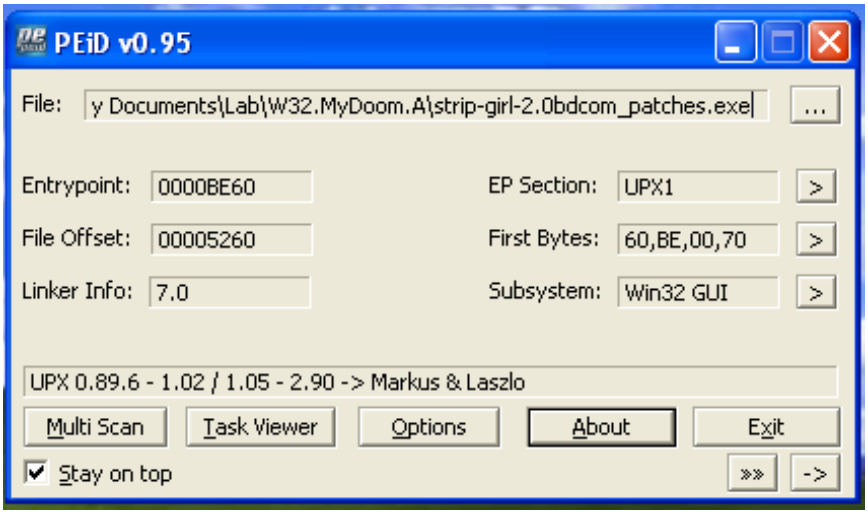
Contents

1 Introduction

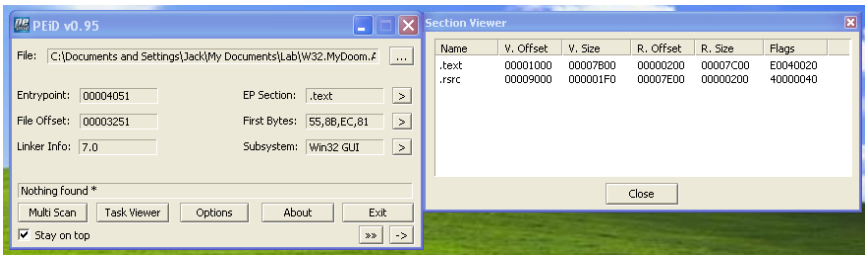
2 Static Analysis

2.1 Data

2.1.1 Unpacking

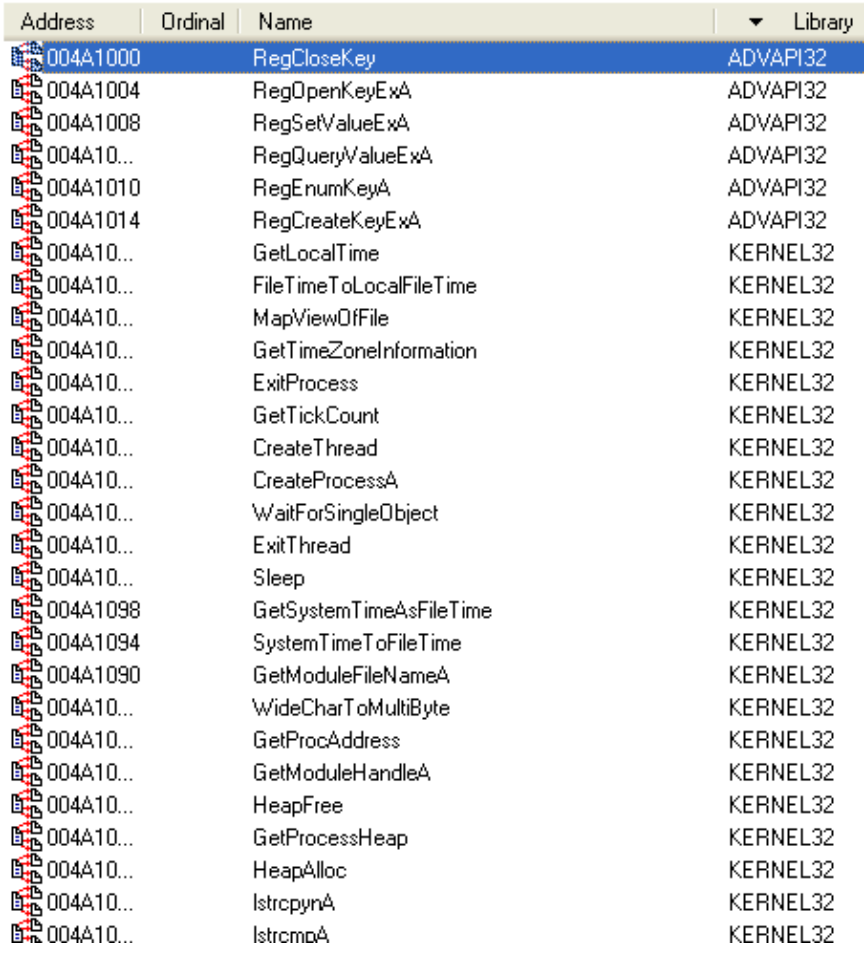


Initially opening the malware in PEiD, it appears to be packed with UPX.



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...		lstrcmpA	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

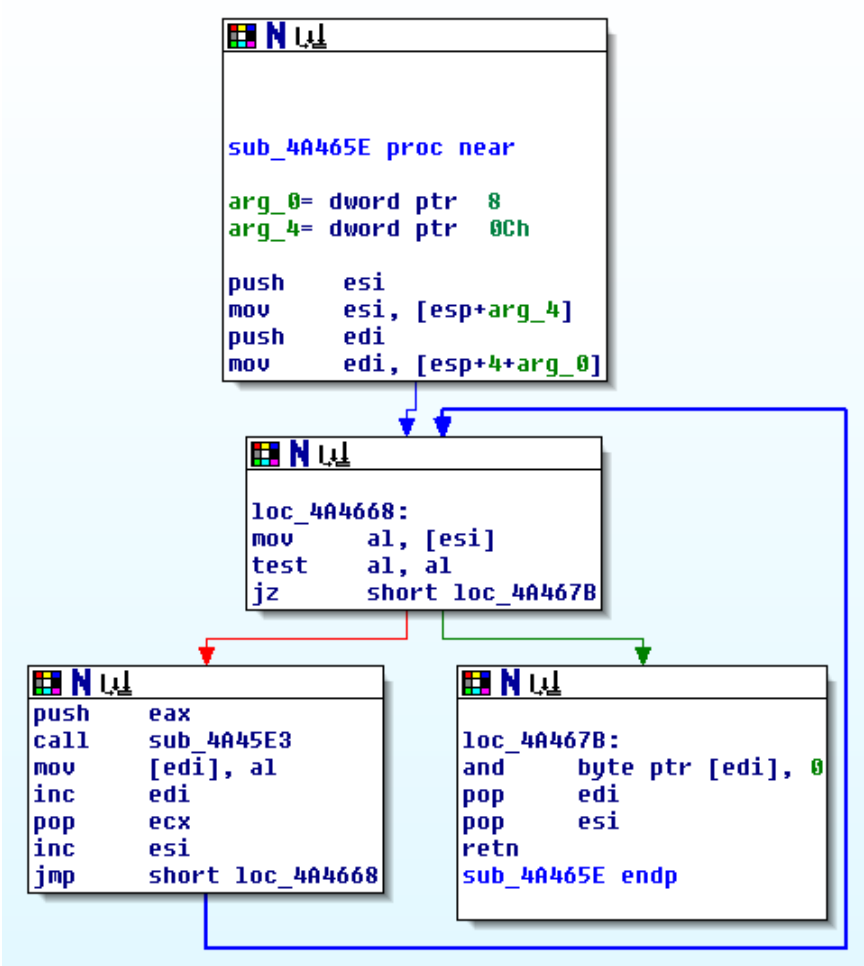
2.1.3 Strings

Address	Length	T...	String
".."	.text:00...	0000000D	C	fuvztncv.qyy
".."	.text:00...	00000044	C	Fbsgjner\\Zvpebfbsg\\Jvaqbjf\\Pheeraglrefvba\\Rkcybere\\PbzQyt32\\Ir...
".."	.text:00...	0000000F	C	FjroFvcpFzgkF0
".."	.text:00...	0000000C	C	gnfxzba.rkr
".."	.text:00...	00000008	C	GnfxZba
".."	.text:00...	0000002E	C	Fbsgjner\\Zvpebfbsg\\Jvaqbjf\\Pheeraglrefvba\\Eha
".."	.text:00...	0000000B	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	VagreargTrgPbaarpgrqFgngr
".."	.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-svany
".."	.text:00...	00000008	C	jvanzc5
".."	.text:00...	00000007	C	QyQve0
".."	.text:00...	00000018	C	Fbsgjner\\Xnmnn\\Genafsre
".."	.text:00...	0000000D	C	iphlpapi.dll
".."	.text:00...	0000000B	C	DnsQuery_A
".."	.text:00...	0000000B	C	dnsapi.dll
".."	.text:00...	00000011	C	GetNetworkParams
".."	.text:00...	00000007	C	sandra
".."	.text:00...	00000006	C	linda

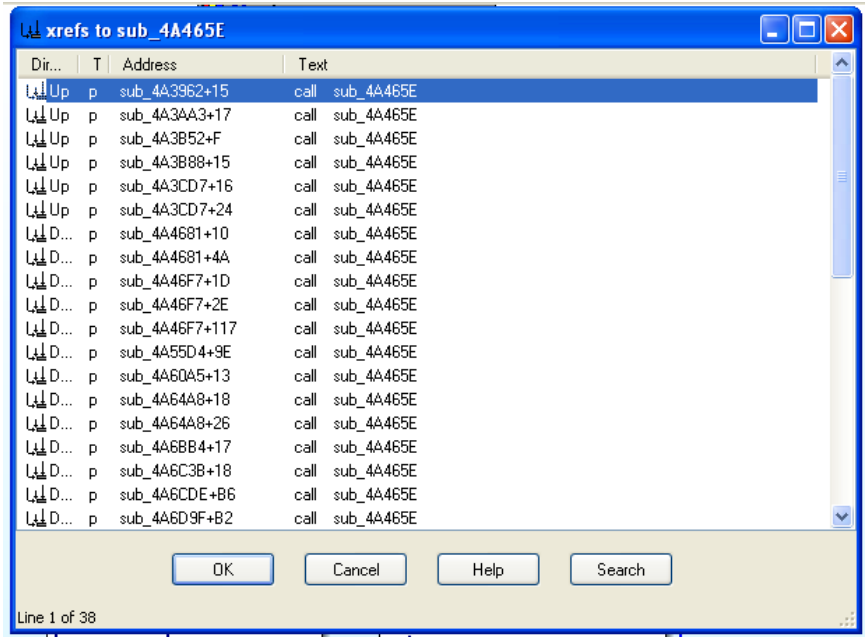
Looking at the strings, there seems to be many regular strings such as people's names, imports, and names, but also a few strings with all alphabetic characters and numbers. This points to some sort of encoding scheme used for the rest of the garbled strings, especially for the ones that look like paths,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

```
.text:004A2588 aAbcdefghijkl_0 db 'abcdefghijklmnopqrstuvwxyz',0 ; DATA XREF: sub_4A45E3+1A↓o
.text:004A25A3                 align 4
.text:004A25A4 aABcdefghijklmn db 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',0 ; DATA XREF: sub_4A45E3+A↓o
```

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.

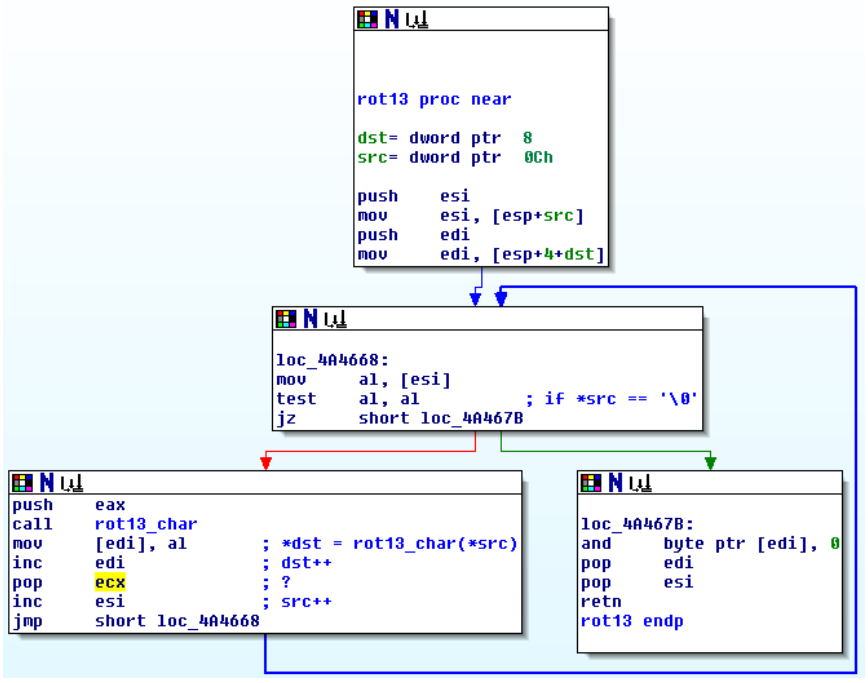


It seems to call the previous function in a loop. Searching for XREFs to *this* function, we know we've found the decoding function.



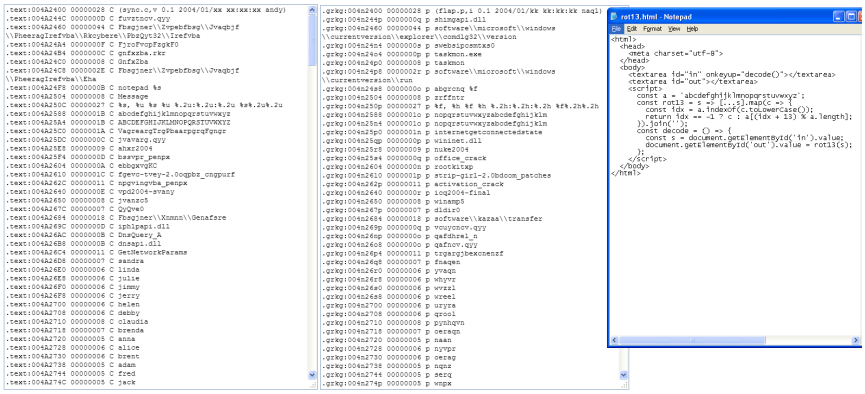

```
lea     eax, [ebp+name]
push    offset aJjj_fpb_pbz ; "jjj.fpb.pbz"
push    eax
call    sub_4A465E
```

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 have 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 aJvanzc5      ; DATA XREF: sub_4A46F7:loc_4A47FC↓r
                                        ; "jvanzc5"
                dd offset aVpd2004Svany ; "vpd2004-svany"
                dd offset aNpgvingvba_pen ; "npgvingvba_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"
                dd offset aSyma         ; "syma"
                dd offset aIcrosof      ; "icrosof"
                dd offset aMsn_         ; "msn."
                dd offset aHotmail      ; "hotmail"
                dd offset aPanda        ; "panda"
                dd offset aSopho        ; "sopho"
                dd offset aBorlan       ; "borlan"
                dd offset aInpris       ; "inpris"
                dd offset aExample      ; "example"
                dd offset aMydomai      ; "mydomai"
                dd offset aNodomai      ; "nodomai"
                dd offset aRuslis       ; "ruslis"
                dd offset a_gov         ; ".gov"
                dd offset aGov_         ; "gov."
                dd offset a_mil         ; ".mil"
                dd offset aFoo          ; "foo."
```

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.

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

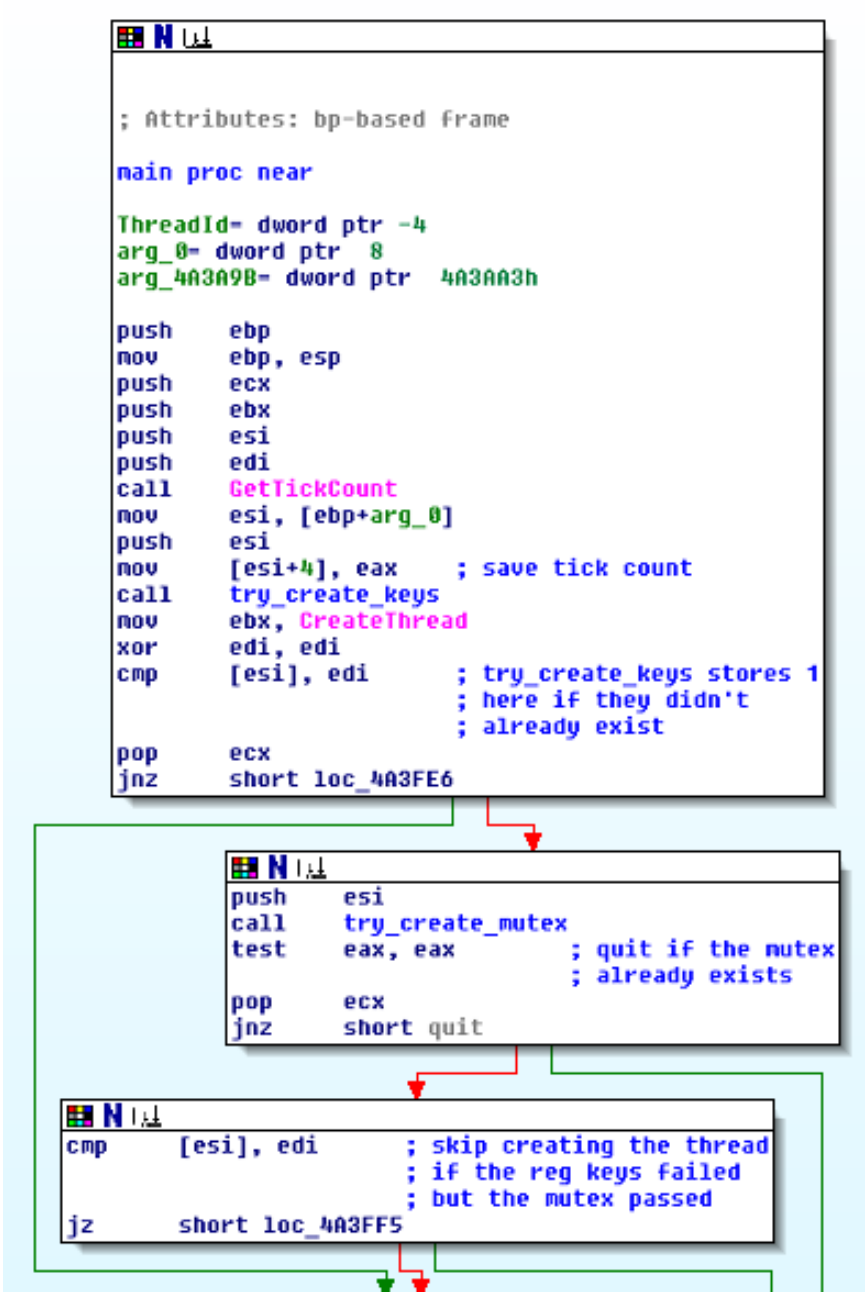
We will see later in the code where these dates are used.

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.

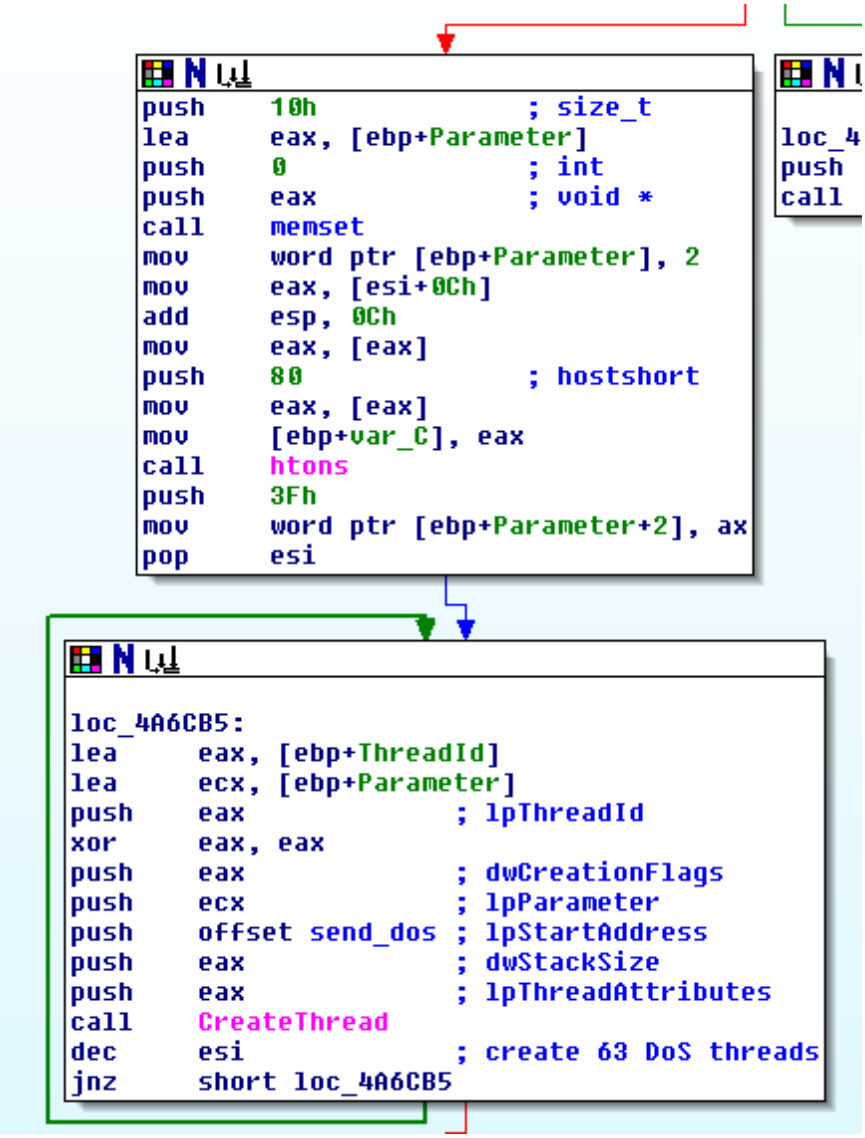
2.3 Malicious Functionality

2.3.1 DoS Attack

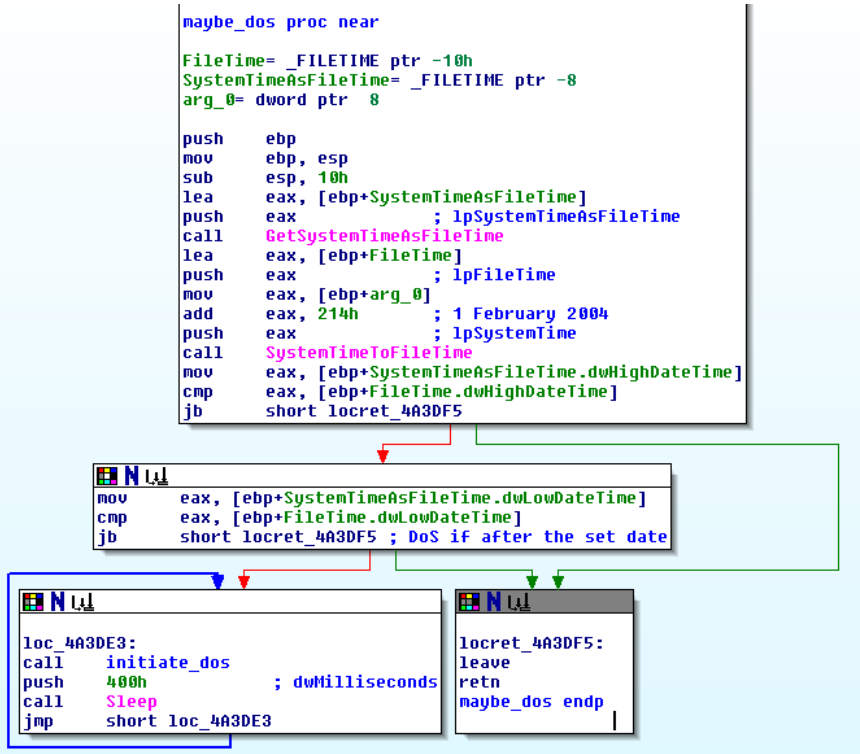
Looking at the XREFs for the string that contains an HTTP GET header to www.sco.com, we find a function that appears to execute a denial of service attack by repeatedly opening sockets and sending a GET request.

```
; 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 aTrgUggc1_1Ubfg ; "GET / HTTP/1.1\r\nHost: www.sco.com\r\n\r\n"
push    eax
call    rot13
pop     ecx
pop     ecx
push    0FFFFFFFFh      ; 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.



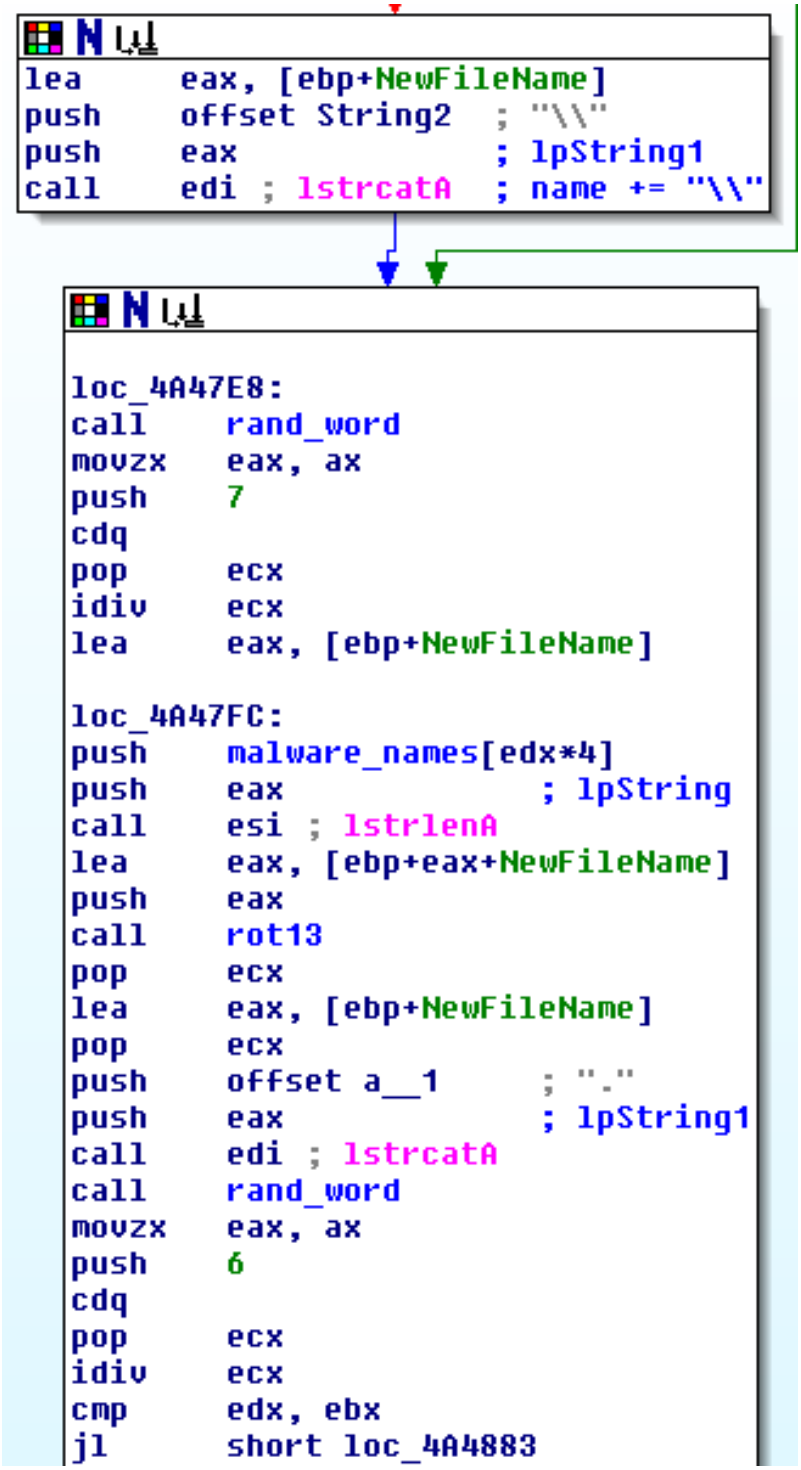
2.3.2 Replication

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.

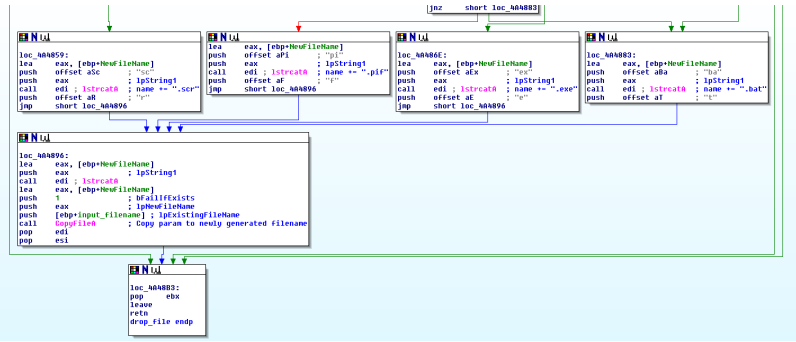
   The function reads the Kazaa shared directory from the registry, which makes contained files available to other users on the P2P file sharing platform.

```
; 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 aQyqve0  ; "QyQve0"
push    eax
call    rot13           ; ValueName = "DlDir0"
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
```

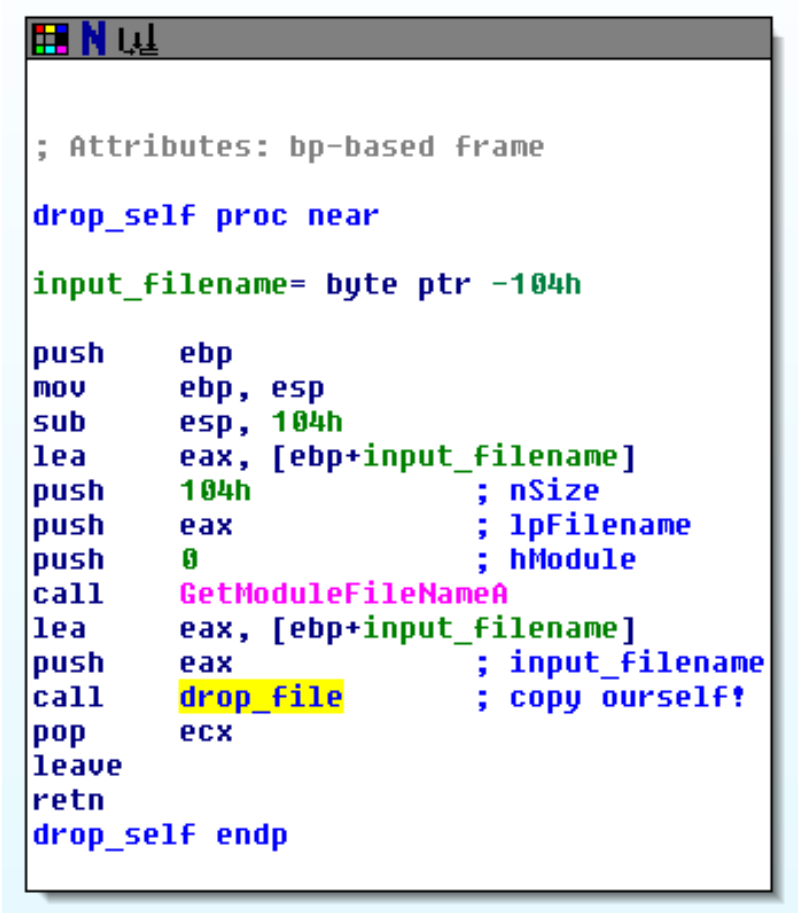
The function adds \\ and a random filename from the previously discovered list to the path,


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 ; lstrlenA
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