, even if that option isn’t available in the Server Control tool, and that they made a duplicate entry in the hardcode list of files that need to be to located (htm is present twice).

File types searched:

*.*txt/*.*jpg/*.*doc/*.*pdf/*.*bmp/*.*docx/*.*mdb/*.*xls/*.*csv/*.*html/*.*avi/
*.*mp3/*.*wave/*.*htm/*.*rar/*.*zip/*.*htm
(again?!)/*.*gif/*.*7z/*.*jar/*.*JPEG/*.*mp4/*.*3gp/
*.*dat/*.*MPEG/*.*SWF/*.*WMV/*.*xml/*.*MHTML/

Total of 29 extensions, with one duplicate. 27 extensions are present in the Server Control tool and one that is not (MHTML).

It saves the log file as “logfi.dll” for each hard drive and creates a backup as “logfi.dll.BMH”. It will overwrite the logs for each iteration of the loop.

It only search files on remote and fixed drives, not on USB/external drives; that’s for the logging part.

Once the Parsing is complete, Timer 12 re-enables itself and exits.

Monitoring

Timer 3: PIK handler – Webmail, social network and IM screen capture

Interval: 60 seconds.

Timer 3 creates a “begirnagir.htp” file.

It then checks whether the user has been surfing or using the following applications and takes a screen capture if found:

gmail, hotmail, yahoo! mail, google+, msn messenger, blogger, massenger (?), profile,icq, paltalk, yahoo! messenger for the web, skype, facebook.

The screen captures are saved as a JPG using the following name convention: mm-dd-yyyy-hhnnss. The “Now” and “FormateDateTime” functions are used.

Timer 13: DESK handler – Screen capture

Interval: 3 minutes

Note: The GUI used to control the bot says 2 minutes, but the code doesn’t lie.

Timer 13 takes screen captures every 3 minutes. They are saved using the following name convention: mm-dd-yyyy-hhnnss. The “Now” and “FormateDateTime” functions are used.

The files are in JPG format.

Timer 14: SOUND handler – Recording sound

This timer is responsible for starting the audio recording using the mci* functions from winmm.dll.

The following commands are used: “OPEN NEW TYPE WAVEAUDIO ALIAS mysound”, “SET mysound TIME FORMAT MS BITSPERSAMPLE 8 CHANNELS 1 SAMPLESPERSEC 8000 BYTESPERSEC 8000” and finally “RECORD mysound”.
Once the commands are sent, Timer 30 is enabled, and Timer 14 returns.

Timer 30: Started by Timer 14 (sound command handler)

Interval: 60 seconds

This timer does anything apart from start Timer 31 when it is time to save the recoded audio.

Timer 31: Started by Timer 30 (when it is time to save audio recordings)

When sufficient time has passed since the start of audio recording, Timer 31 disables Timer 30, stops the recording by sending the following command: “STOP mysound”.

To save audio files, it sends the “SAVE mysound” command. The files are saved using the following name convention: mm-dd-yyyy-hhnnss. The “Now” and “FormateDateTime” functions are used.

The final file is saved as .wav.BMH.

Timer 31 is then disabled, and Timer 14 (Sound handler) is re-nabled for the next audio recording.

Timer 32: Set up keylogger

Interval: 60 seconds

Even though the keylogger setup is performed when the application starts, in the FormCreate routine Timer 32 sets up the keylogger every 60 seconds. The details of the keylogger have already been described earlier in this document.

Timer 2: Creation of keylogger logs

Interval: 10 seconds

Timer 2 starts by getting the current user name (GetUserName API Function), and then checks if the “poki65.pik” file is present. This file is the current ongoing keylogging file. If it’s not present, it looks for “solt.html”, which indicates whether the keylogger has created its first log yet.

If none of those files are present, it means it is the first time the keylogger has started logging.

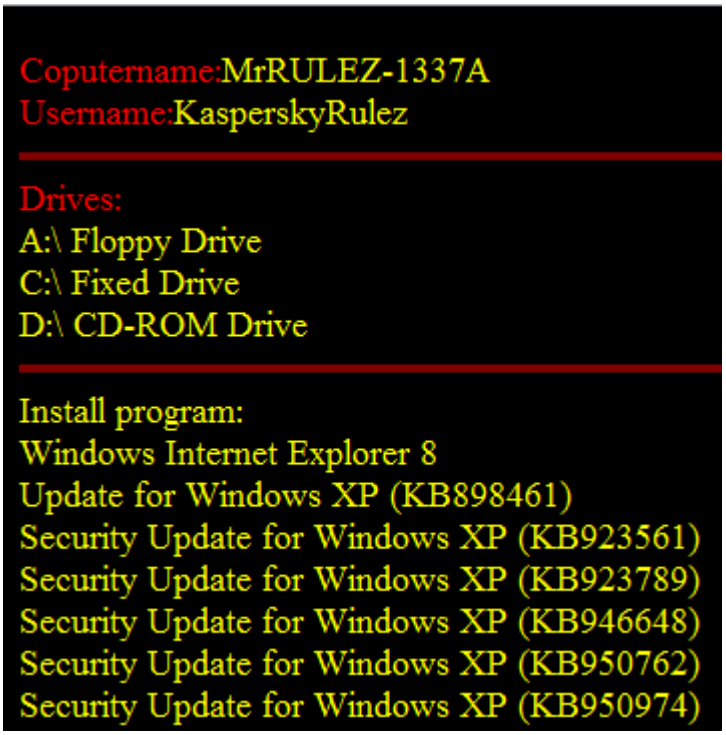
The first log file is different from subsequent log files, as it contains more information. The Madi keylogger files use HTML tags and colors to make them easier to read.

For the first log, it executes “cmd.exe /c ipconfig /allcompartments > ipconfig.txt”

It waits 5 seconds and appends the content of “ipconfig.txt” to the HTML content that is created.

The computer name as well as the current user name is appended to the log, followed by the list of available drives: Floppy Drive, Fixed Drive, Network Drive, CD-Rom Drive and RAM Disk.

Finally, a full list of installed software, including security patches, is appended to the log file,
as can be seen on the screenshot below:



Once this part is completed, it creates a file called “solt.htm” containing the word “wertik”.

It will continue formatting the poki65 log file. At the very beginning you can see the “Content-Language” set to “fa”, which is Persian.

```
dw 3Ch
unicode 0, <body bgcolor="#000000">
dw 3Eh, 3Ch
unicode 0, <font size="4">
dw 3Eh, 3Ch
unicode 0, <font color="sky blue">
dw 3Eh, 3Ch
unicode 0, <META HTTP-EQUIV="Content-Language" CONTENT="fa">
dw 3Eh, 3Ch
```

This is how the keylogger logs are generated.

Timer 4: Insert time stamps and tags to display screen captures into keylogger logs.

Interval: 1 millisecond

Timer 4 is responsible for inserting IMG tags inside the keylogger log. It is also responsible for adding the time stamp taken from the C&C server (see Miscellaneous section, Timer 7 and 8).

Timer 6: Backup keylogger log for exfiltration

Timer 6 searches for the poki65.pik file – the current log session. If not found, it returns.

It then looks for the size of the log file. If it is lower than 15 KB, it will return.
Only log files bigger than 15 KB are exfiltrated. If the size criteria is met, they are copied using the following name convention: mm-dd-yyyy-hhnnss.HTM. Timer 6 then deletes “poki65.pik” and returns.

Note: A new log will be created by Timer 2 (solt.html tells the keylogger not to list drives, installed software etc. again).

DATA STEALING

Data stealing is handled by several timers. Each type of stolen data is stored in a special folder in the server. Files exfiltrated to the C&C servers are Base64 encoded.

BIND:

Timer 28: Started during Form Creation (related to the BIND command)

Note: When the infostealer starts, Timer 28 is enabled if the file “nrbindek.dll” is present (created by the BIND command).

Timer 28 searches for “*.*exe” files on all fixed hard drives. For each *EXE* file found that doesn’t belong to the “Windows”, “Program Files” or “Program Files (x86)” folders, an entry (full path to *EXE* file) is added to the file filebind.xls.

Once the hard drives have been scanned, Timer28 returns.

Filebind.xls therefore contains all the executables on the fixed hard drives, except from those in Windows and Program Files.

Timer 29: Started during Form Creation (related to the BIND command)

Note: The code of this timer is some of the worst that is used in the infostealer. The programming, obfuscated with Delphi, is very bad.

Timer 28 generates a list of *EXE* files that don’t belong to the “Windows”, “Program Files” or “Program Files (x86)” folders. For each entry of that file, Timer 29 will make a backup of the executables. The *.bind* extension is appended to their original name.

Many files are used to monitor the exfiltration status of the executables.

However, Timer 29 doesn’t actually seem to exfiltrate anything, probably because of bugs.

Timer 9: Check for files ready to be uploaded

Interval: 5 seconds

Timer 9 is disabled. If either Timer 19 or Timer 20 is enabled, it means there is already an active exfiltrating task. Timer 9 is enabled and it returns.

Otherwise, Timer 9 searches for files *.*KILOP as well as *.htm.BMH* files in the malware directory. KILOP files are Base64 encoded versions of files to exfiltrate. If no file is found, Timer 9 is enabled and returned.

If files are present, they are ready to be exfiltrated, and Timer 19 is enabled.
Before returning, Timer 9 is enabled.

Timer 19: Check if IE instrumentation has been used to visit the upload page.

Interval: 25 seconds

Timer 19 searches for a specific page title:

If the page title “new title hastam – Microsoft Internet Explorer” is found, Timer 19 returns.

“OKshodiha – Windows Internet Explorer” means a file is ready to be uploaded – Timer 20 is enabled and returns.

If none of those captions are found, Timer 19 starts IE_Instrumentation and visits the Sendfilejj.html page, enables Timer 20, then returns.

Timer 20: File upload

Timer 20 searches for *.*KILOP files, computes the BOTID (see Timer 1 for details), and fills the POST parameters. The “S0”, “S1” and “S2” forms present in the Sendfilejj.html are “filled” and the file is uploaded using IE Instrumentation.

T3, is the BOTID+Folder used for uploading (see below)

T2 is the file name

T1 is the Base64 encoded content of the file.

To compute T3, the following folder is appended to the BOTID (each victim has a root folder named after the BOTID on the C&C).

“/Pi/” for .jpg.BMH – Screen Captures

“Te/ for .htm.BMH – Keylogger logs

“/So/ for .wav.BMH – Audio Recordings

“/Fi/” for important.file.BMH

/Fi/CoolDisk/” for .fildik.BMH (data stolen from removable drives)

Files are sent via the Sendfilejj.html page hosted on the C&C, which is a wrapper for the “sik.php” script used to receive exfiltrated data.

Timer 5: Base64 encoder for exfiltrated data

Interval: 1 millisecond

When triggered, it disables Timer 5, searches for *.*BMH files (files that will be exfiltrated once Base64 encoded) in the malware folder. When one file is found, it checks if the file is indeed on the disk and accessible. It Base64 encodes it and saves it as nameoffile.BMH.KILOP. The non encoded version (BMH) is deleted, Timer 5 is re-enabled and it returns. Files are handled one by one, but the timer interval is very small, therefore it's almost instantaneous.

Note: The resulting encoded files are those handled by Timer 20 described above. The process occurs as follows: Timer 9 enables Timer 19, which enables Timer 20 to upload files generated by Timer 5.

Timer 21: Filesend.xls parser

Filesend.xls has a list of files to exfiltrate.

Upon execution, Timer 21 is disabled. If "filesend.xls" is present, it is opened and read.

All the files to be exfiltrated are separated by the "*" character as in the example below:

*C:Documents and Settings%USER%

Desktoptoolsstealme.txt**C:Documents and Settings%USER%

Desktoptoolsstealme2.txt*

Timer 21 parses each entry, and will check whether the file exists. If it does, a copy of the file will be made in the malware directory with a .file.BMH extension. (In my example, we have: "stealme.txt.file.BMH".)

Timer 10: Tracking what was uploaded and cleaning IE instrumentation pages

When a file has been uploaded using Timer 20, a POST is made to the sik.php file, a page is returned containing the name of the uploaded file, as well as the hardcoded string "Save Shode" as you can see on the screen capture below:

```
</head>
<title>Save shode</title>
<html>

msg_25.txt.file.WTF.KILOP<body>
```

Timer 10 is responsible for keeping track of some of the uploaded files. Exfiltrated files are added to the "rafteha.zip", which lists the files that have already been handled. The last file path to be handled is saved to the "fileomade.xls" file.

Timer 15: Check for "filesend.xls"

Timer 15 is disabled upon execution and "filesend.xls" is sought. If present, Timer 15 is enabled and it returns.

If not, it checks whether Timer 1 is enabled. If Timer 1 is enabled, it enables Timer 15 and returns.

If Timer 1 isn't enabled, Timer 15 checks the status of Timer 18. If it is enabled, Timer 15 re-enables itself and returns.

If "filesend.xls" isn't present and both Timer 1 and Timer 18 are disabled, it creates a "pangtkp.bat" file, which contain "ping C&C_IP >C:DOCUME~1%USER%TEMPLA~1iexplore.exe.pkklm".

That bat is executed, and both Timer 1 and 5 are enabled before returning.

There are other timers that are in some way or other related to exfiltration and data stealing, but they are all fairly similar. There is a lot of redundancy in the malware.

Timer 23: List all removable drives on the machine

Timer 23 lists all the removable drives on the machine, enables Timer 24, Timer 23 disables itself and returns.

Timer 24: Search and copy files from removable drives

Timer 24 receives the list of removable drives computed by Timer 23, and searches all the files on the devices.

Stolen files will be copied to the malware directory with fildik.BMH extensions, which will later be encoded as fildik.BMH.KILOP (Base64) and exfiltrated.

The list of processed files are stored inside raftehacool.zip.

Miscellaneous

The infostealer contains 52 timers. Some of them do not perform any important tasks. The authors decided to ping the C&C server and save the results under specific file names. Those files are checked and parsed in order to find out if the C&C is up and if certain actions can be taken. This is pretty amateurish programming.

Timer 44: simple ping via pangtipo.bat

Timer 44 is disabled upon execution. Timer 44 checks whether Timer 45 is enabled and returns if it is. (Timer 44 is enabled prior to returning.)
If Timer 45 is disabled, a "pangtipo.bat" file is created, which contains "ping C&C_IP >C:DOCUME~1%USER%TEMPLA~1iexplore.exe.pkxml".

The bat file is executed, Timer 44 is enabled and Timer 44 returns.

Timer 11: Simple ping from pangtip.bat

Timer 11 is disabled upon execution.If Timer 16 is already enabled, Timer 11 re-enables itself and returns. If Timer 17 is already enabled, Timer 11 re-enables itself and returns.

If none of the timers are enabled, it creates the "pangtip.bat" file, which contains "ping C&C_IP >C:DOCUME~1%USER%TEMPLA~1iexplore.exe.pkxm" and executes it via ShellExecute.

Timer 16 is enabled and returns.

Note: Timers 1, 7, 11, 15, 44 and 48 generate these batch files under different names and the results are saved under different names too.

Timer 7: Was “timeip.php” visited?

Timer 7 is disabled upon execution. Timer 7 checks whether the timeip.php page was visited. If not, it visits the page using IE instrumentation, Timer 7 disables itself and enables Timer 8 (see description below).

It creates the “pangip.bat” file, which contain “ping C&C_IP”. Results are saved as “iexplore.exe.pkam”.

Note: the file name used to save the output of the ping commands is based on the infostealer executable name, which is “iexplore.exe”. If the executable is renamed, the log files will have different names.

Timer 8: Parse the results of the timeip.php visit

The timeip.php script returns the current time and the IP address of the victim. The results of the visit (done with IE instrumentation in Timer 7) are saved into a buffer which is used during the keylogger log creation (see Timer 4 description).

Timer 22: Ensure there is a backup copy of UpdateOffice (downloader)

Note: The downloader is the only malware that starts after Windows boots. It’s therefore important to ensure various backup copies are made.

Timer 22 checks if “UpdateOffice.exe” is present in the infostealer directory (templates). It shouldn’t be, as it is only present in the printhood directory. (See Downloader description at the beginning of the article.)

Since it is not present, it calls a subroutine to get the path to the “Printhood” directory (GetSpecialFolderLocation with CSIDL_PRINTHOOD parameter). While concatenating the “UpdateOffice.exe” and the “Printhood” folder, the “” character is missing, and therefore, the routine is bugged. The returned string is: “C:Documents and Settings%USER%PrintHoodUpdateOffice.exe” instead
of “C:Documents and Settings%USER%PrintHoodUpdateOffice.exe”.

It then copies (or at least tries to, as the path is wrong) “C:Documents and Settings%USER%PrintHoodUpdateOffice.exe” as “srAntiq.dll” in the Templates folder.

If “OfficeUpdate.exe” isn’t present in the “printhood”, a copy is made from “srAntiq.dll”.

It retrieves the path to the Startup Folder using the CSIDL_STARTUP: “C:Documents and Settings%USER%Start MenuProgramsStartup”.

Timer 22 checks whether “OfficeUpdate.exe” is present in that folder; if not, it will make a copy of srAntiq.dll to the Startup folder and returns from Timer 22.

Timer 25: Check for “fsdiskget.dll”

Timer 25 checks if “fsdiskget.dll” is present in the malware directory; if not, it returns.

If the file is present, it enables Timer 23 (see the Data Stealing section for a description).

Timer 42: lbdiskgo.dll, soltanik.dll and res.exe checking

Timer 42 checks whether a flag (set by Timer 34 and cleared by Timer 33) is set to 0 and if lbdiskgo.dll, soltanik.exe and res.exe are present. If they are, it enables Timer 33; otherwise, it returns.

Timer 43: lbdiskgo.dll / ladine.dll / res.exe checking

Timer 43 returns directly if neither “lbdiskgo.dll” or “ladine.dll” are present. If res.exe is present, it enables Timer 44 and Timer 48; otherwise, it returns.

Timer 45: Visit the ReReReRe.htm page

Timer 45 deletes pangtipo.bat, reads iexplore.pkxml to make sure the C&C replied. (Timer 1 and Timer 16 provides some more details on the use of such .bat files.)

Uses FindWindow to check whether IE Instrumentation has been used to visit the special ReReReRe.html page, which contains the following title: “r!r!r!r!”.

It looks for different variants such as “r!r!r!r! – Windows Internet Explorer” or “r!r!r!r! – Microsoft Internet Explorer”.

If one of them is found, it means the page was visited using IE instrumentation. It disables Timer 45 and returns.

If none of them are found, Timer 45 will visit the URL http://C&CIP/ASLK//asgari/mah/ReReReRe.htm, enable Timer 46 (see below), disable Timer 45 and return.

Timer 46: Parse “ReReReRe.htm” (downloaded by Timer 45)

Timer 46 goes through all the different running instances of instrumented IE, looking at the title of each HTM page. The main interest here is “r!r!r!r!”.

```
<title>r!r!r!r!</title><!doctype html><html><head><meta htt
 including webpages, images, videos and more. Google has ma
.google={kEI:"PrX1TrTdCImPswaOyYDnBg",getEI:function(a){var
.location.protocol=="https:"},kEXPI:"17259,23756,24692,2487
,34680,34901,35211,35268,35300",ei:"PrX1TrTdCImPswaOyYDnBg"
```

This page is the ReReReRe.htm file downloaded by Timer 45. Timer 46 looks for a special EOF (End Of File) marker: “tamamshodfile”. This marker is used by the infostealer to make sure the htm page was fully downloaded.

Once the page has been confirmed as valid, it looks for the textarea id S1 which holds double Base64 encoded PE Files.

```
<body>
<meta http-equiv="Content-Type" content="text/html; chars


<form name="f" onSubmit="validateForm(f.T1,f.T2,f.T3)"  m

<textarea id="S1" name="T2" style="display:none" >VFZwUUF
GhrSkJVYUdseklIQnliMmR5WVcwZ2JYVnpkQ0JpWlNCeWRXNGdkVzVrWl
QUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQ
BQU1RQ0FBQUFBQUF3dWdvQUFCQUFBQURBQ2dBQUFFQUFBQkFBQUFBQ0FB
E0QXNBQU9nQkFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQVFBc0FjSk1BQUF
UFBQUFBQUFBUTA5RVJRQUFBQUJrcXdvQUFCQUFBQUNzQ2dBQUJBQUFBQU
QUFET0NnQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFEQUxtbGtZWFJoQUFBeUp3Q
kR0VBQUJnQUFBQUFNQXNBQUFJQUFBRDJDZ0FBQUFBQUFBQUFBQUFBQUFC
FBQUFBQUFBQUVBQUFGQXVkSEZ1QndBQUFBQVFBQUFBMEEEwQUFBWUFBQUI
UFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQU
QVFRUUFBREIwSnZiMnhsWVc0QkFBQUFBQUVBQUFBQUVFQUFCVVpoYkhOb
CQkFBQUVFVjI5eVpBTUFBQUFBLy84QUFKQ0FFRUFBQ2daVGRISnBibWVN
```

The Base64 encoded data is saved as: ASLASLKK223.dll.

Timer 47: Double decoding of Base64 encoded payload from ReReReRe.htm

Note: Timer 46 saves the payload as ASLASLKK223.dll.

Since the payload file is double encoded, the decoding is performed in two steps:

- ASLASLKK223.dll is decoded to ASLASLKK224.dll to get a single encoded Base64 file.
- ASLASLKK224.dll is decoded to “res.exe” : Final PE file.

Res.exe is a copy of the Resource Hacker utility. ASLASLKK224.dll is deleted.
The use of Res.exe is described in the analysis of Timer 39 below.

Once Timer 47 has finished enumerating all the IE instances, it will call a cleaning routine. It searches for ” – r!r!r!r! – Windows Internet Explorer” and different variants described in Timer 45 and sends a “WM_Close” Message to IE Windows in order to close them.

Among all those captions, it also searches for ” – 404 – File or directory not found.” and variants of 404 pages.

Once the cleaning is completed, Timer 47 disables itself and returns.

Timer 49: Visit the SeSeSeSe.htm page

Timer 49 is almost identical to Timer 45. The only difference is the page visited: SeSeSeSe.htm instead of ReReReRe.htm

See the Timer 45 description for details.

Timer 50: Parse “SeSeSeSe.htm” (downloaded by Timer 49)

Timer 50 is almost identical to Timer 46. The only difference is the page parsed: SeSeSeSe.htm instead of ReReReRe.htm and local file names. The double encoded payload is saved as “ASLASLKK2231.dll”.

See the Timer 46 description for details.

Timer 51: Double decoding of Base64 encoded payload from SeSeSeSe.htm

Note: Timer 50 saves the payload as ASLASLKK2231.dll.

Since the payload file is double encoded, the decoding is performed in two steps:

- ASLASLKK2231.dll is decoded to ASLASLKK2241.dll to get a single encoded Base64 file.
- ASLASLKK2241.dll is decoded to “Ladine.dll”: final PE file.

Note: At the time of writing, the SeSeSeSe.htm page had been removed from the C&C server.

A C&C server used by older variants of the infostealer is still available and the old page name was “SSSS.htm”. The embedded file is a template of a downloader executable (see Timers 35, 36, 37, 38 and 39 for further information).

Once Timer 51 has finished enumerating all the IE instances, it will call a cleaning routine. It searches for ” – s!s!s!s! – Windows Internet Explorer” and different variants described in Timer 45, and sends a “WM_Close” Message to IE Windows in order to close them.

Among all those captions, it also searches for ” – 404 – File or directory not found.” and variants of 404 pages.

Once the cleaning is completed, Timer 51 disables itself and returns.

BETA/NON-WORKING FEATURES: New executable generation

There are a few timers in the infostealer that are related to a missing file. I managed to find a copy of the missing file from an older command and control server, in order to understand the intentions of the authors.

The missing file is downloaded by Timer 50: SeSeSeSe.htm. It’s not present on the current C&C servers.

If we were to replace the SeSeSeSe.htm with an old copy (originally SSSS.htm), Timer 51 would produce a file called “Ladine.dll”, which is a template executable of the Trojan downloader used to install the infostealer.

Timer 52: Copy Ladine.dll to “Soltanik.exe”

Timer 52 makes a copy of “Ladine.dll” under the name “soltanik.exe”, which is the template file.

Timer 35: Clean files from Timer 39

Timer 35 is disabled. A special BOTID is created by concatenating “CoolDiskGo(” with “BOTID_TMP)”, e.g.: CoolDiskGo(MYCOMPUTER-8712422C6C7704EF)

Timer 35 puts the C&C IP address in a global variable that will be used by Timer 38.

Timer 35 tries to delete the following several files created by Timer 39: 1.txt, res.ini, res.log, Icon_1.ico,output.rc and server.exe.

It does several other things which are not relevant to what I describe here, so I've omitted any reference to those actions.

Timer 36 is enabled before returning.

Timer 36: Enable Timer 37 if 1.txt isn't found – logic/code bug

Timer 36 is disabled upon execution. If "1.txt" isn't present, Timer 37 is enabled. Otherwise, it calls a Base64 decoding function. 1.txt must be a valid Base64 encoded stream of bytes; otherwise, an exception occurs and Timer 36 returns.

Timer 37: Update resource for the template downloader: Soltanik.exe

Timer 37 is disabled upon execution. A structure exception handler is installed before reading 1.txt.

In the event of an exception the SEH handler will enable Timers 42, 34, 33, 35, and Timer 37 will return.

Timer 37 expects 1.txt to be present and here is the logic bug. Timer 37 is only enabled when 1.txt isn't present, by Timer 36. Let's ignore the reasons for its creation and continue analyzing the intentions of the authors.

Timer 37 calls the BeginUpdateResource (with the bDeleteExistingResources parameter set to 0), and start updating the resources of the template executable (soltanik.exe) in RCDATA.

A MAHDI entry is added, with the Base64 content from 1.txt. This works in exactly the same way as the downloaders with social engineering features.

Timer 38 is enabled, and Timer 37 returns.

Note: At the end of Timer 37, Soltanik has been modified to have a new resource: MAHDI.

Timer 38: Update more resources from the template downloader (Soltanik.exe)

Several entries are added to RCDATA:

- Shelikn : Special BotID generated by Timer 35
- SiteW: C&C IP address
- Bind: Empty (according to analysis of the downloaders that use social engineering, it should be the extension of the embedded file dropped to social engineering victims. If MAHDI contains a Base64 encoded picture, Bind should be set to .JPG).
- Filee: SCR
- Roze: 0

Once the resources have been updated, Timer 39 is enabled and Timer 38 returns.

Timer 39: Generate a final binary: Server.exe with updated icon

At the end of Timer 38, soltanik.exe has been fully updated with new resources. Upon execution, Timer 39 disables itself and starts generating a special command line for the Resource Hacker tool that was created as "Res.exe" by Timer 47.

The command line is the following:

```
"-extract , C:\Documents and Settings\MALRE\Templates\\output.rc, ICONGROUP , , "
```

There is a bug in the routine. An executable name is missing right after "-extract".

The command line dumps the Main Icon to disk (Icon_1.ico) and creates a file called "output.rc".

At this point, it is impossible to know which file was meant to be used as the source of a new icon. For the sake of our analysis, let's pretend the bug doesn't exist and that a valid file name was provided.

Afterwards, a second command line is passed to "Res.exe":

```
"-addoverwrite soltanik.exe, Server.exe, Icon_1.ico, ICONGROUP, MainIcon, 0"
```

This final command line will generate Server.exe, a copy of soltanik.exe whose icon has been changed to the one extracted in the previous command.

Server.exe is now fully updated. Its resources are filled, and its icon changed. It's not clear why the authors did this, but despite all the bugs it was possible to understand the overall aim of the routine: to create a Server.exe file from soltanik.exe with a new icon and added resources. What happens to Server.exe? Nothing, this is a non-working feature. It appears Madi has the ability to generate new downloaders, at least, in theory.

The 7 remaining timers won't be described as they are of little interest.

Conclusions

In this article we closely analyzed the infostealer used in the Madi campaign. The coding style and the usage of Delphi, together with the programming techniques indicate a rudimentary approach.

Most of the data-stealing actions and communication with the C&C servers take place via external files, which is rather messy. Whoever coded it is probably still reading through the first chapters of their Delphi manuals.

This is maybe why it is surprising to note its effectiveness, considering the data received from the sinkhole. During the monitored period, a little over 800 victims were connected to the servers. All of them fell prey to the various social engineering techniques used by the malware.

To sum up, we can say the following:

- the components of the Madi campaign are surprisingly unsophisticated
- no exploits or advanced 0-day techniques are used anywhere in the malware
- despite that, the overall success of the campaign is surprisingly high

- nevertheless, we should remember that even low quality malware can steal data
- Madi was a low investment, high profit project
- its authors remain unknown

We will continue to monitor the Madi malware and update you on our findings in the future.
