ATMii: a small but effective ATM robber

securelist.com/atmii-a-small-but-effective-atm-robber/82707/

Authors

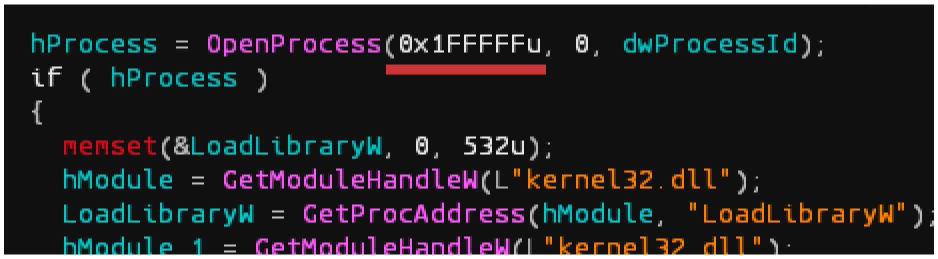
Expert Konstantin Zykov

While some criminals blow up ATMs to steal cash, others use less destructive methods, such as infecting the ATM with malware and then stealing the money. We have written about this phenomenon extensively in the past and today we can add another family of malware to the list – Backdoor.Win32.ATMii.

ATMii was first brought to our attention in April 2017, when a partner from the financial industry shared some samples with us. The malware turned out to be fairly straightforward, consisting of only two modules: an injector module (exe.exe, 3fddbf20b41e335b6b1615536b8e1292) and the module to be injected (dll.dll, dc42ed8e1de55185c9240f33863a6aa4). To use this malware, criminals need direct access to the target ATM, either over the network or physically (e.g. over USB). ATMii, if it is successful, allows criminals to dispense all the cash from the ATM.

exe.exe – an injector and control module

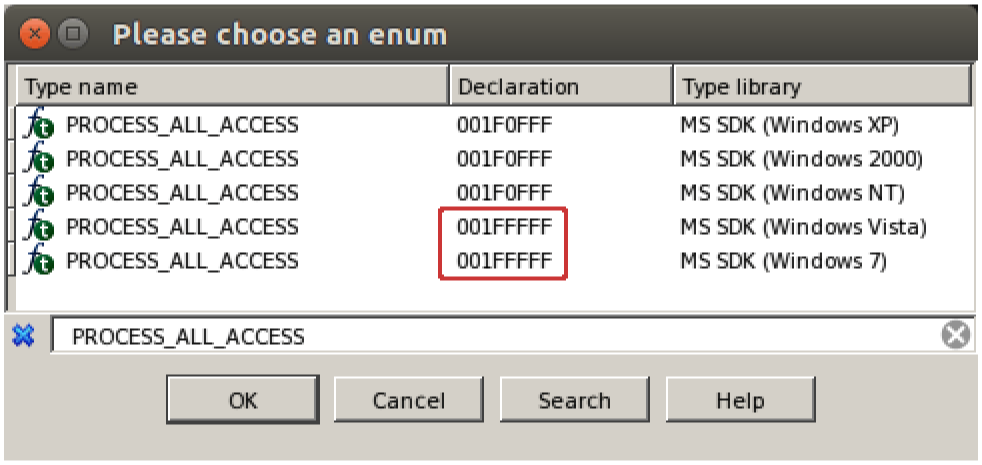
The injector is an unprotected command line application, written in Visual C with a compilation timestamp: Fri Nov 01 14:33:23 2013 UTC. Since this compilation timestamp is from 4 years ago – and we do not think this threat could have gone unnoticed for 4 years – we believe it is a fake timestamp. What's also interesting is the OS that is supported by the malware: One more recent than Windows XP. We can see this in the image below, where the first argument for the OpenProcess() function is 0x1FFFFu.



```
hProcess = OpenProcess(0x1FFFFFu, 0, dwProcessId);
if ( hProcess )
{
  memset(&LoadLibraryW, 0, 532u);
  hModule = GetModuleHandleW(L"kernel32.dll");
  LoadLibraryW = GetProcAddress(hModule, "LoadLibraryW");
  hModule_1 = GetModuleHandleW(L"kernel32.dll");
```

OpenProcess call with the PROCESS_ALL_ACCESS constant

It is the *PROCESS_ALL_ACCESS* constant, but this constant value differs in older Windows versions such as Windows XP (see the picture below). This is interesting because most ATMs still run on Windows XP, which is thus not supported by the malware.



Please choose an enum

Type name	Declaration	Type library
PROCESS_ALL_ACCESS	001F0FFF	MS SDK (Windows XP)
PROCESS_ALL_ACCESS	001F0FFF	MS SDK (Windows 2000)
PROCESS_ALL_ACCESS	001F0FFF	MS SDK (Windows NT)
PROCESS_ALL_ACCESS	001FFFFF	MS SDK (Windows Vista)
PROCESS_ALL_ACCESS	001FFFFF	MS SDK (Windows 7)

PROCESS_ALL_ACCESS

A list of PROCESS_ALL_ACCESS values per Windows version

The injector, which targets the *atmapp.exe (*proprietary ATM software*)* process, is fairly poorly written, since it depends on several parameters. If none are given, the application catches an exception. The parameters are pretty self-explanatory:

param	short description

/load	Tries to inject *dll.dll* into *atmapp.exe* process
/cmd	Creates/Updates *C:\ATM\c.ini* file to pass commands and params to infected library
/unload	Tries to unload injected library from *atmapp.exe* process, while restoring its state.

/load param

<exe.exe> /load

The application searches for a process with the name atmapp.exe and injects code into it that loads the “dll.dll” library (which has to be in the same folder as the exe.exe file). After it has been loaded it calls the DLLmain function.

/unload param

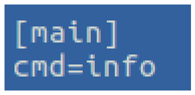
<exe.exe> /unload

As the name already suggests, it is the opposite of the /load parameter; it unloads the injected module and restores the process to its original state.

/cmd param

<exe.exe> /cmd [cmd] [params]

The application creates/updates *C:\ATM\c.ini* which is used by the injected DLL to read commands. The file is updated each time the .exe is run with the */cmd* param.



```
[main]
cmd=info
```

Contents of c.ini after execution of “exe.exe /cmd info”

The executable understands the following set of commands:

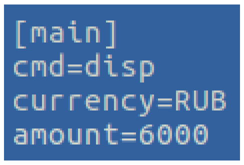
command	description
scan	Scans for the CASH_UNIT XFS service
disp	Stands for “dispense”. The injected module should dispense “*amount*” cash of “*currency*” (*amount* and *currency* are used as parameters)
info	Gets info about ATM cash cassettes, all the returned data goes to the log file.

die	Injected module removes *C:\ATM\c.ini file*

dll.dll injecting module

After injection and execution of the *DllMain* function, the *dll.dll* library loads *msxfs.dll* and replaces the *WFSGetInfo* function with a special wrap function, named ***m****WFSGetInfo*.

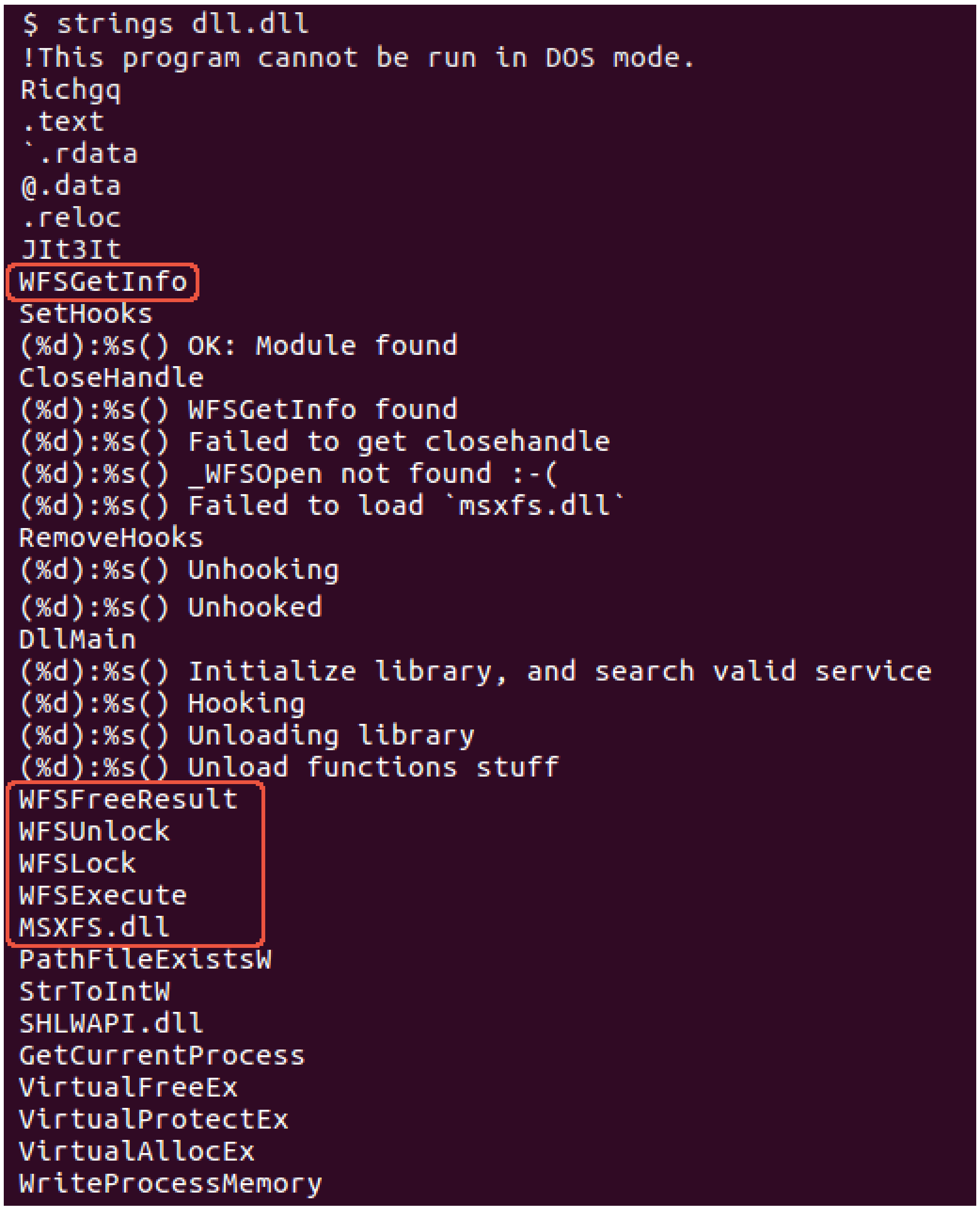
At the time of the first call to the fake *WFSGetInfo* function, *C:\ATM\c.ini* is ignored and the library tries to find the ATM's CASH_UNIT service id and stores the result, basically in the same way as the *scan* command does. If the CASH_UNIT service is not found, *dll.dll* won't function. However, if successful, all further calls go to the ***m****WFSGetInfo* function, which performs the additional logic (reading, parsing and executing the commands from the *C:\ATM\c.ini* file).



```
[main]
cmd=disp
currency=RUB
amount=6000
```

Contents of C:\ATM\c.ini after execution of "exe.exe /cmd disp RUB 6000"

Below is an output of the *strings* program uncovering some interesting log messages and the function names to be imported. The proprietary MSXFS.DLL library and its functions used in the ATMii malware are marked with red boxes.



```
$ strings dll.dll
!This program cannot be run in DOS mode.
Richgq
.text
`.rdata
@.data
.reloc
JIt3It
WFSGetInfo
SetHooks
(%d):%s() OK: Module found
CloseHandle
(%d):%s() WFSGetInfo found
(%d):%s() Failed to get closehandle
(%d):%s() _WFSOpen not found :-(
(%d):%s() Failed to load `msxfs.dll`
RemoveHooks
(%d):%s() Unhooking
(%d):%s() Unhooked
DllMain
(%d):%s() Initialize library, and search valid service
(%d):%s() Hooking
(%d):%s() Unloading library
(%d):%s() Unload functions stuff
WFSFreeResult
WFSUnlock
WFSLock
WFSExecute
MSXFS.dll
PathFileExistsW
StrToIntW
SHLWAPI.dll
GetCurrentProcess
VirtualFreeEx
VirtualProtectEx
VirtualAllocEx
WriteProcessMemory
```

“scan” command

Because of the architecture of XFS, which is divided into services, the injected library first needs to find the dispense service. This command must be successfully called, because the *disp* and *info* commands depend on the service id retrieved by *scan. Scan* is automatically called after the *dll* has been injected into *atmapp.exe.*

After collecting the WFS_INF_CDM_STATUS data, additional data gets added to the *tlogs.log*. An example can be found below:

…
(387):cmd_scan() Searching valid service
(358):FindValidService() Checking device index=0

```
(70):CheckServiceForValid() ——————————————————
(72):CheckServiceForValid() Waiting for lock
(76):CheckServiceForValid() Device was locked
(86):CheckServiceForValid() WFSGetInfo Success 0
(182):CheckServiceForValid() Done-> szDevice: WFS_CDM_DEVONLINE, szDispenser:
WFS_CDM_DISPOK, szIntermediateStacker: WFS_CDM_ISEMPTY, szSafeDoor:
WFS_CDM_DOORCLOSED
(195):CheckServiceForValid() Unlocking device
(390):cmd_scan() Service found 0
…
```

Part of a tlogs.log possible log after successfully executed "scan" command

"info" command

Before the criminals can dispense cash, they first need to know the exact contents of the different cassettes. For this, they use the *info* command which provides exhaustive information on all cassettes and their contents. The list of used XFS API functions is the same as with the *scan* command, but this time *WFSGetInfo* is called with the WFS_INF_CDM_CASH_UNIT_INFO (303) constant passed as a param.

Below is an example of the data in log file returned by the *info* command.

```
…
(502):ExecuteCmd() Executing cmd
(506):ExecuteCmd() CMD = info
(402):cmd_info() ! hFoundGlobalService = 0
(213):GetDeviceInformation() ——————————————————
(220):GetDeviceInformation() Device locked 0
(337):GetDeviceInformation() Module: C:\program files\dtatmw\bin\atmapp\atmapp.exe
Cash Unit # 1, name=SOMENAME
Type: 3
Status: HIGH
Currency ID: 0x52-0x55-0x42
Note Value: 5000
Notes Count: 3000
Notes Initial Count: 3000
Notes Minimum Count: 10
Notes Maximum Count: 0
…
```

Example5 Part of a tlogs.log possible log after successfully executed "info" command

"disp" command

The dispense command is followed by two additional params in the command file: *currency* and *amount.* Currency must contain one of the three-letter currency codes of notes kept in the CASH_UNIT_INFO structure (currency codes are described in ISO_4217 e.g. RUB, EUR). The *amount* code holds the amount of cash to dispense and this value must be a multiple of ten.

“die” command

Does nothing except deleting *C:\ATM\c.ini* command file.

Conclusion

ATMii is yet another example of how criminals can use legitimate proprietary libraries and a small piece of code to dispense money from an ATM. Some appropriate countermeasures against such attacks are default-deny policies and device control. The first measure prevents criminals from running their own code on the ATM’s internal PC, while the second measure will prevent them from connecting new devices, such as USB sticks.

- ATM
- Backdoor
- Financial malware
- Malware Technologies

Authors

Konstantin Zykov

ATMii: a small but effective ATM robber

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