

Contents

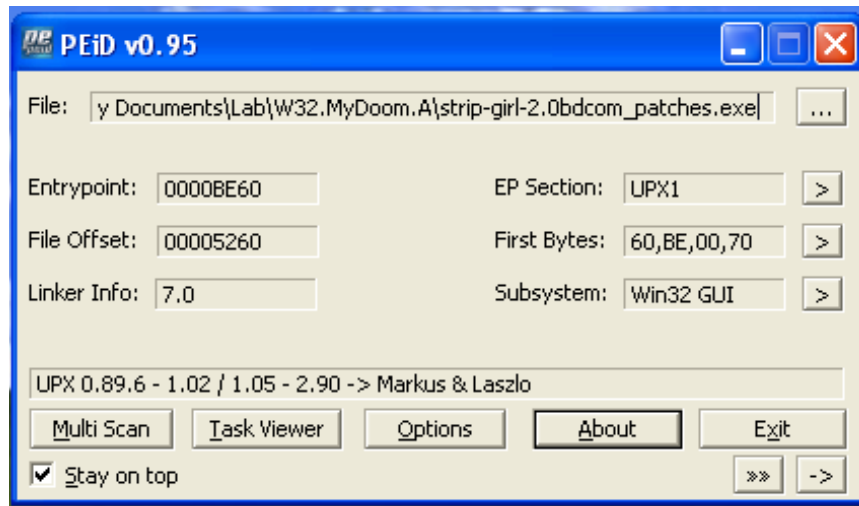
1	Introduction	2
2	Static Analysis	2
2.1	Data	2
2.1.1	Unpacking	2
2.1.2	Imports	3
2.1.3	Strings	4
2.1.4	Arrays	8
2.2	Control Flow	9
2.2.1	Entry Point	9
2.2.2	Main Function	11
2.3	Malicious Functionality	13
2.3.1	DoS Attack	13
2.3.2	Replication	15
3	Mitigation	20
3.1	Detection	20
3.2	Recovery	20

1 Introduction

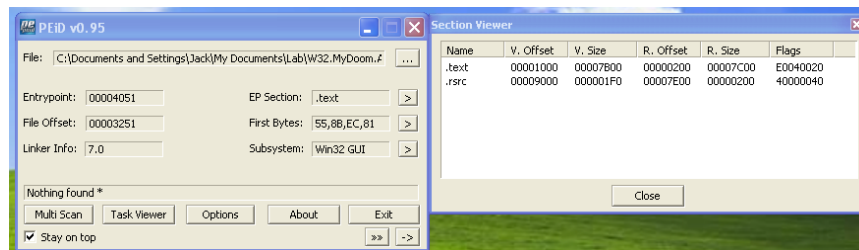
2 Static Analysis

2.1 Data

2.1.1 Unpacking



Initially opening the malware in PEiD, it appears to be packed with UPX.



After unpacking with the UPX tool, we can see the original sections.

2.1.2 Imports

Address	Ordinal	Name	Library
004A1000		RegCloseKey	ADVAPI32
004A1004		RegOpenKeyExA	ADVAPI32
004A1008		RegSetValueExA	ADVAPI32
004A10...		RegQueryValueExA	ADVAPI32
004A1010		RegEnumKeyA	ADVAPI32
004A1014		RegCreateKeyExA	ADVAPI32
004A10...		GetLocalTime	KERNEL32
004A10...		FileTimeToLocalFileTime	KERNEL32
004A10...		MapViewOfFile	KERNEL32
004A10...		GetTimeZoneInformation	KERNEL32
004A10...		ExitProcess	KERNEL32
004A10...		GetTickCount	KERNEL32
004A10...		CreateThread	KERNEL32
004A10...		CreateProcessA	KERNEL32
004A10...		WaitForSingleObject	KERNEL32
004A10...		ExitThread	KERNEL32
004A10...		Sleep	KERNEL32
004A1098		GetSystemTimeAsFileTime	KERNEL32
004A1094		SystemTimeToFileTime	KERNEL32
004A1090		GetModuleFileNameA	KERNEL32
004A10...		WideCharToMultiByte	KERNEL32
004A10...		GetProcAddress	KERNEL32
004A10...		GetModuleHandleA	KERNEL32
004A10...		HeapFree	KERNEL32
004A10...		GetProcessHeap	KERNEL32
004A10...		HeapAlloc	KERNEL32
004A10...		lstrcpynA	KERNEL32
004A10...		lstrcmA	KERNEL32

The malware contains various imports related to:

- Modifying the Registry
- Reading/Writing files
- Managing Threads and Synchronization
- Mutexes
- DLL Loading
- String Handling
- Socket Programming

2.1.3 Strings

Address	Length	T...	String
"..." .text:00...	0000000D	C	fuvztncv.qyy
"..." .text:00...	00000044	C	Fbsginer\\Zvpebfbsg\\Jvaqbjf\\Pheeraglfvba\\Rkcybere\\PbzQyt32\\lr...
"..." .text:00...	0000000F	C	FjroFvcpFzgkF0
"..." .text:00...	0000000C	C	gnfxzba.rkr
"..." .text:00...	00000008	C	GnfxZba
"..." .text:00...	0000002E	C	Fbsginer\\Zvpebfbsg\\Jvaqbjf\\Pheeraglfvba\\Eha
"..." .text:00...	00000008	C	notepad %s
"..." .text:00...	00000008	C	Message
"..." .text:00...	00000027	C	%s, %u %s %u %2u:%2u:%2u %s%2u%2u
"..." .text:00...	0000001B	C	abcdefghijklmnopqrstuvwxyz
"..." .text:00...	0000001B	C	ABCDEFGHIJKLMNOPQRSTUVWXYZ
"..." .text:00...	0000001A	C	VagrearTgrPbaarpgrqFgng
"..." .text:00...	0000000C	C	jvavarg.qyy
"..." .text:00...	00000009	C	ahxr2004
"..." .text:00...	0000000D	C	bssvpr_penpx
"..." .text:00...	0000000A	C	ebbgxvgKC
"..." .text:00...	0000001C	C	fgevc-tvey-2.0oqpbz_cngpurf
"..." .text:00...	00000011	C	npgvingvba_penpx
"..." .text:00...	0000000E	C	vpd2004-svary
"..." .text:00...	00000008	C	jvanzc5
"..." .text:00...	00000007	C	QyQve0
"..." .text:00...	00000018	C	Fbsginer\\Xnmnr\\Genafsre
"..." .text:00...	0000000D	C	iphlpapi.dll
"..." .text:00...	00000008	C	DnsQuery_A
"..." .text:00...	00000008	C	dnsapi.dll
"..." .text:00...	00000011	C	GetNetworkParams
"..." .text:00...	00000007	C	sandra
"..." .text:00...	00000006	C	linda

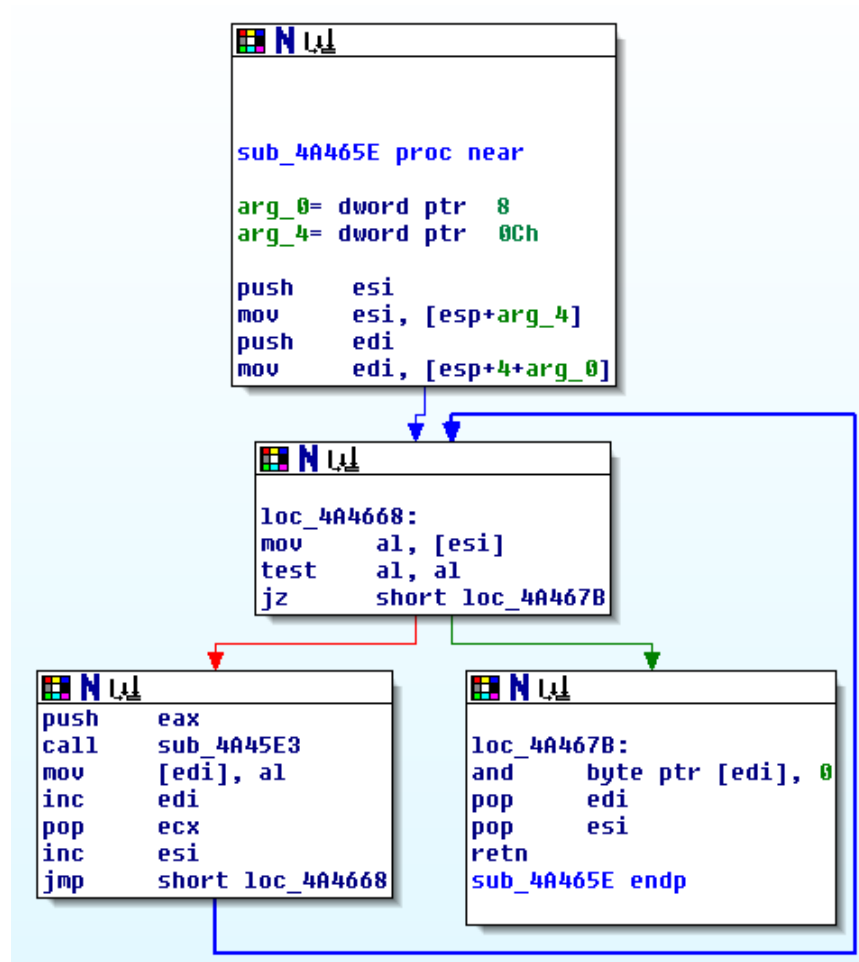
Looking at the strings, there seems to be many regular strings such as people's names, imports, and names, but also a few strings with all alphabetic characters and numbers. This points to some sort of encoding scheme used for the rest of the garbled strings, especially for the ones that look like paths, which could give a hint to where the virus modifies registry keys or files.

```

.text:004A2588 aAbcdefghijk1_0 db 'abcdefghijklmnopqrstuvwxyz',0 ; DATA XREF: sub_004A2588+1A10
.text:004A25A3 align 4
.text:004A25A4 aAbcdefghijk1mn db 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',0 ; DATA XREF: sub_004A2588+A10

```

Looking in the .text section, both of these "charsets" are used in a single subroutine. Searching for XREFs to this function, we find a single one which looks like the following.



It seems to call the previous function in a loop. Searching for XREFs to *this* function, we know we've found the decoding function.

xrefs to sub_4A465E			
Dir...	T	Address	Text
Up	p	sub_4A3962+15	call sub_4A465E
Up	p	sub_4A3AA3+17	call sub_4A465E
Up	p	sub_4A3B52+F	call sub_4A465E
Up	p	sub_4A3B88+15	call sub_4A465E
Up	p	sub_4A3CD7+16	call sub_4A465E
Up	p	sub_4A3CD7+24	call sub_4A465E
D...	p	sub_4A4681+10	call sub_4A465E
D...	p	sub_4A4681+4A	call sub_4A465E
D...	p	sub_4A46F7+1D	call sub_4A465E
D...	p	sub_4A46F7+2E	call sub_4A465E
D...	p	sub_4A46F7+117	call sub_4A465E
D...	p	sub_4A55D4+9E	call sub_4A465E
D...	p	sub_4A60A5+13	call sub_4A465E
D...	p	sub_4A64A8+18	call sub_4A465E
D...	p	sub_4A64A8+26	call sub_4A465E
D...	p	sub_4A68B4+17	call sub_4A465E
D...	p	sub_4A6C3B+18	call sub_4A465E
D...	p	sub_4A6CDE+B6	call sub_4A465E
D...	p	sub_4A6D9F+B2	call sub_4A465E

Line 1 of 38

```

lea    eax, [ebp+name]
push   offset aJjj_fpb_pbz ; "jjj.fpb.pbz"
push   eax
call   sub_4A465E

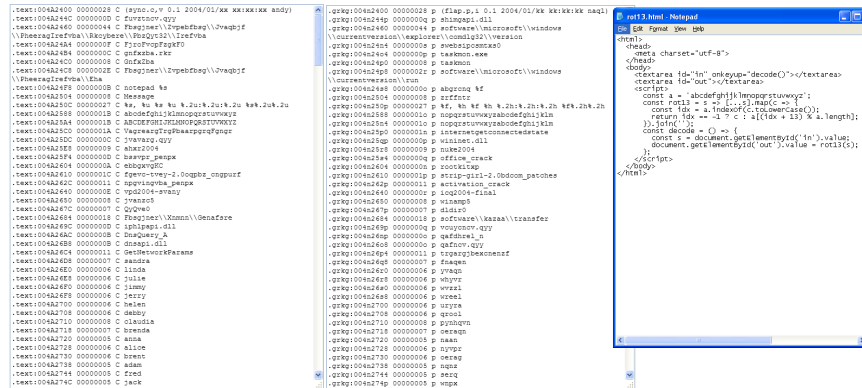
```

After analyzing both functions, it is pretty clear that this is a rot13 decoding function.



We created a quick tool to do rot13 decoding, and dumped all the strings

into it to see what I could find.



We can immediately see lots of previously garbled file names, registry paths, and more are now clear. In specific, `strip-girl-2.0bdcom_patches` was the name of the sample file. We can also see a reference to Kazaa, an early P2P file sharing application, which along with some filenames that sound like warez.

2.1.4 Arrays

Searching for xrefs to these strings, we find several large arrays in the `.text` segment that contain these encoded strings, all of which have been named accordingly or guessed what they are.

```

email_domains      dd offset aNby_pbz          ; DATA XREF: sub_4A6CDE+AE↓r
                   ; "nby.pbz"
                   dd offset aZfa_pbz          ; "zfa.pbz"
                   dd offset aLnubb_pbz        ; "lnubb.pbz"
                   dd offset aUbgznvy_pbz      ; "ubgznvy.pbz"

; LPCSTR names
names              dd offset aJohn            ; DATA XREF: sub_4A586A+88↓r
                   ; "john"
                   dd offset aJohn            ; "john"
                   dd offset aAlex            ; "alex"
                   dd offset aMichael         ; "michael"
                   dd offset aJames           ; "james"
                   dd offset aMike            ; "mike"
                   dd offset aKevin           ; "kevin"
                   dd offset aDavid           ; "david"

malware_names      dd offset aJvancz5         ; DATA XREF: sub_4A46F7:loc_4A47FC↓r
                   ; "jvancz5"
                   dd offset aUpd2004Svany    ; "upd2004-svany"
                   dd offset aNpgvinguba_pen  ; "npgvinguba_penpx"
                   dd offset aFgevcTvey2_0oq  ; "fgevc-tvey-2.0oqpbz_cngpurf"
                   dd offset aEbbgxvgkc       ; "ebbgxvgkc"
                   dd offset aBssvpr_penpx     ; "bssvpr_penpx"
                   dd offset aAhxr2004        ; "ahxr2004"

```



```

domains_1      dd offset aAvp          ; DATA XREF: sub_4A551A:loc_4A5598↓r
                                   ; sub_4A551A+86↓o
                                   ; "avp"
                                   ; "syms"
                                   ; "icrosof"
                                   ; "msn."
                                   ; "hotmail"
                                   ; "panda"
                                   ; "sopho"
                                   ; "borlan"
                                   ; "inpris"
                                   ; "example"
                                   ; "mydomai"
                                   ; "nodomai"
                                   ; "ruslis"
                                   ; ".gov"
                                   ; "gov."
                                   ; ".mil"
                                   ; "foo."
dd offset aSyma
dd offset aIcrosof
dd offset aMsn_
dd offset aHotmail
dd offset aPanda
dd offset aSopho
dd offset aBorlan
dd offset aInpris
dd offset aExample
dd offset aMydomai
dd offset aNodomai
dd offset aRuslis
dd offset a_gov
dd offset aGov_
dd offset a_mil
dd offset aFoo

```

2.2 Control Flow

2.2.1 Entry Point

After back-referencing some of the various functions, we found ourselves in the middle of a bunch of random code creating and sleeping threads all over the place. We decided to just take it from the top and trace the execution starting at the entry point.

The malware starts out by initializing the WinSock DLL and copying two dates into a large stack variable that is then passed to the rest of the code.

```

public start
start proc near

WSAData= WSAData ptr -3C4h
var_234= dword ptr -234h
var_20= dword ptr -20h
var_10= dword ptr -10h

push    ebp
mov     ebp, esp
sub     esp, 3C4h
push    esi
push    edi
call    sub_4A4215
lea     eax, [ebp+WSAData]
push    eax                ; lpWSAData
push    2                  ; wVersionRequested
call    WSAStartup         ; initialize WinSock DLL
push    234h               ; size_t
lea     eax, [ebp+var_234]
push    0                  ; int
push    eax                ; void *
call    memset
mov     esi, offset dword_4A2428
lea     edi, [ebp+var_10]
movsd
movsd
movsd
movsd
mov     esi, offset dword_4A2438
lea     edi, [ebp+var_20]
movsd
movsd
movsd
lea     eax, [ebp+var_234]
push    eax
movsd
call    main
add     esp, 10h
push    0                  ; uExitCode
call    ExitProcess

```

```

*.text:004A2428 dword_4A2428 dd 207D4h, 0C0000h, 1C0002h, 39h ; DATA XREF: start+3210
*.text:004A2438 dword_4A2438 dd 207D4h, 100000h, 90010h, 12h ; DATA XREF: start+3E10

```

Converting the 7D4h to decimal we get 2004, which tips us off that we

are working with dates. The mysterious constants turn out to be instances of a `struct SYSTEMTIME`, which formatted, are:

- 02:28:39 UTC on 12 February 2004
- 16:09:18 UTC on 01 February 2004

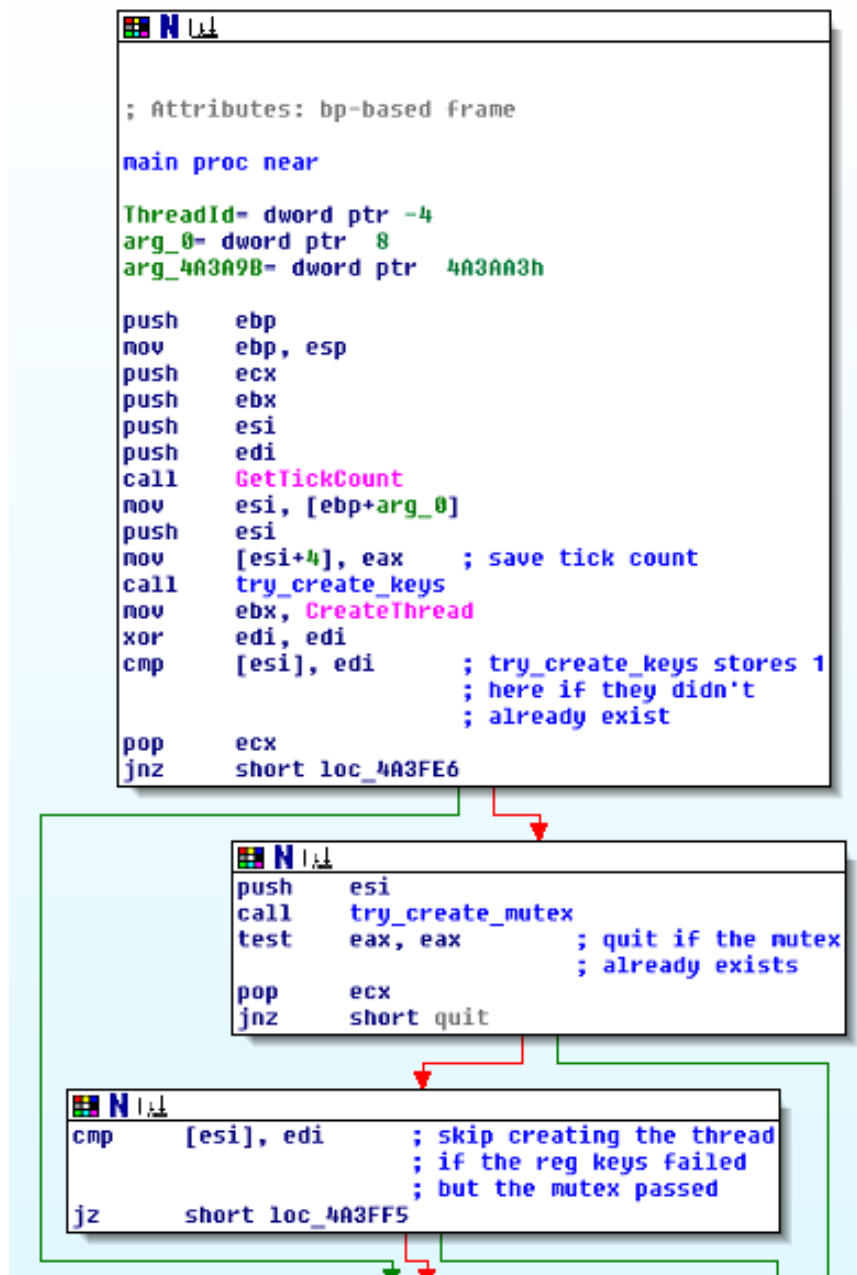
We will see later in the code where these dates are used.

2.2.2 Main Function

The main logic of the malware starts after the dates are copied into the buffer.

The malware firsts checks for the existence of 2 registry keys, creating them if they are not present.

If the registry keys exist, the malware tries to create a mutex. This prevents duplicate versions of the malware from conflicting.



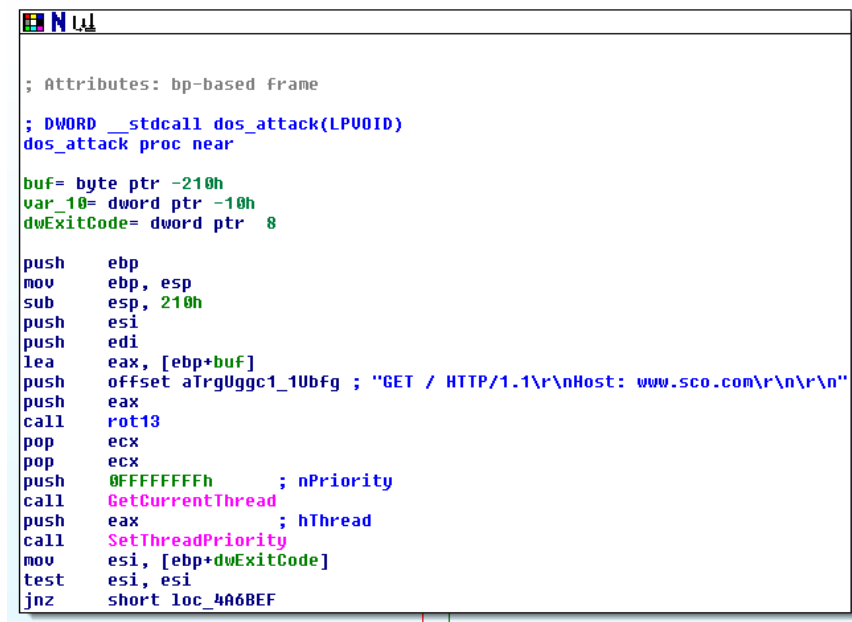
After this, the malware checks the current time against a hard coded date in the .text section, quitting if it is past, thus stopping the spread of the malware on February 12th, 2004 at precisely 02:28:39 UTC.

After this check, the malware proceeds with the rest of its malicious actions.

2.3 Malicious Functionality

2.3.1 DoS Attack

Looking at the XREFs for the string that contains an HTTP GET header to www.sco.com, we find a function that appears to execute a denial of service attack by repeatedly opening sockets and sending a GET request.

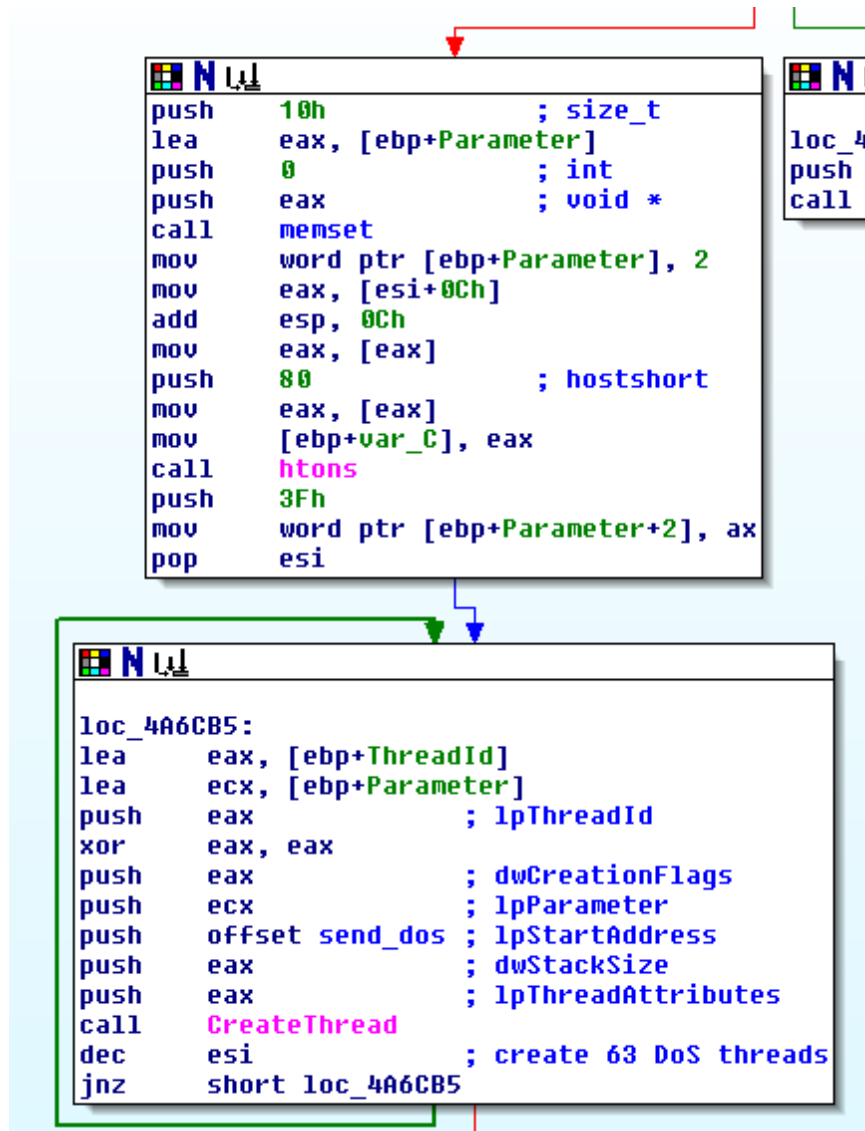


```
; Attributes: bp-based frame
; DWORD __stdcall dos_attack(LPVOID)
dos_attack proc near

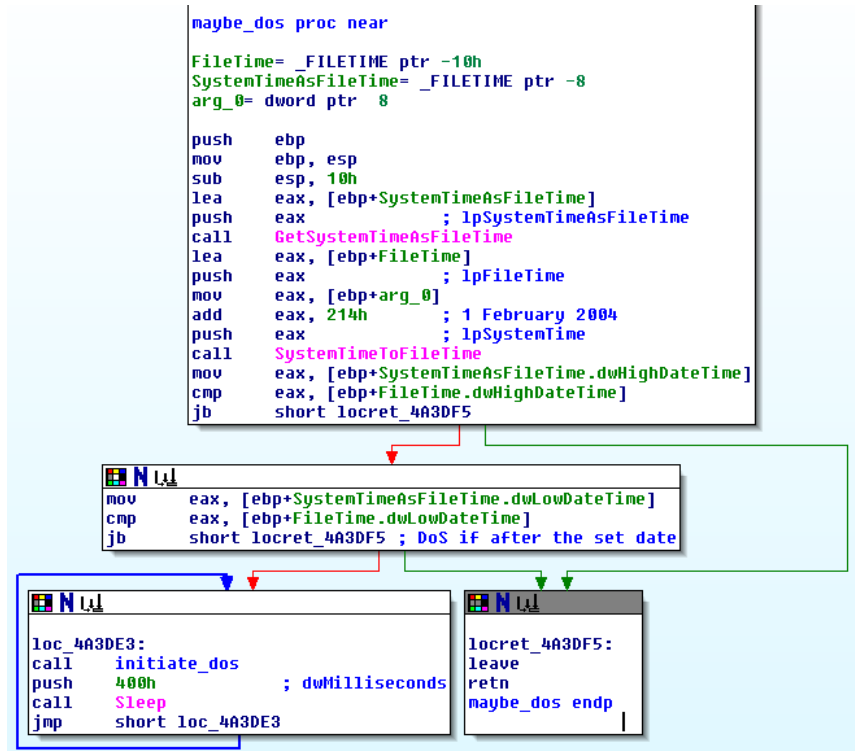
buf= byte ptr -210h
var_10= dword ptr -10h
dwExitCode= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 210h
push    esi
push    edi
lea     eax, [ebp+buf]
push    offset a1rgUggc1_1Ubfg ; "GET / HTTP/1.1\r\nHost: www.sco.com\r\n\r\n"
push    eax
call    rot13
pop     ecx
pop     ecx
push    0FFFFFFFh ; nPriority
call    GetCurrentThread
push    eax ; hThread
call    SetThreadPriority
mov     esi, [ebp+dwExitCode]
test    esi, esi
jnz     short loc_4A6BEF
```

The function creates 64 threads that repeatedly send HTTP requests to the target site as an attempt to take it down.



Before executing the attack, the malware again checks the current time against another hard coded date in the .text section, only executing the attack if it is past February 1st, 2004 at precisely 16:09:18 UTC.



2.3.2 Replication

1. Kazaa

Looking at the code XREFs to the strings related to malicious filenames and registry paths, we find a function that seems to create a malicious file.

The function reads the Kazaa shared directory from the registry, which makes contained files available to other users on the P2P file sharing platform.

```

; int __cdecl drop_file(LPCSTR input_filename)
drop_file proc near

SubKey= byte ptr -168h
ValueName= byte ptr -128h
NewFileName= byte ptr -108h
cbData= dword ptr -8
hKey= dword ptr -4
input_filename= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 168h
push    ebx
lea     eax, [ebp+SubKey]
push    offset aFbsgjnerXnmnnG ; "Fbsgjner\\Xnmnn\\Genafsre"
push    eax
mov     [ebp+cbData], 100h
call    rot13                ; SubKey = Software\\Kazaa\\Transfer
lea     eax, [ebp+ValueName]
push    offset aQyQue0       ; "QyQue0"
push    eax
call    rot13                ; ValueName = "D1Dir0"
push    [ebp+cbData]         ; size_t
xor     ebx, ebx
lea     eax, [ebp+NewFileName]
push    ebx                  ; int
push    eax                  ; void *
call    memset               ; memset(NewFileName, 0, 100h)
add     esp, 1Ch
lea     eax, [ebp+hKey]
push    eax                  ; phkResult
push    1                    ; samDesired
lea     eax, [ebp+SubKey]
push    ebx                  ; ulOptions
push    eax                  ; lpSubKey
push    80000001h            ; hKey
call    RegOpenKeyExA
test    eax, eax             ; return if the key doesn't exist
jnz     loc_4A48B3

```

The function adds \\ and a random filename from the previously discovered list to the path,


```

lea    eax, [ebp+NewFileName]
push   offset String2 ; "\\\"
push   eax             ; lpString1
call   edi ; lstrcatA ; name += "\\\"

```

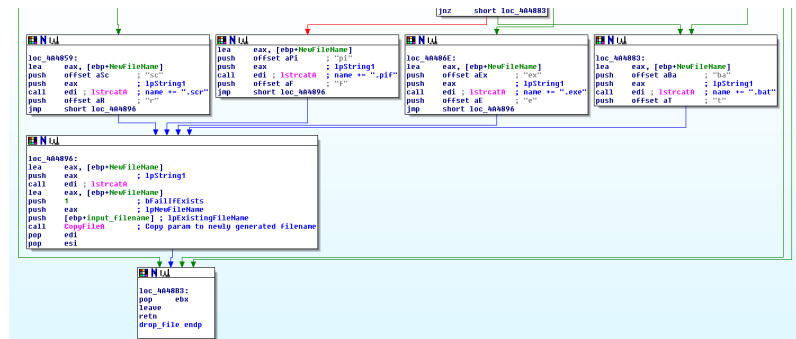
```

loc_4A47E8:
call   rand_word
movzx  eax, ax
push   7
cdq
pop     ecx
idiv   ecx
lea    eax, [ebp+NewFileName]

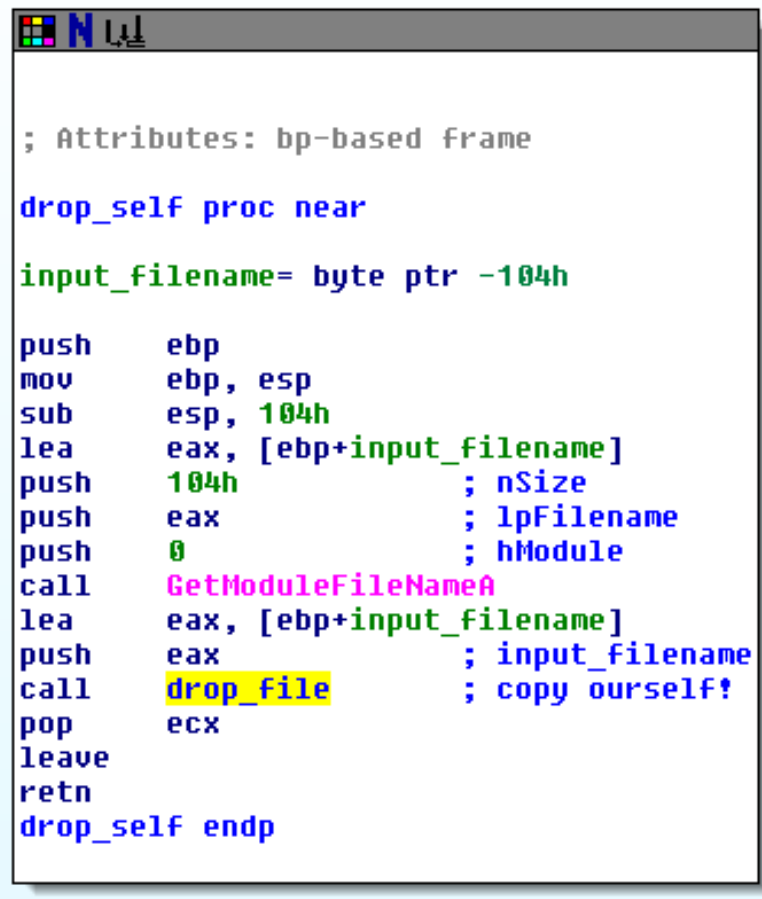
loc_4A47FC:
push   malware_names[edx*4]
push   eax             ; lpString
call   esi ; strlenA
lea    eax, [ebp+eax+NewFileName]
push   eax
call   rot13
pop     ecx
lea    eax, [ebp+NewFileName]
pop     ecx
push   offset a__1      ; "."
push   eax             ; lpString1
call   edi ; lstrcatA
call   rand_word
movzx  eax, ax
push   6
cdq
pop     ecx
idiv   ecx
cmp     edx, ebx
jl     short loc_4A4883

```

adds an extension, and copies the input file to the shared directory.



This function has a single XREF, where it is called with the module's own filepath.



```
; Attributes: bp-based frame

drop_self proc near

input_filename= byte ptr -104h

push    ebp
mov     ebp, esp
sub     esp, 104h
lea     eax, [ebp+input_filename]
push    104h           ; nSize
push    eax           ; lpFilename
push    0             ; hModule
call    GetModuleFileNameA
lea     eax, [ebp+input_filename]
push    eax           ; input_filename
call    drop_file     ; copy ourself!
pop     ecx
leave
retn
drop_self endp
```

From this, we can deduce that one of the ways that the malware spreads is by sharing itself via Kazaa as fake warez.

2. Email

By looking at the code XREFs to email related strings in the .text section, it is clear that this malware spreads itself by sending malicious emails via SMTP.

Due to the size, complexity, and since we already had a good idea of the purpose of this code, fully reversing the mail based spreading functionality was outside the scope of our initial analysis.

3 Mitigation

3.1 Detection

3.2 Recovery