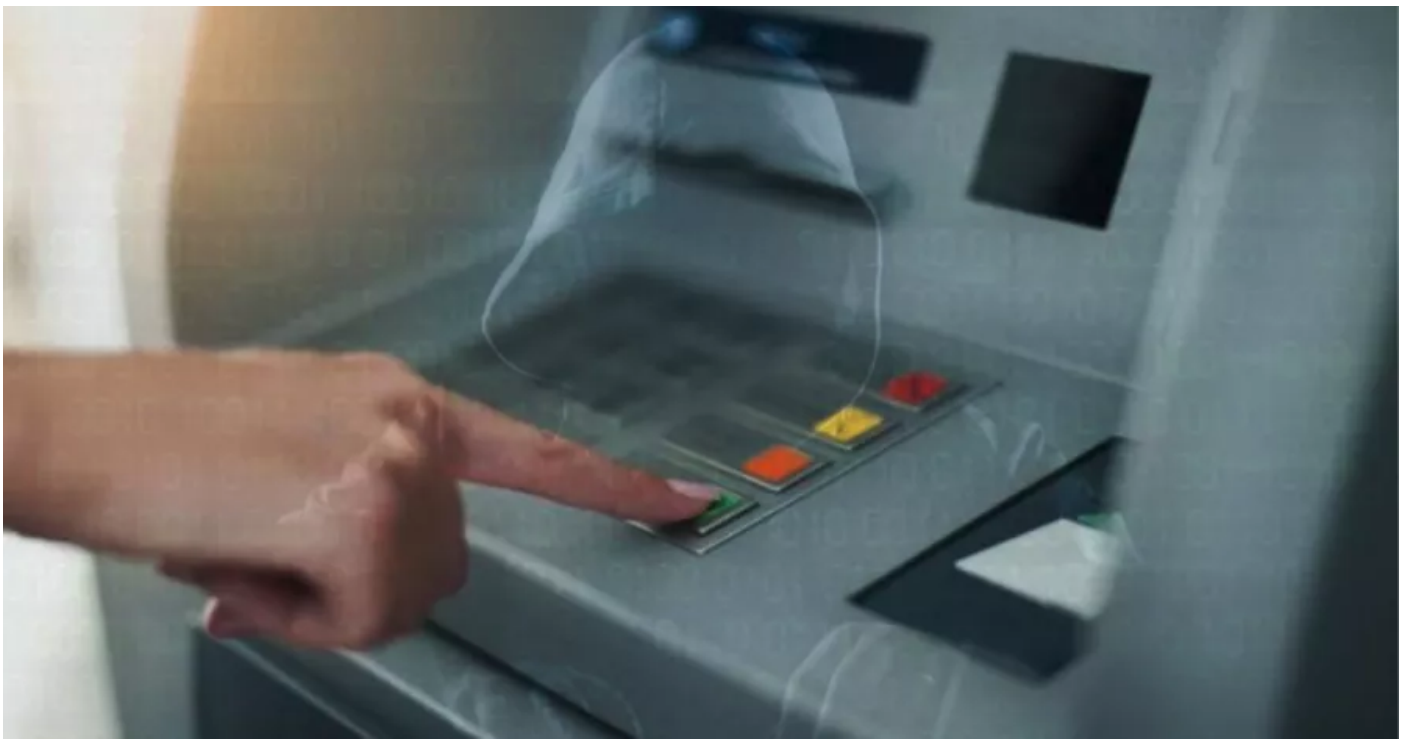
ATMitch: New Evidence Spotted In The Wild

blog.yoroi.company/research/atmitch-new-evidence-spotted-in-the-wild

ZLAB-YOROI May 7, 2019



Introduction

In the first days of April, our threat monitoring operations spotted a new interesting malware sample possibly active in the wild since 2017. Its initial triage suggests it may be part of an advanced attacker arsenal targeting the Banking sector, possibly related to the same APT group Kaspersky Lab tracked two years ago after the compromise of a Russian bank, where a particular malware tool dubbed ATMitch has been unveiled. In the past, this piece of malware was manually installed on the victim ATM after a wide enterprise network intrusion, enabling the cyber-criminals to manipulate the cash-withdrawal process on the machine.

The recent, unattended discovery of such kind of sample within the Info-Sec community led us to a deep dive into this particular malware tool, spearhead of a sophisticated cyber arsenal.

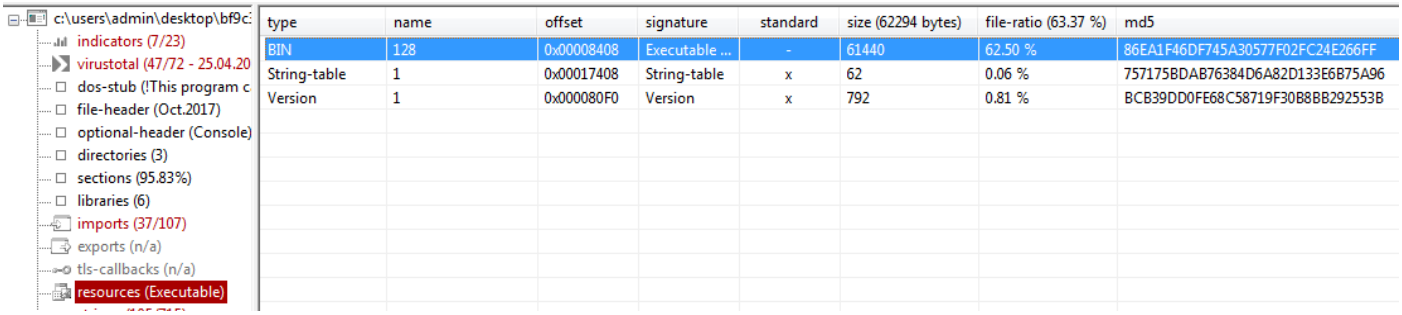
Technical Analysis

The executable sample is a PE32 x86 file named “tester.exe”. It seems to be custom loader for the real malicious payload able to take control of the target machine.

Sha256	bf9c35d8f33e2651d619fe22a2d55372dedd0855451d32f952ecfc73fa824092
Threat	ATMitch ATM malware
Brief description	ATMitch initial loader
Ssdeep	1536:sPNdY/P/r6aqTzN7gqJT/0vniPJiz3yUrvGkc+uylR:sPiz657gqJT/06xiT/vaVyl

Table 1: Information about Dropper/Loader of ATMitch

The static data about this sample reveal the sample has been compiled on *8th Oct 2017*, months later than the Kaspersky disclosure of the ATMitch attack operation. This element is not enough to date the sample with 100% accuracy due to possible tampering, anyway the other static details suggest the date could be genuine due to the absence of scrambled artifacts.



type	name	offset	signature	standard	size (62294 bytes)	file-ratio (63.37 %)	md5
BIN	128	0x00008408	Executable ...	-	61440	62.50 %	86EA1F46DF745A30577F02FC24E266FF
String-table	1	0x00017408	String-table	x	62	0.06 %	757175BDAB76384D6A82D133E6B75A96
Version	1	0x000080F0	Version	x	792	0.81 %	BCB39DD0FE68C58719F30B8BB292553B

Figure 1: Payload as resource of the Loader

When started, the executable creates a new folder on "*C:\intel*" and then starts inspecting all the running processes. It looks for a really particular one: "*fwmain32.exe*". This lookup reveals how deeply environmental aware is this implant. In fact, the "*fwmain32*" process is part of the software services produced by Wincor Nixdorf International GmbH, one of the major vendors providing retail and banking hardware such as ATMs.

4038	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("[System Process]", "fwmain32.exe")
4039	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("System", "fwmain32.exe")
4040	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("smss.exe", "fwmain32.exe")
4041	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("csrss.exe", "fwmain32.exe")
4042	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("wininit.exe", "fwmain32.exe")
4043	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("services.exe", "fwmain32.exe")
4044	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("lsass.exe", "fwmain32.exe")
4045	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("lsm.exe", "fwmain32.exe")
4046	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4047	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("VBoxService.exe", "fwmain32.exe")
4048	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4049	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4050	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4051	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4052	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4053	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4054	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4055	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("spoolsv.exe", "fwmain32.exe")
4056	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4057	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4058	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4059	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("KMS-QAD.exe", "fwmain32.exe")
4060	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("SearchIndexer.exe", "fwmain32.exe")
4061	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("wmpnetwk.exe", "fwmain32.exe")
4062	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4063	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("svchost.exe", "fwmain32.exe")
4064	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("csrss.exe", "fwmain32.exe")
4065	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("winlogon.exe", "fwmain32.exe")
4066	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("taskhost.exe", "fwmain32.exe")
4067	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("dwm.exe", "fwmain32.exe")
4068	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("explorer.exe", "fwmain32.exe")
4069	11:23:59.13...	1	bf9c35d8f33e2...	_stricmp ("VBoxTray.exe", "fwmain32.exe")

Figure 2: Research of “fwmain32.exe” process by malware

Once the “*fwmain32.exe*” process is found, the loader injects the actual payload in its own memory, infecting it. The payload DLL, initially stored into the loader resources section, will be implanted into the target process using the “SetThreadContext” injection technique (Thread Hijacking).

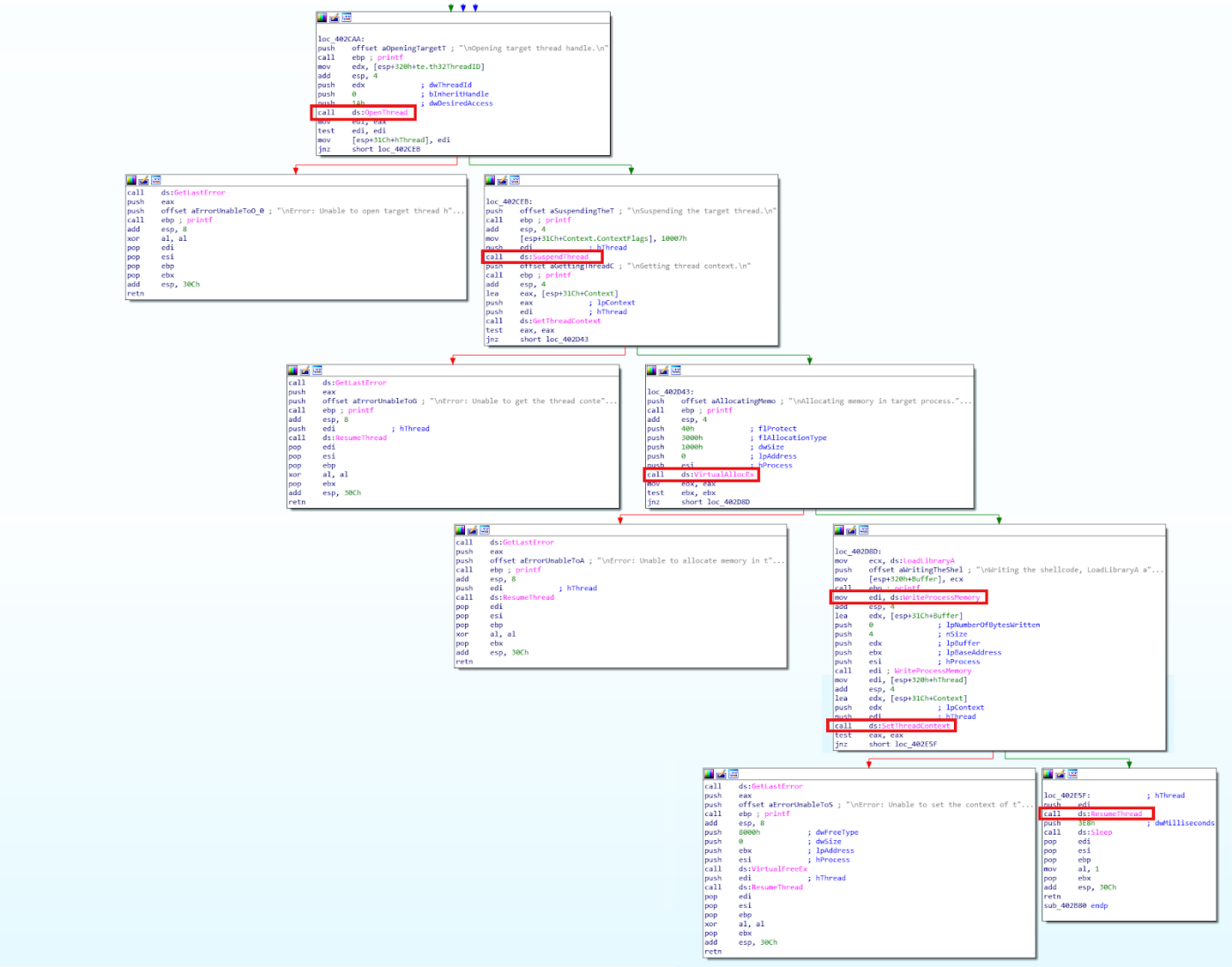


Figure 3: Complete Thread Hijacking flow

The figure above shows the sample calls on the *OpenThread* and the *SuspendThread* functions to pause the current execution. After allocating the right memory amount in the target process, it writes the shellcode target memory space using the *WriteProcessMemory* function and sets up the new process context with *SetThreadContext*. Finally, using the *ResumeThread* function the payload is able to start its malicious execution.

When the loader succeeded to inject the payload into the "*fwmain*" process, it also shows a popup window reporting the outcome of the injection phases.

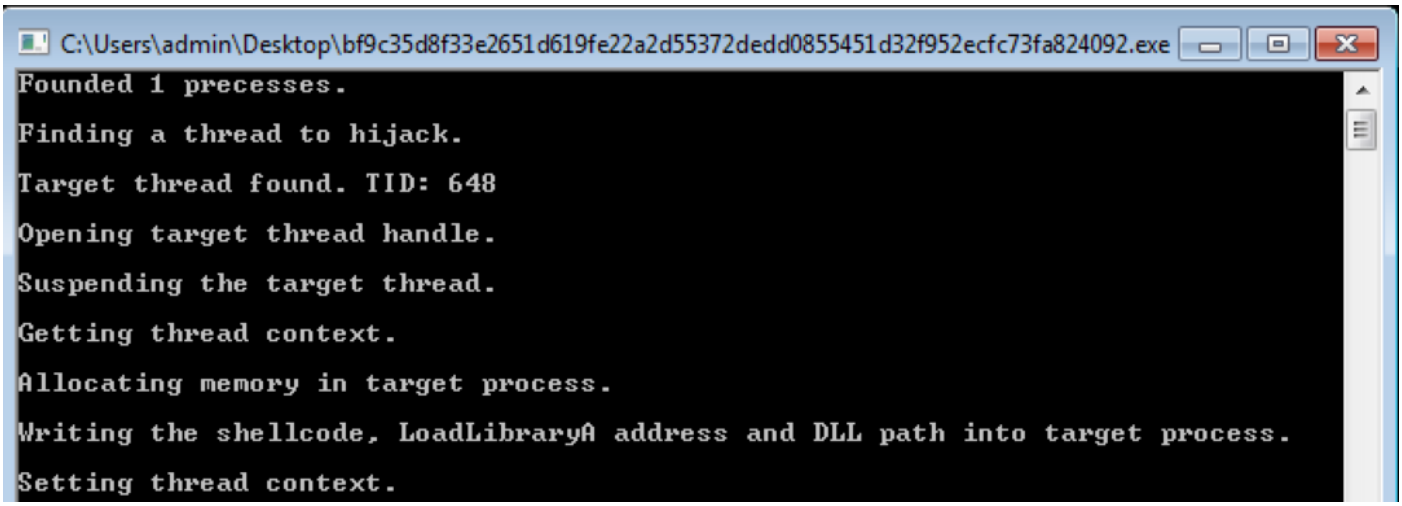


Figure 4: Prompt window reporting the log of the injection phase

ATMitch Payload

Sha256	e372631f96face11e803e812d9a77a25d0a81fa41e4ac362dc8aee5c8a021000
Threat	ATMitch ATM malware
Brief description	ATMitch payload
Ssdeep	768:N/qZvnFW5PJizM5qy1ucRM7YNNsrGkc+uW9LMQDFd+MbfRprj:N/0vniPJiz3yUrvGkc+uylR

Table 2: Information about the payload (DLL contained as resource in the Dropper/Loader)

The injected DLL has a very characteristic dependency: it requires the "*msxfs.dll*". This library provides access to the E**X**tension for **F**inancial **S**ervice (XFS) API, the communication interface needed to interact with AMT components such as PIN pad and cash dispenser. Again, this is a very particular dependency can only be resolved on special purpose Windows environment, like the Wincor machines.

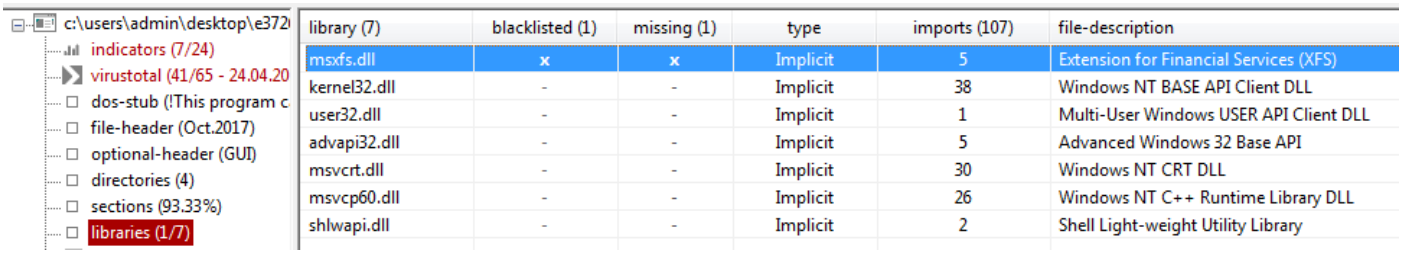


Figure 5: "msxfs.dll", library required by malware to communicate with ATM device

The malware is quite simple: it reads commands from a file included into *"c:\intel"* folder and interacts with the ATM drivers in order to retrieve information about the current amount and to dispense money at the right time. In the following screen is shown a function used to initiate the communication with the PinPad and Dispenser ATM components.

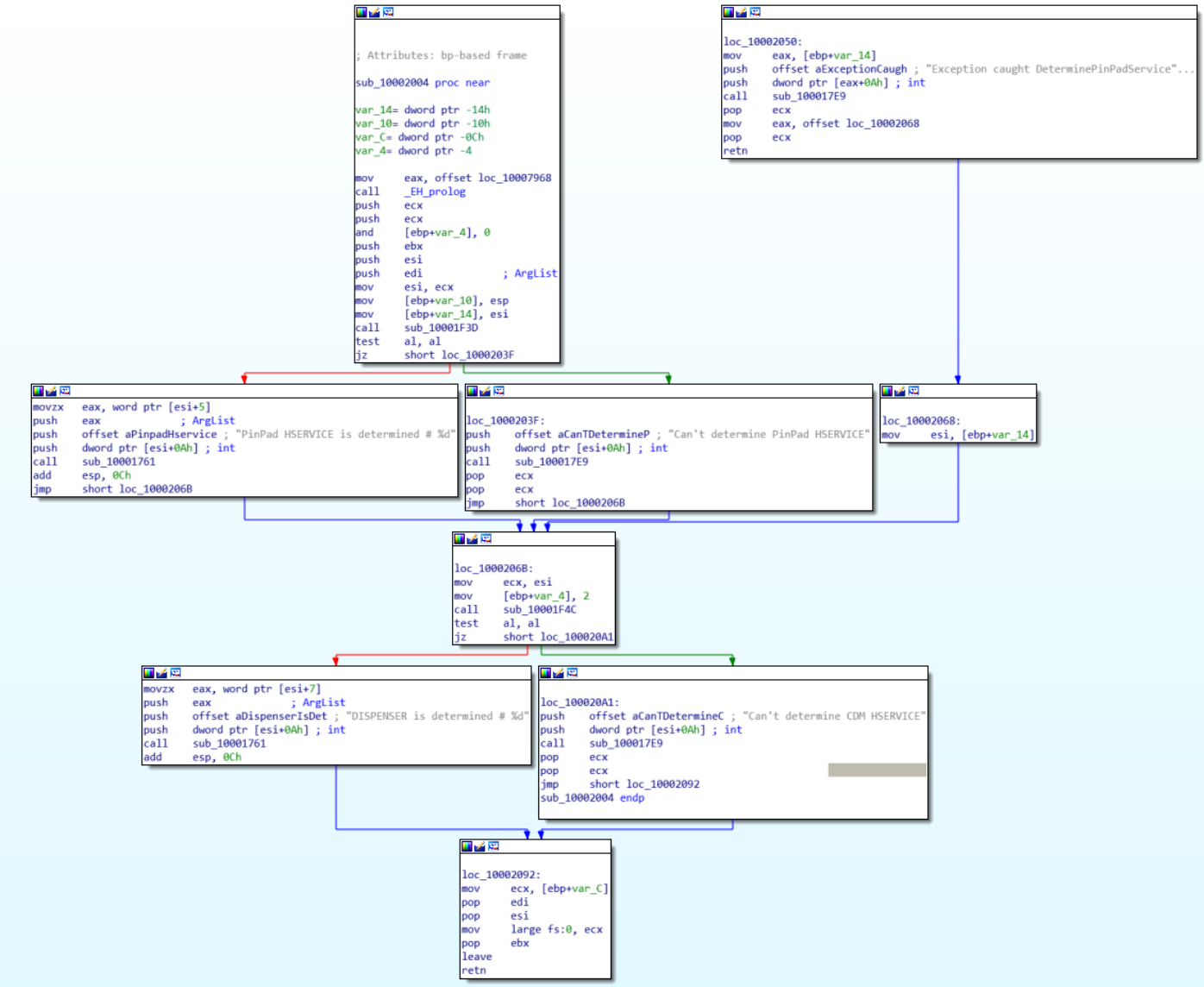


Figure 6: Discovering of PinPad and Dispenser components

Using the functions provided by “msxfs.dll” library, the malware can easily interact with these components. For example, using the *WFSExecute* function it is possible to send one of the supported commands to the dispenser, like *OPEN_SHUTTER* or *OPEN_SAFE_DOOR*.

```
/* CDM Execute Commands */

#define     WFS_CMD_CDM_DENOMINATE              (CDM_SERVICE_OFFSET + 1)
#define     WFS_CMD_CDM_DISPENSE                (CDM_SERVICE_OFFSET + 2)
#define     WFS_CMD_CDM_PRESENT                 (CDM_SERVICE_OFFSET + 3)
#define     WFS_CMD_CDM_REJECT                  (CDM_SERVICE_OFFSET + 4)
#define     WFS_CMD_CDM_RETRACT                 (CDM_SERVICE_OFFSET + 5)
#define     WFS_CMD_CDM_OPEN_SHUTTER            (CDM_SERVICE_OFFSET + 7)
#define     WFS_CMD_CDM_CLOSE_SHUTTER           (CDM_SERVICE_OFFSET + 8)
#define     WFS_CMD_CDM_SET_TELLER_INFO         (CDM_SERVICE_OFFSET + 9)
#define     WFS_CMD_CDM_SET_CASH_UNIT_INFO      (CDM_SERVICE_OFFSET + 10)
#define     WFS_CMD_CDM_START_EXCHANGE          (CDM_SERVICE_OFFSET + 11)
#define     WFS_CMD_CDM_END_EXCHANGE            (CDM_SERVICE_OFFSET + 12)
#define     WFS_CMD_CDM_OPEN_SAFE_DOOR          (CDM_SERVICE_OFFSET + 13)
#define     WFS_CMD_CDM_CALIBRATE_CASH_UNIT     (CDM_SERVICE_OFFSET + 15)
#define     WFS_CMD_CDM_SET_MIX_TABLE           (CDM_SERVICE_OFFSET + 20)
#define     WFS_CMD_CDM_RESET                   (CDM_SERVICE_OFFSET + 21)
#define     WFS_CMD_CDM_TEST_CASH_UNITS         (CDM_SERVICE_OFFSET + 22)
#define     WFS_CMD_CDM_COUNT                   (CDM_SERVICE_OFFSET + 23)
```

Figure 7: Part of commands accepted by ATM

In the specific case, the malware uses the function to dispense money through the command WFS_CMD_CDM_DISPENSE, as shown in figure:

```
push    eax             ; ArgList
push    offset aDispenseDispen ; "Dispense, dispense device is %d"
push    dword ptr [ebx+0Ah] ; int
call    vsnprintf_sub_10001761
add     esp, 0Ch
lea     eax, [ebp+var_10]
push    eax
mov     ax, [ebx+7]
push    0
push    edi
push    12Eh            ; 302 WFS_CMD_CDM_DISPENSE
push    eax             ; device ID
call    WFSExecute
push    [ebp+var_10]
mov     [ebp+arg_0], eax
call    WFSFreeResult
```

Figure 8: Command “WFS_CMD_CDM_DISPENSE” used by malware to dispense money

The core of the malware is the following switch structure: after reading the new command from the specific file, it compares the command code with the embedded ones, such as “code 2” for retrieving information or “code 7” for dispensing money.

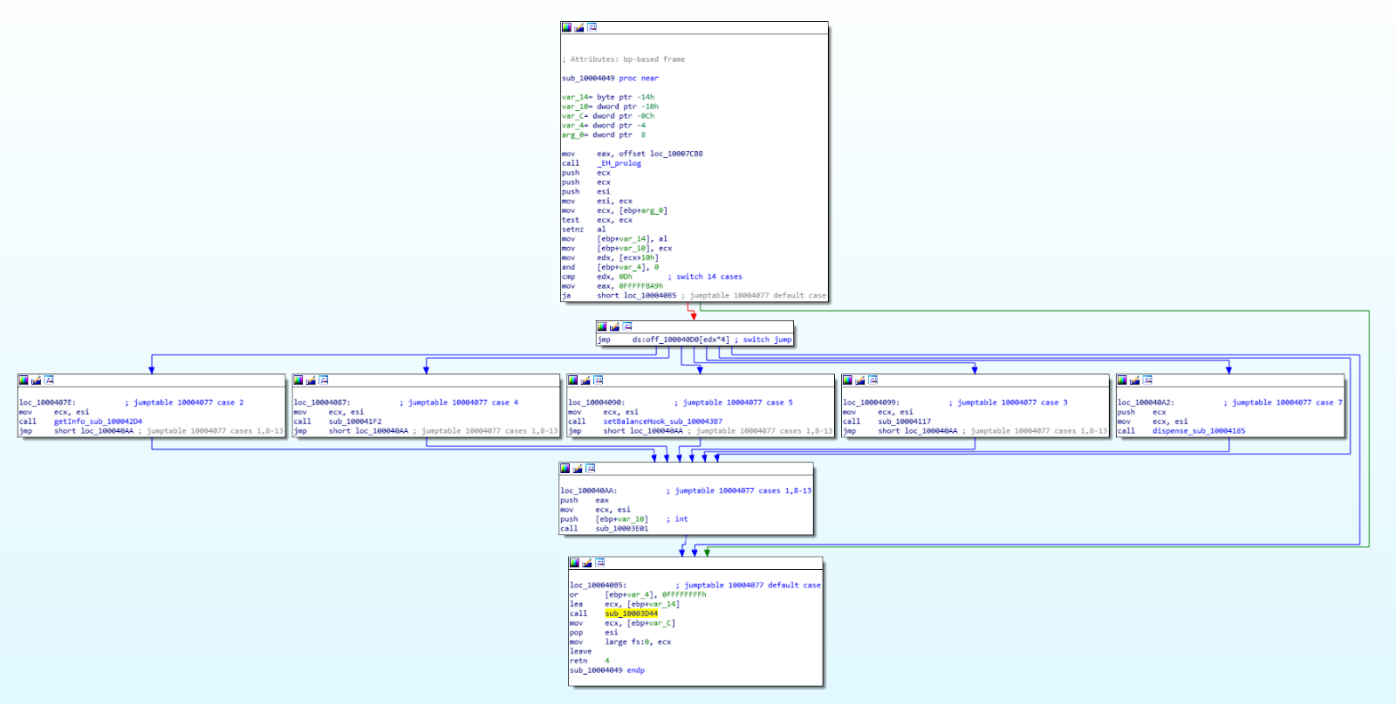


Figure 9: Malware' switch structure

Moreover, the malware provides a well-structured logging system: all actions are traced and logged into *"c:\intel__log.txt"*. In relation to the action that needs to be logged, it is able to set a specific logging level (FATAL, ERROR, DEBUG etc.).

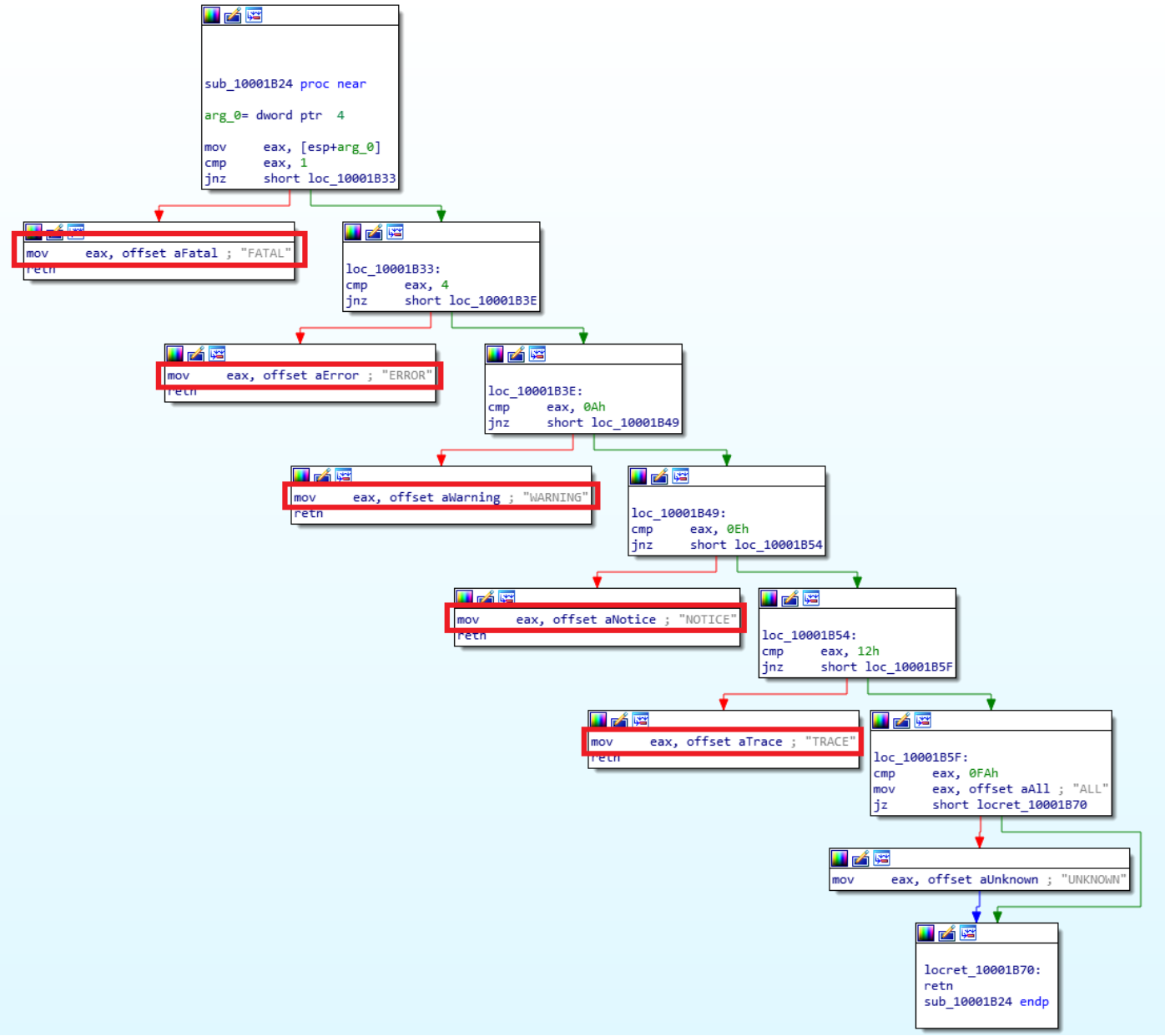


Figure 10: Logging-level of the malware logging system

Conclusion

This recently discovered ATMitch sample is one of the key assets used by advanced attackers during bank cyber-robberies, potentially even by the Carbanak or the GCMAN group. Who manually install it within segregated hosts and write commands directly into the target machine, without any command and control traffic. The usage of Remote Desktop to directly connect to the target machine is also supported by the presence of a prompt window (Figure 4) which shows the correct execution of the first stage. Probably the last steps of an attack flow involving ATMitch are the following:

1. The attacker connects to the ATM machine using Remote Desktop;
2. The attacker transfers the loader EXE and runs it: the prompt window shows if everything went well;
3. The attacker deletes the initial file in order to remove tracks;
4. The attacker writes commands in the appropriate file;
5. The malware executes the new commands and writes in the log file;
6. The attacker examines the log file to know the state of the command execution.

So, the eventual presence of this malware could be the tip of the iceberg of a more complex and articulated attack perpetrated by advanced cyber-criminals.

Indicators of Compromise

Hashes

- bf9c35d8f33e2651d619fe22a2d55372dedd0855451d32f952ecfc73fa824092
- e372631f96face11e803e812d9a77a25d0a81fa41e4ac362dc8aee5c8a021000

Yara Rules

```
import "pe"
rule ATMitch {
meta:
          description = "Yara Rule for ATMitch Dropper/Payload"
          author = "ZLAB Yoroi - Cybaze"
          last_updated = "2019-05-03"
          tlp = "white"
          category = "informational"

   strings:
        $str1 = {4A 75 E6 8B C7 8B 4D FC}
         $str2 = {EC 53 8D 4D DC 88}
        $str3 = "MSXFS.dll"
        $str4 = "DISPENSE"
        $str5 = "PinPad"
        $str6 = "cash"
        $str7 = {40 59 41 50 41 58 49 40 5A}
        $str8 = "WFMFreeBuffer"

condition:
        pe.number_of_sections == 4 and pe.number_of_resources == 3 and $str1 and $str2 or $str3
and $str4 and $str5 and $str6 and $str7 and $str8
}
```

This blog post was authored by Antonio Farina, Davide Testa, Antonio Pirozzi and Luca Mella of Cybaze-Yoroi Z-LAB