Артем Баранов
Ведущий вирусный аналитик ESET

Win32/Gapz - новый виток в развитии буткитов

В последние несколько лет увеличилось распространение вредоносных программ (буткитов), модифицирующих загрузочные сектора в процессе заражения системы. Среди самых известных представителей - TDL4, Olmasco и Rovnix. Каждый из них использует различные способы заражения жесткого диска – это либо модификация главной загрузочной записи (MBR), либо модификация первых секторов загрузочного раздела, т. е. VBR или IPL (первые сектора тома, куда передается управление из MBR - **Volume Boot Record/Initial Program Loader**). Наглядно эти семейства показаны на рисунке ниже.

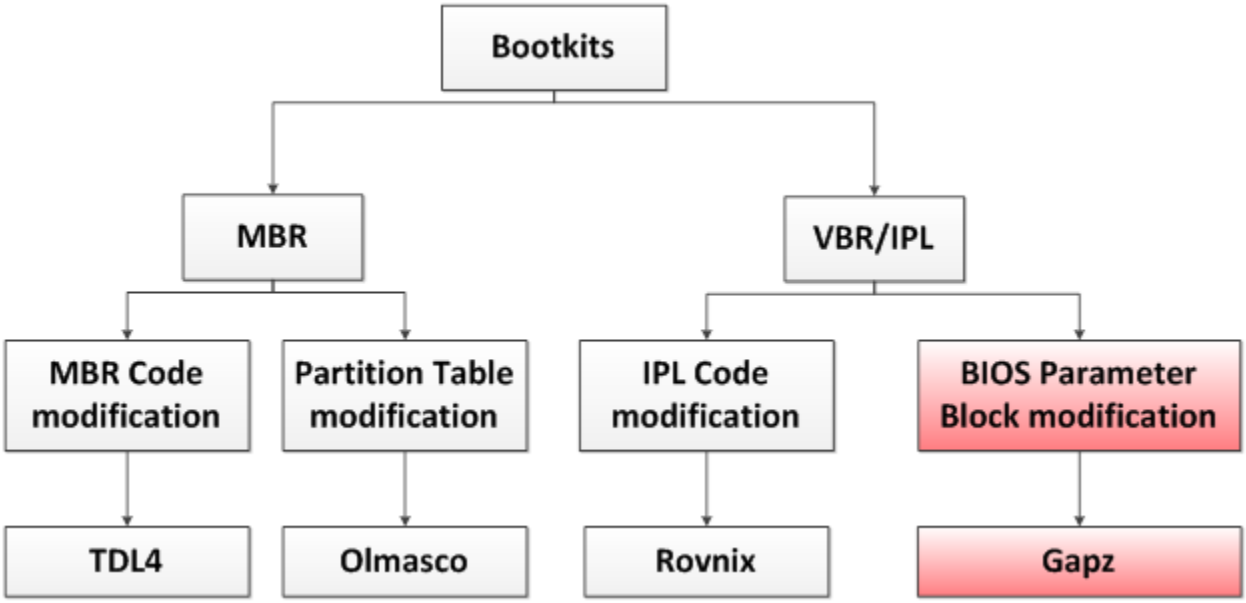

Рис. 1. Схема различных семейств буткитов и методов заражения диска.

Существует несколько причин, по которым злоумышленники используют буткиты в современных угрозах:

- Возможность запуска вредоносного кода раньше кода ОС, что дает неоспоримые преимущества и позволяет контролировать процесс загрузки системы.
- Как следствие, буткит позволяет обходить систему мониторинга целостности ключевых компонентов ядра – PatchGuard (практически единственный способ обеспечить выживаемость руткита в x64-среде).
- Возможность глубокого сокрытия кода, делая его невидимым для AV-сканеров.
- Буткит имеет посекторную архитектуру хранения своего тела на диске, что дает возможность выносить вредоносный код и код полезной нагрузки далеко за пределы файловой системы и даже разделов диска, делая почти невозможным его обнаружение.
- Безопасная установка руткита в системе.

В отчете ESET по угрозам и трендам за 2012 год мы указали, что буткиты являются одним из ключевых трендов среди вредоносного ПО. Мы также не обошли стороной и Win32/Gapz, так как он содержит ряд технических особенностей, которые делают его действительно интересным. Александр Матросов и Евгений Родионов проделали большую работу, занимаясь анализом данного буткита. Наш сегодняшний пост посвящен этому анализу.

Дроппер

Начнем с дроппера – компонента, который является изначальным носителем кода буткита и отвечает за его установку в системе. Мы детектируем его как: **Win32/Gapz.X**, X-версия. Нами обнаружены три его версии – A, B и C. На рисунке 2 приведены их характеристики:

Название объекта	Дата компиляции	LPE эксплойт	Объект для заражения
Win32/Gapz.A	11/09/2012 30/10/2012	CVE-2011-3402 CVE-2010-4398 COM Elevation	VBR
Win32/Gapz.B	06/11/2012	CVE-2011-3402 COM Elevation	отсутствует
Win32/Gapz.C	19/04/2012	CVE-2010-4398 CVE-2011-2005 COM Elevation	MBR

Рис. 2.

В соответствии с нашими наблюдениями, первая известная версия дроппера была скомпилирована в апреле прошлого года и содержала много отладочной информации, т. е. не предназначалась для массового распространения. Скорее всего, Win32/Gapz начали массово распространять в конце лета или начале сентября 2012 года. Для поднятия своих привилегий в системе Win32/Gapz использует LPE-эксплойты и COM Elevation метод.

В процессе анализа мы обнаружили, что заражению Win32/Gapz подвержены: 32-битные системы Windows XP SP2 и старше (кроме Windows Vista и Vista SP1), а также 64-битные Windows XP SP2 и старше. Обсуждаемая версия дроппера Win32/Gapz способна заражать Windows XP и Windows 7, включая x64 версии, однако на Windows 8 буткит-часть не работает должным образом, и после заражения надлежащая к исполнению в режиме ядра часть не исполнялась.

```
GetVersionExA(&VersionInformation);
if ( VersionInformation.dwMajorVersion == 5 )
{
  if ( VersionInformation.wServicePackMajor < 2u || (LOBYTE(x64) = ::is_x64(-1), x64) )
    ExitProcess(0);
  v1 = 0;
}
else
{
  if ( VersionInformation.dwMajorVersion != 6 )
    ExitProcess(v1);
  v1 = 0;
  if ( !VersionInformation.dwMinorVersion && VersionInformation.wServicePackMajor < 2u )
    ExitProcess(v1);
}
```

Рис. 3. Часть кода дроппера, проверяющая версию ОС.

Дроппер, устанавливающий буткит в систему, тщательно продуман и способен обойти современные проактивные защиты (HIPS), а также поднять свои привилегии до уровня системы. Кроме того, он содержит хитрый метод внедрения кода в адресное пространство процесса. Файл дроппера экспортирует из себя несколько функций, которые указаны ниже на рисунке.

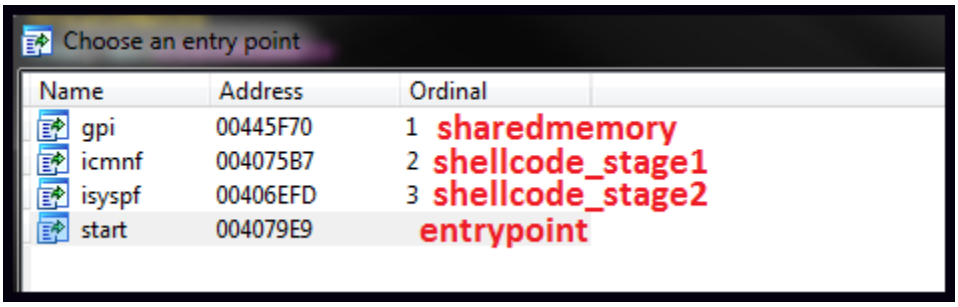

Рис. 4. Функции, экспортируемые исполняемым файлом дроппера.

Есть три экспортируемые функции, на которые следует обратить внимание: *start*, *icmnf* и *isyspf*. Краткое описание:

- *start* - точка входа в дроппер, осуществляет его внедрение в адресное пространство доверенного процесса explorer.exe;
- *icmnf* - отвечает за повышение (эскалацию) привилегий;
- *isyspf* - выполняет заражение жертвы кодом буткита.

Код дроппера использует специальную секцию, которая спроецирована в адресное пространство процесса explorer. Через эту секцию он загружает шелл-код в этот процесс и далее, с помощью специально сформированного API-вызова, производит его активацию. Соответственно, после того, как шелл-код активирован, он подгружает образ дроппера в адресное пространство процесса explorer, вызывает функцию повышения привилегий и инициирует процедуру заражения кодом буткита, записывая его на диск (рис. 5).

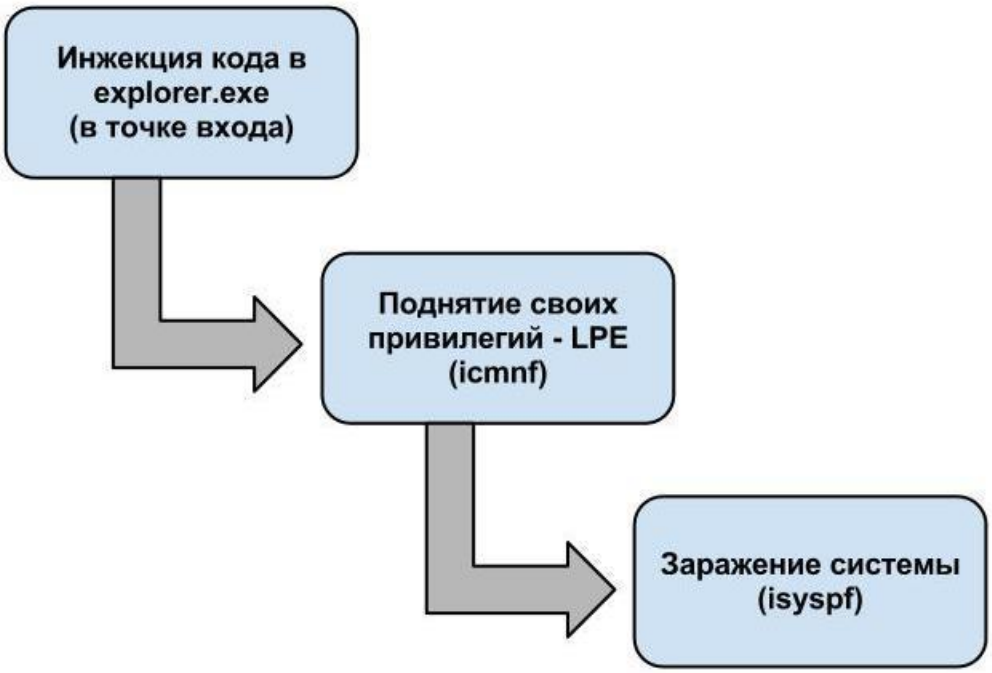

Рис. 5. Стадии выполнения дроппера и заражения жертвы кодом буткита.

После того, как дроппер заразил систему буткитом, его задача считается исполненной, и он удаляет свой файл с диска.

Вредоносный код MBR

Мы обнаружили две модификации буткита Win32/Gapz, которые различаются методами заражения диска жертвы. Самая ранняя модификация появилась в начале лета 2012 года – эта версия была нацелена на заражение MBR. Другая, более поздняя модификация, заражающая VBR, была замечена в конце осени прошлого года

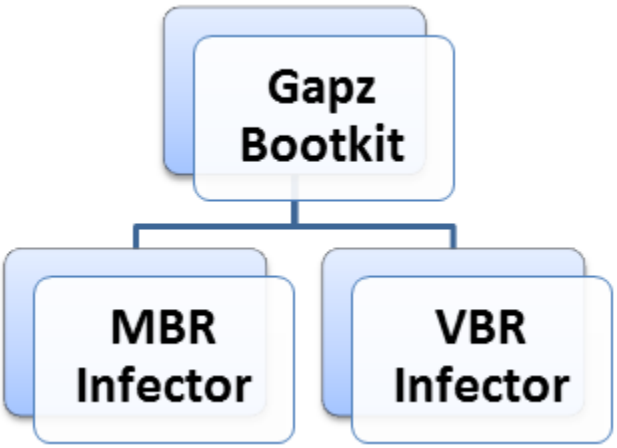

Рис. 6. Две модификации Win32/Gapz, нацеленные на заражения MBR и VBR.

Давайте подробнее рассмотрим более раннюю модификацию буткита, нацеленную на заражение MBR. В этом случае, код буткита можно разбить на несколько частей:

- вредоносный MBR;
- код режима ядра и полезная нагрузка, внедряемая в процессы.

Вредоносный код сохраняет свой код режима ядра и полезную нагрузку либо перед самым первым разделом, либо после последнего раздела на жестком диске. Такой подход очень похож на тот, который использовался в бутките Rovnix, за исключением того, что Rovnix заражает VBR.

Что касается буткит части Win32/Gapz, то в ней нет ничего необычного: как только код из вредоносного MBR исполнился, он восстанавливает оригинальный код в памяти и читает следующие секторы жесткого диска, содержащие код для последующего исполнения, на который и передается управление. Код буткита перехватывает обработчик прерывания 0x13, int 13h и отслеживает, таким образом, загрузку ниже перечисленных модулей ОС для установки туда перехватов:

- ntldr (на системах до Windows Vista)
- bootmgr (на системах Vista+)
- winload.exe (на системах Vista+)

Код буткита идентифицирует каждый из вышеперечисленных модулей, используя специальные последовательности байт. Ниже перечислен список функций, которые буткит перехватывает в этих модулях:

Имя модуля	Перехватываемая функция
ntldr	*BlLoadBootDrivers*
bootmgr	*Archx86TransferTo32BitApplicationAsm*
winload.exe	*OslArchTransferToKernel*
ntoskrnl.exe	*IoInitSystem*

Как только вредоносный код обнаруживает, что конкретный модуль читается с жесткого диска, он модифицирует его таким образом, чтобы вернуть себе контроль после переключения процессора в защищенный режим. Буткит устанавливает перехваты на загрузчик ядра ОС: это либо ntldr в ОС до Windows Vista, либо bootmgr в Vista и выше. В случае с bootmgr, он также перехватывает функцию *OslArchTransferToKernel* в winload.exe.

```
Hook_OslArchTransferToKernel proc far
                pusha
                mov     edi, eax
                mov     eax, 905A4Dh
                mov     ecx, 5DD000h
                std

loc_A2E:                                ; CODE XREF: Hook_OslArchTransferToKernel+15↓j
                repne scasb             ; search for kernel image base address
                jecxz   short loc_A64
                cmp     [edi+1], eax
                jnz     short loc_A2E   ; search for kernel image base address
                cld
                inc     edi
                mov     ecx, [edi+3Ch]
                add     ecx, edi        ; ecx -> pe
                cmp     [ecx+IMAGE_NT_HEADERS.OptionalHeader.Magic], 10Bh
                jnz     short loc_A64   ; check if it is x86 module
                mov     ecx, [ecx+IMAGE_NT_HEADERS.OptionalHeader.SizeOfImage]
                call    $+5
                pop     esi
                add     esi, 1F3h       ; esi -> Address of the hook (c41)
                call    Get_IoInitSystem ; obtain address of IoInitSystem
                cmp     ebx, 0          ; ebx -> IoInitSystem
                jz      short loc_A64
                call    Hook_IoInitSystem ; hook IoInitSystem

loc_A64:                                ; CODE XREF: Hook_OslArchTransferToKernel+10↑j
                                        ; Hook_OslArchTransferToKernel+24↑j ...
                cld
                popa
                push    edx
                push    8
                push    eax
                retf
Hook_OslArchTransferToKernel endp ; sp-analysis failed
```

Рис. 7. Перехватчик функции *OslArchTransferToKernel* в winload.exe.

Следующий этап – установка перехвата на функцию *IoInitSystem*, которая вызывается в процессе инициализации ОС. Она перехватывается вредоносным кодом либо из ntldr, либо из winload.exe, в зависимости от версии ОС.

```
New_IoInitSystem proc near

var_C           = dword ptr -0Ch
var_8           = dword ptr -8
arg_0           = dword ptr  4

                push    [esp+arg_0]
                push    eax             ; return address after
                                        ; executing original IoInitSystem
                push    eax             ; address of the original IoInitSystem
                pusha

loc_C78:                                ; DATA XREF: sub_E23+35↓o
                pushf
                cld
                mov     eax, cr0
                push    eax
                and     eax, 0FFFEFFFFh ; clear write protection bit
                                        ; to enable patching
                mov     cr0, eax
                call    $+5
                pop     esi
                add     esi, 194h
                mov     eax, ds:(original_IoInitSystem - 0E1Fh)[esi] ; get original IoInitSystem
                mov     [esp+34h+var_C], eax
                mov     edi, [esp+34h]  ; edi -> return address
                sub     eax, edi
                mov     [edi-4], eax    ; restore hook in kernel
                lea     eax, (Post_IoInitSystem - 0E1Fh)[esi] ; get address of the routine
                                        ; which will be executed right
                                        ; after IoInitSystem
                mov     [esp+34h+var_8], eax ; overwrite return address
                pop     eax
                mov     cr0, eax
                popf
                popa
                retn
New_IoInitSystem endp ; sp-analysis failed
```

Рис. 8. Код перехвата, устанавливаемого на функцию *IoInitSystem*.

После того, как вредоносный код из *IoInitSystem* был исполнен, буткит восстанавливает модифицированные байты в образе ядра ntoskrnl и передает управление оригинальной *IoInitSystem*. Перед передачей управления оригинальному коду, буткит перезаписывает адрес возврата в стеке на свою функцию, которая, соответственно, будет исполнена по завершении исполнения *IoInitSystem*. С помощью такого трюка вредоносный код получает управление после того, как ядро ОС будет инициализировано.

Далее вредоносный код считывает остальную свою часть с жесткого диска и создает отдельный системный поток, который исполняет эти инструкции и, в конце, возвращает управление ядру. В данной части буткита, которая исполняется в режиме ядра, реализуется руткит-функционал, внедрение полезной нагрузки в процессы и взаимодействие с C&C сервером.

Вредоносный код VBR

Последняя модификация Win32/Gapz заражает VBR тома, который помечен как активный в MBR (Volume Boot Record – первые сектора тома, в которых прописана служебная информация, а также VBR-код, на который передается управление из MBR и который отвечает за дальнейшую загрузку ОС). Буткит использует оригинальный подход для

заражения VBR с последующей передачей управления своему коду. Для максимальной скрытности он модифицирует лишь несколько байт оригинальной VBR. Суть такого подхода заключается в том, что модификации подвергается значение поля «Hidden Sectors» в поле служебной структуры VBR, при этом оставляя код VBR и код IPL нетронутым.

IPL (Initial Program Loader) – код, на который передается управление после исполнения кода VBR; он отвечает за поиск загрузчика в рамках файловой системы тома и передает на него управление. В состав VBR включены следующие части:

- Bootstrap-код (VBR-код), отвечающий за загрузку IPL.
- BIOS Parameter Block (BPB) - структура данных, хранящая блок параметров NTFS.
- Текстовые строки, отображаемые пользователю в случае ошибки.
- 0xAA55 - стандартная двухбайтовая сигнатура, маркер служебного сектора.

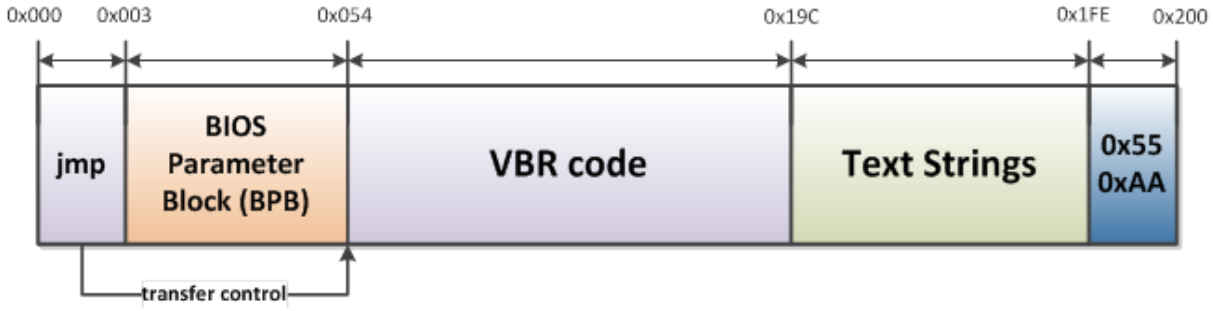

Рис. 9. Схема первого сектора VBR.

В случае с Win32/Gapz самым интересным местом для анализа является BPB, особенно поле «Hidden Sectors». Это поле содержит количество секторов, предшествующих IPL (т. е. то смещение до IPL в секторах, с помощью которого код из VBR определяет, куда передавать управление) и хранящихся на NTFS-томе, как показано ниже:

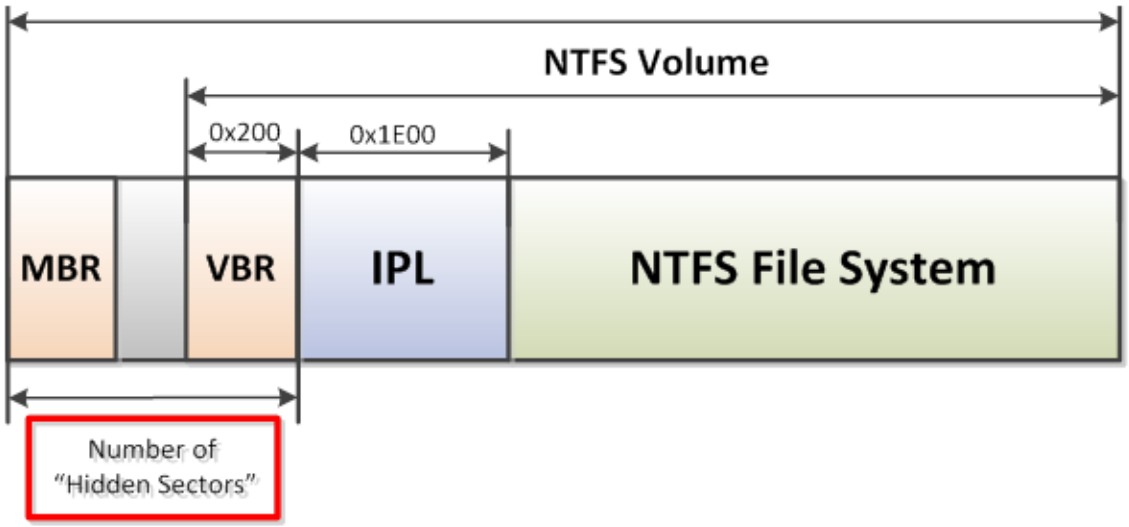

Рис. 10. Структура NTFS-тома.

Таким образом, в процессе загрузки на чистой системе VBR-код считывает 15 секторов, начиная со смещения, указанного в «Hidden Sectors», и передает туда управление. Этим

и пользуется буткит для передачи управления на себя. Он перезаписывает это значение, указывая смещение в секторах до своего вредоносного кода, хранящегося на диске. После заражения том выглядит так:

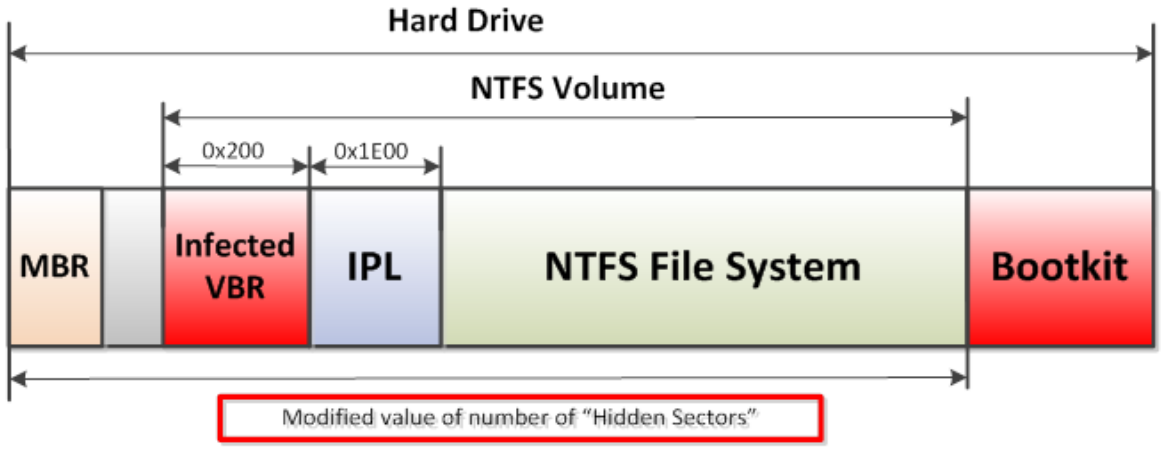

Рис. 11. Модифицированное буткитом значение «Hidden Sectors» приводит к тому, что код VBR передает управление на код буткита, а не на IPL.

В случае заражения системы, VBR-код вызывает на исполнение код буткита вместо легального IPL. Код буткита, как уже упоминалось, записывается либо перед самым первым разделом диска, либо после последнего. В остальном код буткита, по существу, ничем не отличается от версии с MBR-инфектором.

Вредоносный код режима ядра

Основное предназначение непосредственно той части, которая и называется буткитом, заключается в загрузке вредоносного кода режима ядра или руткита в системное адресное пространство, обходя ограничения, накладываемые ОС для такого привилегированного кода. Мы уже упоминали, что этот загружаемый буткитом код, содержит в себе руткит для сокрытия своего присутствия, механизм работы с управляющим сервером C&C, а также полезную нагрузку (payload), которая предназначена для внедрения в процессы.

В отличие от Rovnix, TDL4 и других распространенных буткитов, вредоносный код режима ядра в Win32/Gapz не имеет структуры исполняемого PE-файла. Вместо этого он структурирован особым образом. Код состоит из 12 объединенных между собой блоков, каждый из которых имеет заголовок – структуру, которая хранит служебную информацию о нем. Она имеет следующий вид:

```
struct GAPZ_BASIC_BLOCK_HEADER
{
    //Константа, которая используется для получения
    //адресов функций, которые располагаются в блоке
    unsigned int ProcBase;
    unsigned int Reserved[2];
    //Смещение следующего блока
    unsigned int NextBlockOffset;
    //Смещение до функции, которая осуществляет
    //инициализацию блока
    unsigned int BlockInitialization;
    unsigned int Reserved1[3];
};
```

Каждый из блоков реализует определенный функционал: внедрение полезной нагрузки, взаимодействие с C&C серверами, самозащиту и так далее. Тема функционала кода режима ядра является достаточно объемной и выходит за рамки этого поста.

Bootkits vs. Microsoft ELAM

Теперь мы хотим остановиться на специальном средстве, которое в компании Microsoft решили использовать для борьбы с различного рода угрозами, в том числе руткитами и буткитами, которые пытаются загрузить себя в системе раньше других драйверов.

Это средство называется ELAM (Early Launch Anti-Malware Module) и поставляется в составе ОС, начиная с Windows 8. По сути ELAM – это драйвер, предоставляемый антивирусным вендором, которому гарантирован приоритет при загрузке драйверов режима ядра. С точки зрения же ядра ОС, ELAM представляет собой API для антивирусных драйверов, а также набор правил, которых такому драйверу следует придерживаться.

Одна из главных возможностей этого средства заключается в том, что он гарантированно позволяет AV-драйверу загружаться раньше всех прочих драйверов в системе и, следовательно, выходить за рамки обычных правил автозагрузки, регламентируемых для остальных драйверов.

Стоит отметить, что ELAM сам по себе не может стать эффективным средством для борьбы с буткитами, поскольку это все равно часть ядра ОС, а буткит получает управление гораздо раньше.

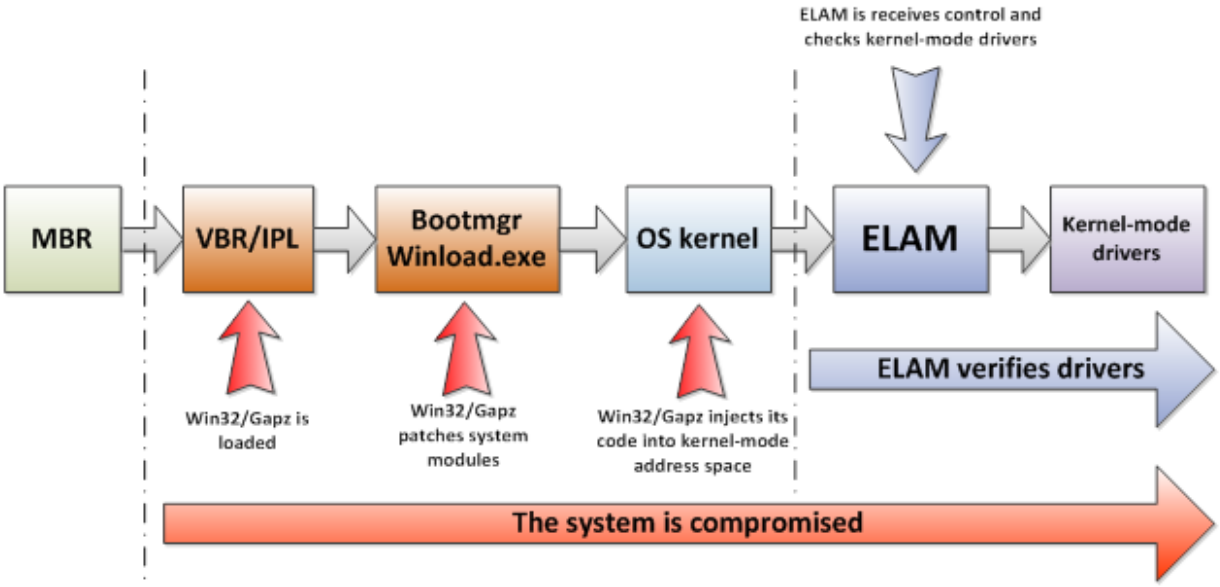

Рис. 10.

На рис. 10 мы видим, что буткит может компрометировать систему с активным ELAM, делая его бесполезным инструментом. Поскольку буткит загружается раньше инициализации ОС, он будет уже находиться в памяти на тот момент, когда ELAM получит управление.

В то же время, следует отметить, что ELAM является частью декларируемой Microsoft концепции «Безопасной загрузки» (Secure Boot). Согласно этой концепции, встроенное в UEFI программное обеспечение, которое получает управление при старте компьютера, может контролировать целостность и легитимность кода, на который передается управление в процессе выполнения кода, предшествующего выполнению основного кода ОС. Концепция Secure Boot, совместно с новым стандартом UEFI, призвана предотвратить компрометацию критичных структур данных ОС/UEFI со стороны буткитов.