АНТИВИРУСНАЯ ЗАЩИТА БИЗНЕС-КЛАССА

LoJax: первый известный UEFI руткит, используемый во вредоносной кампании

03 октября 2018 года

Кибергруппа Sednit, также известная как APT28, Strontium и Fancy Bear, работает как минимум с 2004 года. Считается, что группа стоит за рядом резонансных кибератак. Некоторые ИБ-компании и Министерство юстиции США назвали Sednit ответственной за взлом Национального комитета Демократической партии перед выборами в США в 2016 году. Группе приписывают взлом глобальной телевизионной сети TV5Monde, утечку электронных писем Всемирного антидопингового агентства (WADA) и другие инциденты. У Sednit множество целей и широкий спектр инструментов, некоторые из которых мы уже задокументировали ранее, но в этой работе мы впервые детально опишем применение UEFI руткита.

Краткий обзор

Мы обнаружили, что как минимум с начала 2017 года Sednit применяют троянизированную версию старого пользовательского агента программы для защиты устройств от краж разработчика Absolute Software – LoJack. Инструмент привлек внимание ИБ-специалистов по причине использования модуля UEFI/BIOS в качестве механизма для обеспечения персистентности. Мы назвали троянизированную версию этой программы **LoJax**.

Присутствие в зараженных системах известных инструментов Sednit вместе с образцами LoJax, а также тот факт, что некоторые командные серверы, используемые троянизированными агентами, ранее были частью инфраструктуры Sednit, позволяет нам с высокой долей уверенности связать UEFI руткит с этой группой.

Вместе с агентами LoJax обнаружены инструменты для чтения системной прошивки UEFI, и в одном из случаев этот инструмент мог делать дамп, патчить и переписывать часть системной SPI флеш-памяти. Конечной целью инструмента является установка вредоносного модуля UEFI в систему, защита SPI флеш-памяти которой уязвима или неправильно сконфигурирована.

Модуль UEFI отвечает за внедрение агента LoJax в систему, это первый идентифицированный UEFI руткит группы Sednit. Так как он находится в системной прошивке, он может пережить переустановку Windows и замену жесткого диска.

Был как минимум один случай, когда этот руткит был успешно установлен в системную SPI флеш-память. По нашей информации, это первый UEFI руткит, обнаруженный in-the-wild.

Введение

Группа Sednit использует ряд семейств вредоносного ПО. Подробное описание часто используемых инструментов группы мы приводили в отчете.

Мы отслеживаем активность Sednit на протяжении нескольких лет и выпустили множество отчетов о работе группы – от описания уязвимостей нулевого дня до кастомных программ, таких как Zebrocy. Компоненты, описываемые в данном отчете, образуют отдельную группу.

UEFI руткиты описывались в докладах ИБ-компаний ранее. Известен, например, rkloader, который фигурировал в презентации об утечке данных в Hacking Team, и DerStarke, имплант в загрузке macOS EFI/UEFI, из документов Vault7. Мы знаем о существовании этих инструментов, однако отчетов о компрометации UEFI руткитами не выпускалось.

Теперь мы не только доказали использование in-the-wild прошивки с вредоносным модулем LoJax UEFI, но и обнаружили весь спектр инструментов, которые, скорее всего, использовались для его установки. Интересно отметить, что Sednit использует буткит DownDelph, применявшийся в 2013 и 2014 годах для персистентности Downdelph, одного из бэкдоров Sednit первого этапа. Идея схожа, но в новом варианте UEFI использование буткитов более не представляется возможным. Таким образом, эти два компонента существенно отличаются в поведении.

Данная работа разделена на три секции. В первой мы разберем ранние исследования безопасности LoJack/Computrace и возможности вредоносного применения программы. Второй раздел посвящен процессу исследования, которое в итоге привело нас к руткиту UEFI. Наконец, в третьем разделе мы опишем в деталях различные компоненты LoJax и то, как они обеспечивает персистентность в системе даже после переустановки Windows и замены жесткого диска.

Атрибуция

В то время как многие вендоры уже сделали заявления относительно группы Sednit в прошлом, ESET никоим образом не определяет геополитическую принадлежность. С момента публикации исследования 2016 года наша позиция не изменилась. Определить источник кибератаки на основе научного подхода – комплексная задача, находящаяся за пределами сферы компетенций специалистов ESET. То, что мы называем «группой Sednit» – это просто набор ПО и связанная сетевая инфраструктура, которую мы не можем авторитетно связать с какой-либо конкретной организацией.

Цели

В процессе исследования мы обнаружили небольшое число различных образов LoJax. На основании данных нашей телеметрии и изучения других программ Sednit, обнаруженных in-the-wild, мы уверены, что этот конкретный модуль редко использовался в сравнении с другими инструментами. Целями, в основном, стали государственные организации, расположенные на Балканах, в Центральной и Восточной Европе.

Ранние исследования Computertrace/LoJack

LoJack – ПО для защиты компьютеров от кражи и потери, разработанное корпорацией Absolute Software. Ранние версии агента известны под названием Computrace. Как следует из прежнего названия, после активации сервиса компьютер может обращаться к своему командному серверу – владелец будет оповещен о местонахождении устройства в случае пропажи или кражи.

Далее в разделе описывается прежняя архитектура LoJack. Злоумышленниками была троянизирована только старая версия ПО, поэтому мы сфокусируемся на ней. Кроме того, Absolute Software выпустила в мае 2018 года заявление о том, что уязвимости, описанные ниже, не влияют на работу последних версий их агентов.

Программа Computrace привлекла внимание ИБ-специалистов, поскольку использовала необычный метод обеспечения персистентности. Ее целью является защита от кражи, поэтому для нее важна устойчивость к переустановке ОС и замене жесткого диска – все это реализовано в модуле UEFI/BIOS, способном выжить после этих действий. Решение предустанавливается в прошивке значительной части ноутбуков разных производителей, пользователю нужно лишь активировать эту функцию. Активацию можно выполнить в BIOS, как показано на рисунке ниже.

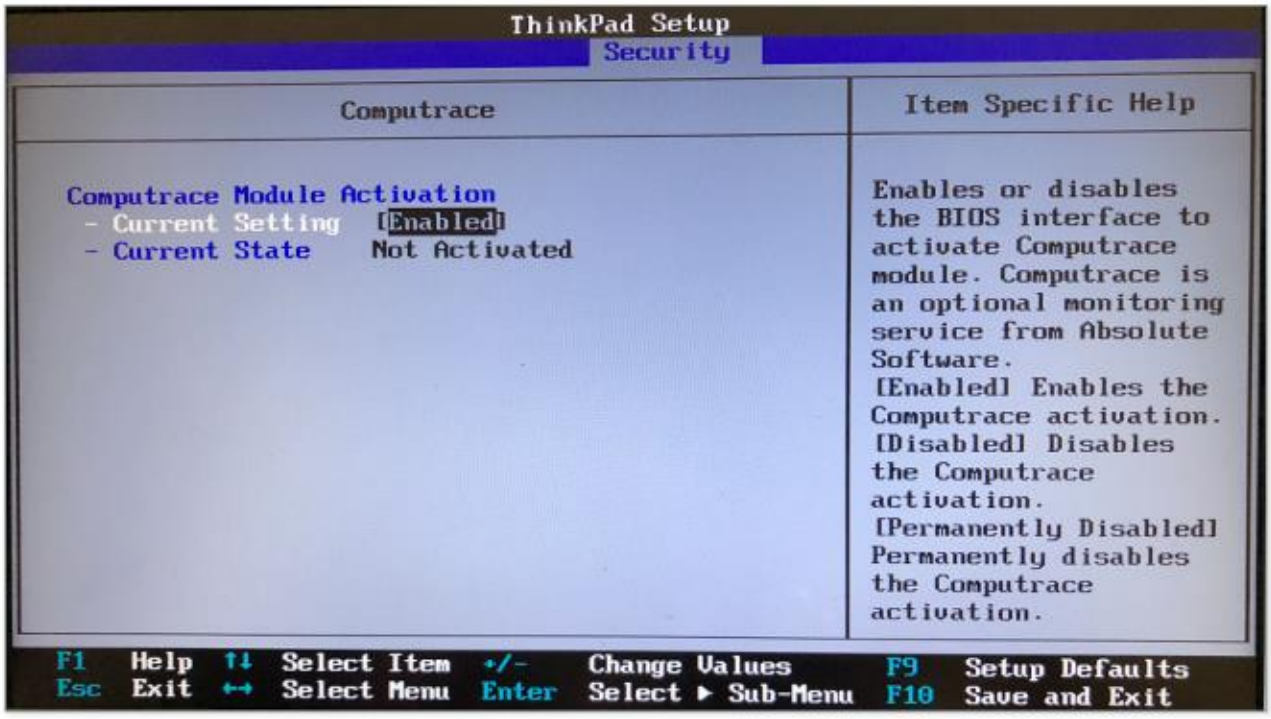

Рисунок 1. Активация Computrace в BIOS

Один из первых отчетов о реализации LoJack/Computrace был опубликован в 2009 году. Была раскрыта глобальная архитектура продукта, процесс сбрасывания модулем UEFI/BIOS пользовательского агента на диск и способ связи агента с веб-сервером, подконтрольным Absolute

Software. Схему можно понять из рисунка ниже.

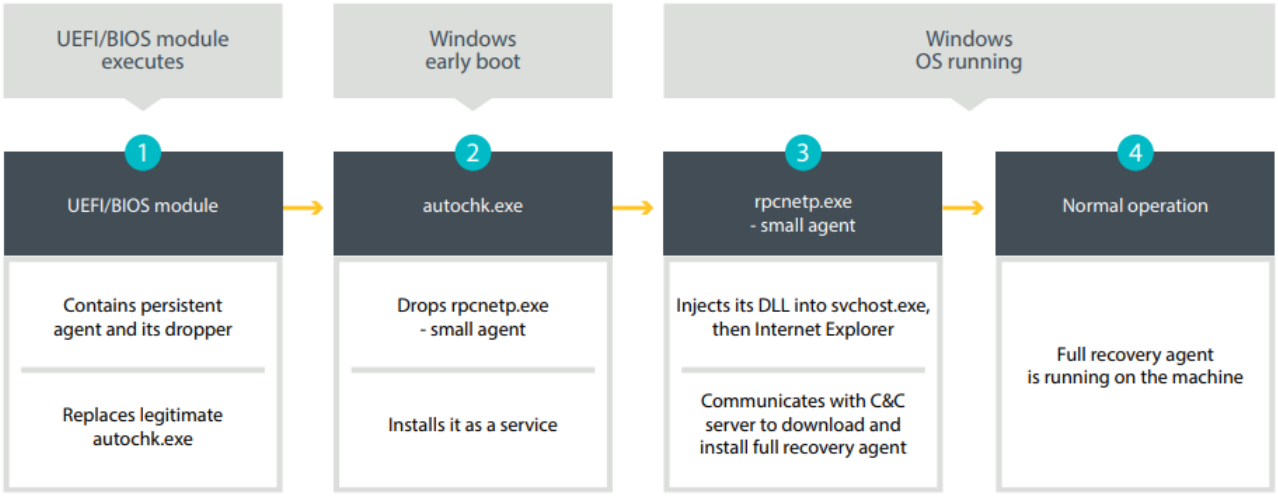

Рисунок 2. Механизм персистентности LoJack (примерно 2008 год)

Ниже приведено описание шагов, перечисленных выше:

1. В случае активации модуль UEFI/BIOS выполняется при загрузке. Он пытается найти раздел FAT/FAT32/NTFS. Затем с помощью драйвера NTFS он создает бэкап файла `autochk.exe` и переписывает его содержимое посредством дроппера, отвечающего за установку компонента пользовательского агента. Файл `autochk.exe` – исполняемый файл Windows, запускаемый на ранней стадии загрузки системы для проверки возможного повреждения жесткого диска.

2. Когда запускается измененный `autochk.exe`, его главной целью является внедрение мини-агента `rpcnetp.exe` и добавление его как службы, чтобы он запускался при каждой перезагрузке. Последний этап работы этого компонента – восстановление оригинальной версии `autochk.exe`.

3. Мини-агент `rpcnetp.exe` – небольшой исполняемый файл, целью которого является обеспечение работы основного агента. Если основной агент не работает, `rpcnetp.exe` пытается подключиться к командному C&C-серверу Absolute Software для его скачивания и исполнения. Сначала мини-агент сделает копию себя, затем внесет изменения в PE заголовок для преобразования в DLL. Эта библиотека затем загружается в память, вызывает процесс `svchost.exe` и инжектирует туда DLL. Далее запускается процесс `iexplore.exe` Internet Explorer, и DLL будет инжектирована в него. Последний процесс затем будет использован для связи через интернет. Инжект в сторонние процессы, выполняемый мини-агентом Computrace, часто встречается во вредоносном ПО и редко ассоциируется с легитимным софтом.

4. Теперь полнофункциональный агент работает в системе и может использовать функции Computrace для отслеживания и восстановления.

Описание этого процесса и сетевого протокола, задействованного между мини-агентом и C&C-сервером, было опубликовано в 2014 году. Ввиду отсутствия механизма аутентификации, если злоумышленники контролируют сервер, с которым связывается мини-агент, они могут заставить его скачать произвольный код. Есть несколько механизмов, позволяющих атакующим связаться напрямую с мини-агентом. Наиболее важный для нас использует способ получения мини-агентом адреса C&C-сервера. По факту эта информация хранится в файле конфигурации в самом исполняемом файле.

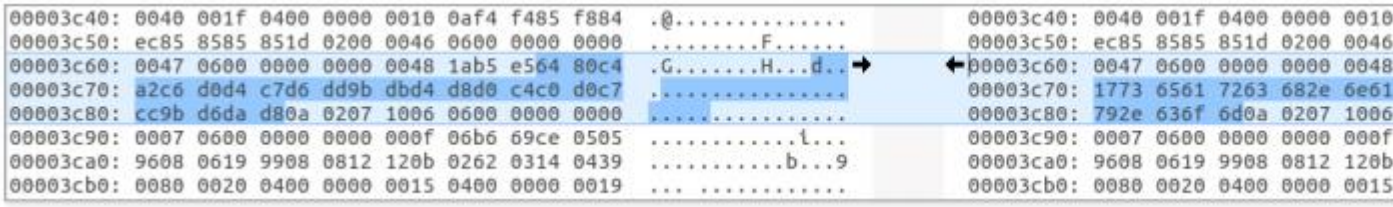

Рисунок 3. Зашифрованный файл конфигурации LoJack с частичной расшифровкой справа

На рисунке показан файл конфигурации мини-агента LoJack. Использованный метод «шифрования» — простой XOR с однобайтовым ключом. Ключ 0xB5 одинаков для всех изученных мини-агентов. Как видно из рисунка, домен C&C задан в файле. Четыре предшествующие байта содержат IP адрес командного C&C-сервера. В отсутствие валидации по содержимому файла конфигурации, злоумышленники с правами на запись в `%WINDIR%` могут поменять его содержимое так, что мини-агент будет связываться с их командным сервером, вместо легитимного. Понимая сетевой протокол, можно заставить мини-агент скачать и исполнить произвольный код. Эти риски давно известны, но до недавнего времени механизм не был использован на практике.

LoJack превращается в LoJax

В мае 2018 года в блоге Arbor Network были описаны троянизированные образцы мини-агента LoJack, `rpcnetp.exe`. Их жестко прописанные сетевые настройки были изменены таким образом, что вредоносные образцы устанавливали связь с C&C-сервером атакующих вместо легитимного сервера Absolute Software. Некоторые домены из числа обнаруженных в троянизированных образцах встречались и ранее – они использовались в конце 2017 года в качестве доменов C&C-серверов для SedUploader – бэкдора первого этапа кибергруппы Sednit. На рисунке ниже показан пример измененной конфигурации в одном из мини-агентов LoJax.

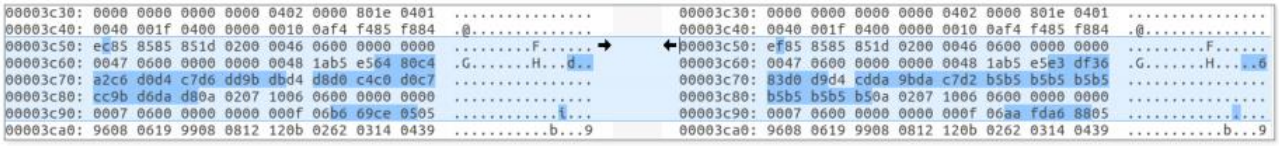

Рисунок 4. Слева – легитимный файл конфигурации, справа – измененный

Различия между легитимным и троянизированным агентами крайне малы, выше приведены почти все. Образцы мини-агента LoJax, которые мы смогли обнаружить – троянизированные версии одного и того же образца мини-агента Computrace, `rpcnetp.exe`. У всех них идентичные временные метки компиляции и лишь несколько десятков байтов отличаются от оригинала. Помимо изменений в файле конфигурации, есть различия в переменных таймера, определяющих интервалы между соединениями с командным C&C-сервером.

На момент публикации мы обнаружили различные мини-агенты LoJax, используемые в атаках на различные организации на Балканах, в Центральной и Восточной Европе, но у нас не было идей насчет способа их установки. Очевидным объяснением была бы установка с помощью одного из известных бэкдоров Sednit. Не стоит забывать, что LoJack, как хорошо себя зарекомендовавший инструмент, был внесен в белый список многими антивирусными вендорами. Таким образом, даже если в данной кампании использовался только мини-агент, который не выживал после переустановки Windows, у него все равно было преимущество – меньше вероятности детектирования в качестве вредоносного ПО. Но что, если компрометация была еще глубже, и атакующие попытались скопировать LoJack, чтобы дойти до системной прошивки?

Охота за компонентом нижнего уровня

Мы зафиксировали атаки LoJax, нацеленные на несколько организаций на Балканах, в Центральной и Восточной Европе. Во всех них нам удалось найти следы вредоносного ПО Sednit, включая:

- SedUploader, бэкдор первого этапа
- XAgent, флагманский бэкдор Sednit
- Xtunnel, сетевой прокси-инструмент, который может передавать любой сетевой трафик между C&C-сервером в интернете и конечным компьютером в локальной сети

Мы обнаружили следы инструментов Sednit в большинстве изучаемых систем, ставших целями LoJax, а также пару систем, где присутствовал только LoJax. Можно предположить, что в некоторых случаях LoJax применялся как отдельный инструмент, вероятно, в качестве дополнительного бэкдора для восстановления доступа операторов Sednit к сети.

Бэкдор XAgent как правило скидывает дополнительные модули в скомпрометированную систему, поэтому на ум сразу приходит, что образцы LoJax доставлялись тем же способом, без каких-либо других механизмов. Можно было бы предположить, что Sednit позаимствовали у решения LoJack только мини-агент. Однако вскоре после начала анализа мы нашли несколько улик, указывающих на то, что источник вдохновения был несколько обширнее.

Драйвер RWEverything (RwDrv) и info_efi.exe

Первая улика была найдена благодаря кастомному инструменту злоумышленников, который при выполнении выгружал информацию о настройках системы нижнего уровня в текстовый файл. Инструмент был обнаружен вместе с некоторыми образцами LoJax. Следующий рисунок показывает фрагмент файла с логами этого инструмента под логичным названием `info_efi.exe`.

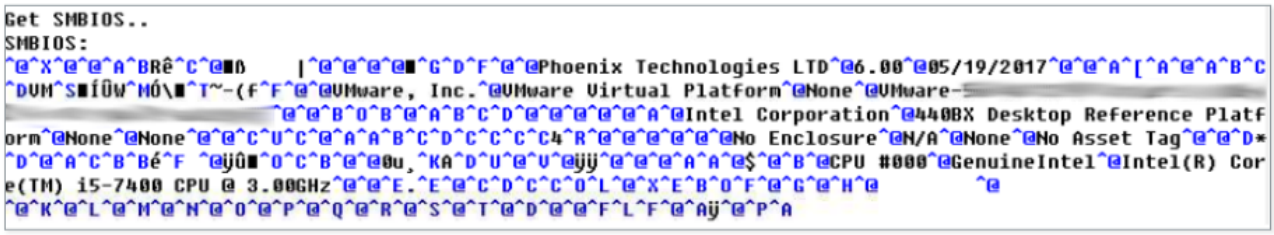

Рисунок 5. Выдержка из файл-логов, генерируемых info_efi .exe

Для чтения такого типа информации инструмент имеет в составе драйвер под названием `RwDrv.sys`. Драйвер ядра идет вместе с RWEverything, бесплатной утилитой, доступной в сети, которая может применяться для прочтения информации по почти всем настройкам нижнего уровня, в том числе интерфейсу PCI Express, памяти, PCI Option ROMs и т.д. Драйвер ядра является легитимным ПО и подписан действительным сертификатом.

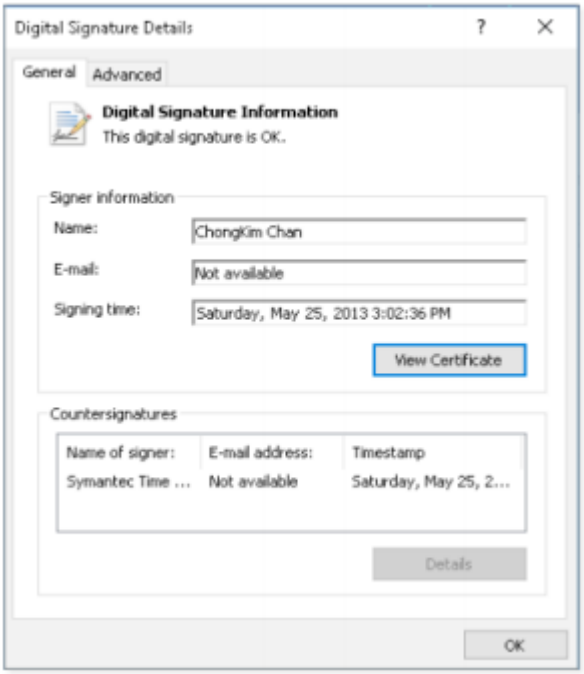

Рисунок 6. Сертификат подписи кода RwDrv.sys

ПО RWEverything идет в комплекте с графическим пользовательским интерфейсом, позволяющим получить доступ ко всей этой информации.

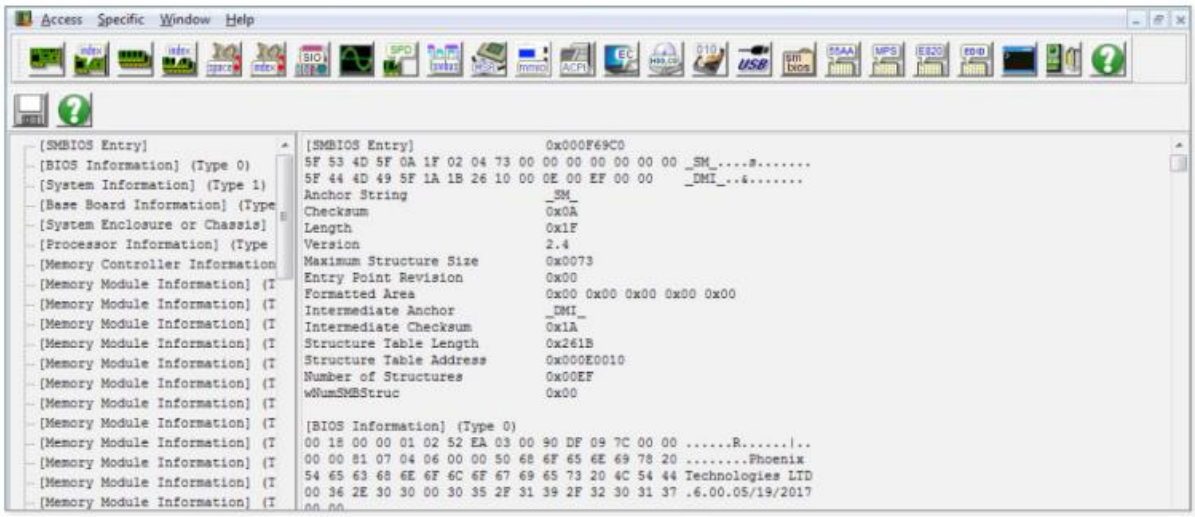

Рисунок 7. Скриншот интерфейса RWEverything

Обнаружение инструмента `info_efi` было первым знаком того, что UEFI модуль LoJax может существовать. При попытке обновить системную прошивку важно иметь информацию о ее вендоре, версии и т.д. Учитывая наличие уязвимостей, позволяющих пользовательским процессам получать доступ и изменять содержимое SPI флеш-памяти там, где хранятся модули UEFI, получение данных о системной прошивке является первым шагом к успешной атаке.

Финальной зацепкой, позволившей нам найти первый UEFI-руткит группы Sednit, стали два инструмента – для дампа SPI флеш-памяти и для записи в нее.

Дамп SPI флеш-памяти

Первым кусочком паззла стал файл под названием `ReWriter_read.exe`. Он содержал весь код, требуемый для дампа системной SPI флеш-памяти с помощью драйвера RWEverything, `RwDrv.sys`. Чтобы драйвер устройства произвел необходимые операции, инструмент для дампа должен отправить корректные коды управления вводом-выводом (IOCTL). В то время как `RwDrv.sys` поддерживает множество различных кодов IOCTL, инструмент-дампер и инструмент для записи, описанные в этом и следующем разделе, используют лишь четыре из них.

RwDrv.sys: поддерживаемые коды IOCTL:

0x22280c — пишет в область памяти, отведенной под ввод-вывод
0x222808 — считывает область памяти, отведенной под ввод-вывод
0x222840 — считывает dword из заданного регистра конфигурации PCI
0x222834 — пишет байт в заданный регистр конфигурации PCI

`ReWriter_read` сначала создает службу с встроенным драйвером ядра `RwDrv.sys` и записывает информацию о конфигурации UEFI/BIOS, соответствующие значения трех полей, содержащихся в регистре управления BIOS (BIOS_CNTL): BIOS Lock Enable (BLE), BIOS Write Enable (BIOSWE) и SMM BIOS Write Protect Disable (SMM_BWP). Несмотря на то, что `ReWrite_read` не использует эти значения, в следующих разделах мы поясним, почему эти поля представляют интерес для данного инструмента.

Следующей задачей инструмента является получение базового адреса области памяти BIOS в SPI и ее размер. Эта информация содержится в регистре главного интерфейса SPI как BIOS Flash Primary Region. Все регистры отображены в памяти в Root Complex Register Block (RCRB), базовый адрес которой может быть получен путем прочтения нужного конфигурационного регистра PCI Configuration Register. `ReWriter_read` получает этот адрес через использование RwDrv IOCTL 0x22840 и прочтение корректного отступа (в нашем случае 0xF0). Как только базовый адрес области BIOS и ее размер известны, инструмент для дампа читает соответствующее содержимое SPI флеш-памяти и пишет его в файл на диске. Процесс чтения SPI флеш-памяти проиллюстрирован в следующем рисунке. Определения сокращений приведены ниже, в глоссарии.

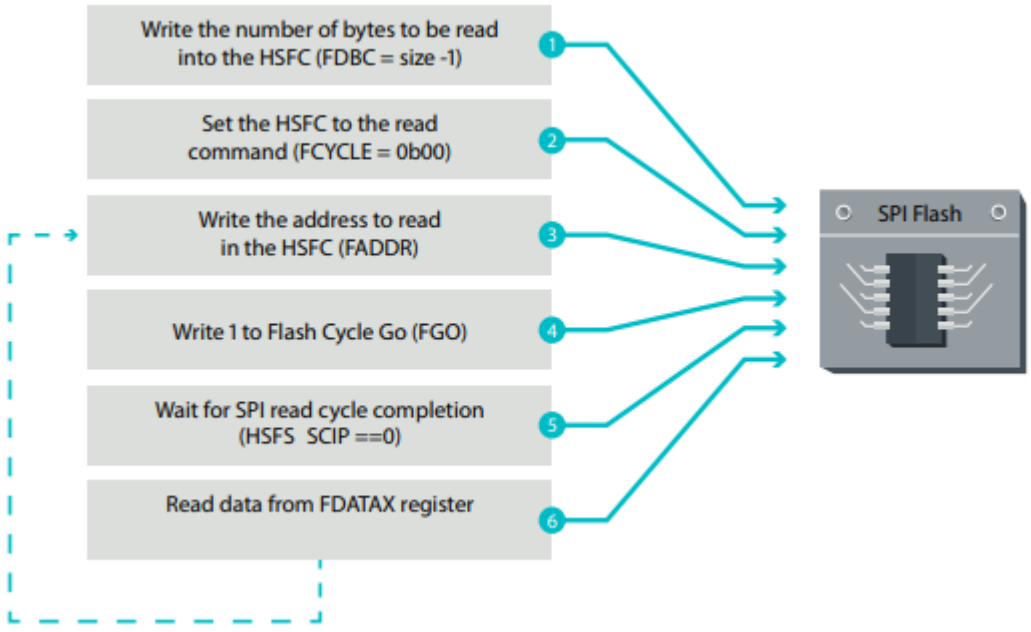

Рисунок 8. Рабочая последовательность чтения из SPI флеш-памяти

Кроме двух первых шагов, исполняемых только один раз, операции повторяются в цикле, пока не будут прочтены все данные из SPI флеш-памяти. Процесс также хорошо описан здесь.
Затем `ReWriter_read` проводит валидацию размера слитого образа. Он парсит дескриптор памяти образа для получения диапазона BIOS, областей Gigabit Ethernet (GbE) и Management Engine (ME). Добавление размеров этих трех областей позволяет инструменту-дамперу вычислить все содержимое SPI флеш-памяти. Если размер совпадает с размером, полученным из чтения области регистра BIOS Flash Primary, образ считается верным.

Патч прошивки UEFI

Вторым кусочком паззла был файл под названием `ReWriter_binary.exe`. В этом файле содержится доказательство того факта, что Sednit добрались до прошивки. Файл содержит код для применения патча выгруженного образа UEFI и обратной записи троянизированной версии в SPI флеш-память. В данном разделе опишем, как устроен этот бинарный файл.

После того, как содержимое флеш-памяти выгружено и проверено указанным выше инструментом, вредоносный UEFI модуль добавляется к образу. Для этого образ UEFI нужно проанализировать для выделения нужной информации.

Хранящиеся в UEFI образе данные разложены по томам посредством файловой системы (FFS). Как следует из названия, это специальная файловая система для хранения образов прошивки. Тома содержат файлы с идентификаторами GUID. Каждый файл обычно состоит из множества секций, одна из которых содержит собственно исполняемый PE/COFF, являющийся образом UEFI. Ниже для более простого понимания схемы приведен скриншот из UEFITool, опенсорсного проекта для работы с образами прошивки UEFI.

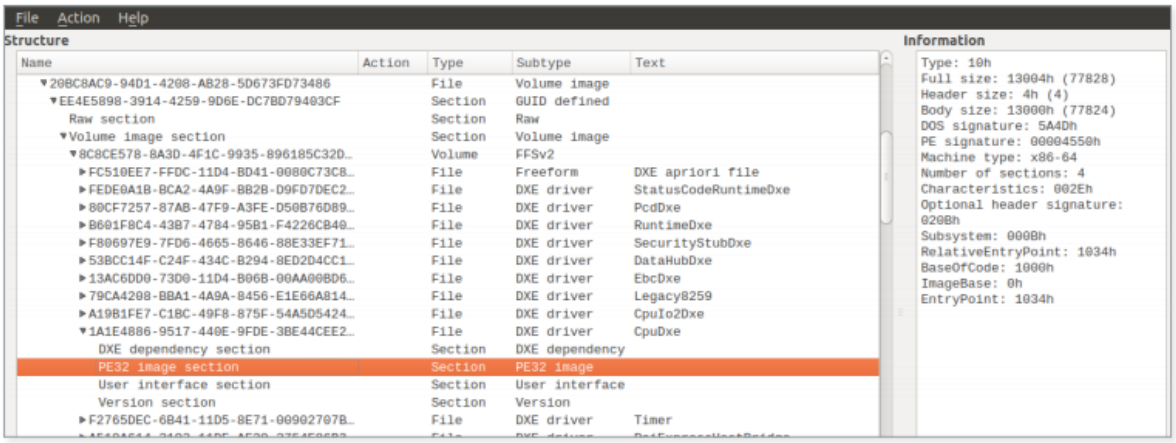

Рисунок 9. Пример загруженного в UEFITool образа прошивки UEFI

ReWriter_binary анализирует все тома прошивки, найденные в области BIOS SPI флеш-памяти и ищет определенные файлы:

- Ip4Dxe (8f92960f-2880-4659-b857-915a8901bdc8)
- NtfsDxe (768bedfd-7b4b-4c9f-b2ff-6377e3387243)
- SmiFlash (bc327dbd-b982-4f55-9f79-056ad7e987c5)
- DXE Core

```
if ( FileType == EFI_FV_FILETYPE_DRIVER )
{
  // If FileGuid == 8f92960f-2880-4659-b857-915a8901bdc8 (Ip4Dxe)
  if ( !((*&FileHeader->Name.Data4[4] - *&gIp4DxeGuid.Data4[4]) |
         (*&FileHeader->Name.Data2 - *&gIp4DxeGuid.Data2) |
         (FileHeader->Name.Data1 - gIp4DxeGuid.Data1) |
         (*FileHeader->Name.Data4 - *gIp4DxeGuid.Data4)) )
  {
    *Ip4DxeOffset = Index;
    Ip4DxeOffset[1] = 0;
    *Ip4DxeFileSize = FileSize;
    *Ip4DxeGuid = FileHeader->Name;
    Ip4DxeFound = 1;
  }
  // If FileGuid == 768bedfd-7b4b-4c9f-b2ff-6377e3387243 (NTFS)
  if ( !((*&FileHeader->Name.Data4[4] - *&gNTFSGuid.Data4[4]) |
         (*&FileHeader->Name.Data2 - *&gNTFSGuid.Data2) |
         (FileHeader->Name.Data1 - gNTFSGuid.Data1) |
         (*FileHeader->Name.Data4 - *gNTFSGuid.Data4)) )
  {
    *NtfsFileOffset = Index;
    NtfsFileOffset[1] = 0;
    *NtfsFileSize = *FileHeader->Size;
    *(NtfsFileSize + 2) = FileHeader->Size[2];
  }
  // If FileGuid == bc327dbd-b982-4f55-9f79-056ad7e987c5 (SmiFlash)
  if ( !((*&FileHeader->Name.Data4[4] - *&gSmiflashGuid.Data4[4]) |
         (FileHeader->Name.Data1 - gSmiflashGuid.Data1) |
         (*FileHeader->Name.Data4 - *gSmiflashGuid.Data4) |
         (*&FileHeader->Name.Data2 - *&gSmiflashGuid.Data2)) )
  {
    *SmiFlashOffset = Index;
    SmiFlashOffset[1] = 0;
  }
}
// If it's the DXE Core
else if ( FileType == EFI_FV_FILETYPE_DXE_CORE )
{
  *IsDxeCoreFirmwareVolume = 1;
```

Рисунок 10. Результат использования декомпилятора Hex-Rays в томах прошивки

Ip4Dxe и NtfsDxe – драйверы DXE. В UEFI прошивке драйверы DXE – образы PE/COFF, созданные либо для аппаратной абстракции, либо для организации работы служб для использования другими драйверами DXE или UEFI приложениями. Такие драйверы обнаруживаются и загружаются DXE Foundation через DXE диспетчер (ядро DXE) на ранней стадии процесса загрузки. После завершения этой фазы все службы, такие как OS loader, доступны для работы с UEFI приложениями. Обычно DXE драйверы хранятся в одном и том же томе. Однако DXE dispatcher может быть в отдельном.

`ReWriter_binary` ищет Ip4Dxe только для того, чтобы понять, содержит ли данный том драйверы DXE. Как мы опишем позднее, этот том становится кандидатом для установки вредоносного драйвера DXE driver. Он также ищет ядро DXE и добавляет том, в котором он расположен, в качестве другого кандидата на место под запись руткита. Свободное доступное пространство в каждом из этих томов хранится и позже используется для проверки достаточности для добавления вредоносного драйвера.

NtfsDxe – драйвер AMI NTFS DXE. Если он присутствует в томе прошивки, его расположение

сохраняется и позже используется, чтобы убрать этот файл из тома. В разделе, посвященном UEFI руткиту, мы увидим, почему он удаляет этот файл.

Что касается образа SmiFlash, то информация, относящаяся к нему, хранится, но в малвари нигде не используется. Что интересно, образ уязвим. Поэтому мы считаем, что операторы Sednit могут работать над использованием этих уязвимостей. Это может позволить им записывать в SPI флеш-память даже правильно сконфигурированных систем. Как мы расскажем далее, в текущем виде инструмент может записывать только в область BIOS неправильно сконфигурированных или достаточно старых систем (на материнских платах с чипсетами старше Platform Controller Hub, представленными примерно в 2008 году).

После выделения необходимых метаданных `ReWriter_binary` патчит дамп образа UEFI и добавляет вредоносный драйвер DXE. Сначала он создает заголовок файла (EFI_FFS_FILE_HEADER). Затем он выбирает том места назначения, руководствуясь расположением Ip4Dxe и ядра DXE, а также свободным местом, доступным в этих томах. `ReWriter_binary` встраивает сжатый раздел, содержащий образ PE и раздел User interface, определяющий человеко-читаемое имя файла: SecDxe. Сжатый раздел добавляется к заголовку файла и пишется в конец тома, в свободное пространство. На рисунке ниже показана структура – ее отображение в UEFITool.

```
▼682894B5-6B70-4EBA-9E90-A607E5676297    File       DXE driver     SecDxe
 ▼Compressed section                     Section    Compressed
   PE32 image section                    Section    PE32 image
   User interface section                Section    User interface
```

Рисунок 11. Вид в UEFITool файла SecDxe

Наконец, если драйвер NtfsDxe присутствует в образе, он будет удален. Файловая система прошивки хранит файлы и их содержимое последовательно, поэтому процесс достаточно прост:

- находится отступ до свободного места в конце тома
- поверх образа NtfsDxe пишется 0xFF байтов
- последующая часть тома прошивки копируется, начиная с отступа там, где располагался NtfsDxe
- остальная часть файловой системы заполняется 0xFF байтами, то есть свободным местом

Запись пропатченной прошивки обратно в SPI флеш-память

После успешного внесения изменений в образ прошивки следующий этап – ее запись обратно в SPI флеш-память. Перед тем, как погрузиться в этот процесс, нам необходимо охарактеризовать некоторые защиты от записи BIOS, важные в данном случае. Другие существующие механизмы, такие как BIOS Range Write Protection, остаются в стороне, так как их ReWriter_binary не проверяет.

Платформа применяет несколько защитных механизмов для блокирования неавторизованных попыток записи в область BIOS. Нужно сказать, что эти механизмы по умолчанию не включены. За их корректную настройку отвечает прошивка. Эти конфигурации представлены в регистре управления BIOS (BIOS_CNTL). Он содержит бит BIOS Write Enable (BIOSWE), который необходимо переключить на «1» для получения возможности записи в область BIOS флеш-памяти SPI. Так как платформа не должна позволять какие-либо попытки записи в область BIOS, есть еще один бит в BIOS_CNTL для защиты BIOSWE – это BIOS Lock Enable (BLE). Когда он выставлен, механизм должен блокировать бит BIOSWE и оставлять значение равное «0». Однако решение имеет уязвимость. Когда приходит запрос смены бита BIOSWE на «1», бит BIOSWE на «1», и лишь после этого платформа прерывает задачу с помощью System Management Interrupt (SMI), код этого SMI выполняет смену бита BIOSWE обратно на «0».

В данной версии решения есть множество проблем. Во-первых, обработчик SMI оставлен для разработчиков прошивки. Поэтому если в прошивке не реализован этот код, бит BLE оказывается бесполезен, так как назад на «0» бит BIOSWE выставляться не будет. Во-вторых, в таком случае мы имеем «уязвимость состояния гонки», которая позволяет полностью обойти этот механизм, даже если обработчик SMI реализован корректно. Для применения этой уязвимости атакующему необходимо запустить поток, непрерывно выставляющий BIOSWE на «1», в то время как другой поток должен писать данные в SPI флеш-память. Согласно работе Калленберга и Войтчука, эта атака работает на многоядерных процессорах и также может успешно применяться на одноядерных процессорах с включенной технологией Hyper-Threading.

Для решения этой проблемы в платформу был добавлен новый механизм защиты, сконфигурированный через BIOS_CNTL. Он введен в семействе чипсетов от Intel Platform Controller Hub (PCH). Если бит конфигурации выставлен, SMM BIOS Write Protect Disable (SMM_BWP) обеспечит возможность записи в область BIOS только в том случае, если все ядра работают в режиме System Management Mode (SMM) и BIOSWE выставлен на значение «1». Это эффективно защищает систему от «уязвимости состояния гонки», описанной выше. Однако, как и в случае с BLE, SMM_BWP нужно активировать со стороны прошивки. Поэтому прошивка, в которой неверно настроены эти механизмы, оставляет в системе риск предоставления неавторизованного права записи в область BIOS.

`ReWriter_binary` читает содержимое регистра управления BIOS, чтобы выбрать верный путь.

Сначала он проверяет, выставлен ли BIOSWE. Если это так, он переходит в фазу записи. Если BIOSWE отключен, он проверяет значение бита BLE. Если оно не установлено, он меняет значение бита BIOSWE и начинает записывать пропатченную прошивку. Если бит BLE установлен, он проверяет отключенное состояние SMM_BWP и применяет «уязвимость состояния гонки», описанную выше. Если бит SMM_BWP выставлен, операция завершается неуспешно. Рисунок ниже иллюстрирует процесс.

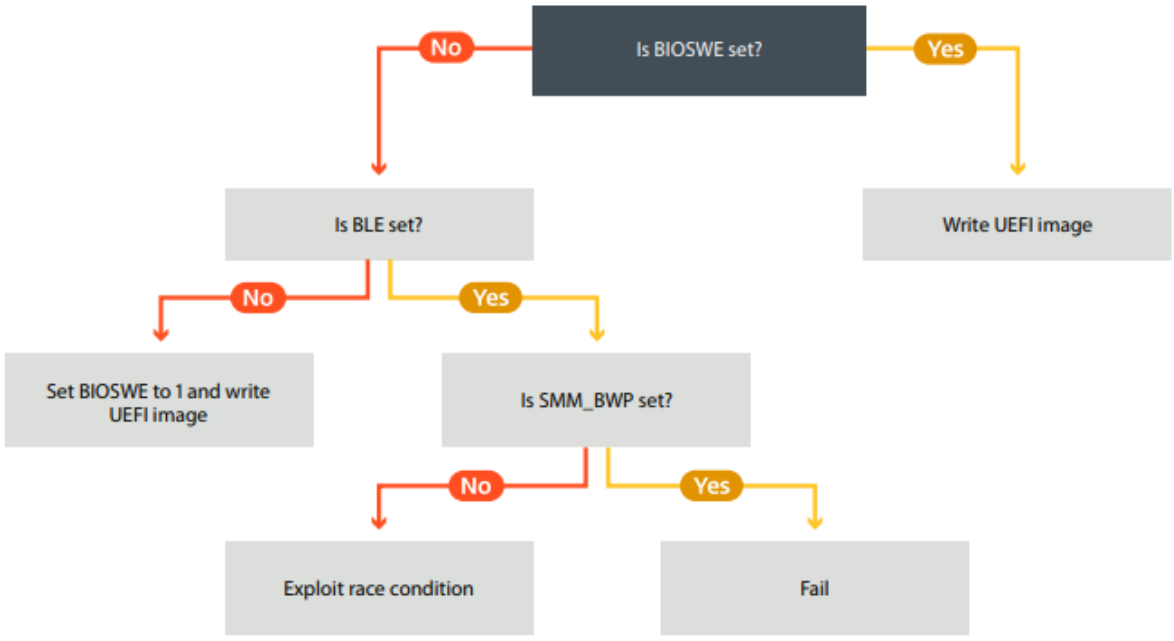

Рисунок 12. Древовидная схема принятия решения в процессе записи

Предполагая, что конкретный анализируемый файл `ReWriter_binary` использовался для развертывания UEFI руткита, можно прийти к заключению, что либо прошивка неверно сконфигурировала защиту от записи BIOS, либо чипсет машины жертвы старше, чем Platform Controller Hub.

`ReWriter_binary` не смог бы заменить прошивку UEFI на хорошо настроенной современной системе. Однако при поиске уязвимого образа SmiFlash UEFI парсинг томов прошивки UEFI предполагает, что злоумышленники могли работать и с более продвинутыми техниками обхода защиты записи BIOS.

Очень похожим на процедуру чтения образом происходит запись в SPI флеш-память:

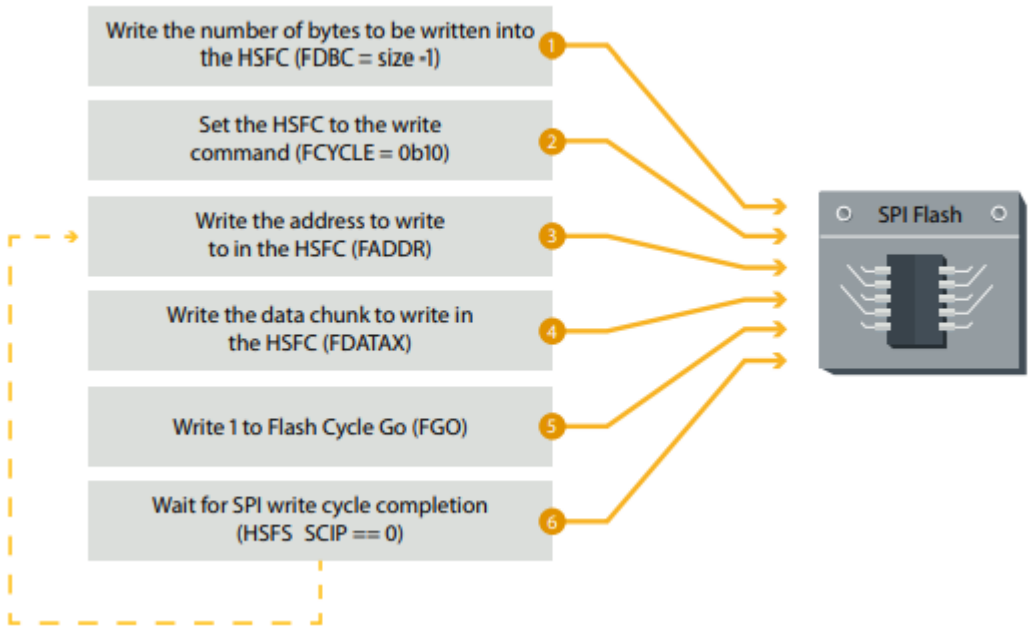

Рисунок 13. Последовательность записи в SPI флеш-память

Кроме первых двух шагов, выполняющихся лишь раз, эти операции повторяются в цикле, пока вся информация не будет записана в SPI флеш-память.

Когда процесс записи выполнен, содержимое SPI флеш-памяти выгружается еще раз в файл `image.bin`. Та же самая проверка целостности, которая проводилась `ReWriter_read`, выполняется на новом слитом образе. Затем образ, прочитанный из SPI флеш-памяти, сравнивается с пропатченным образом в памяти.

Если какие-либо байты отличаются, их адрес записывается в журнал. Наличие или отсутствие различий не влияет на ход работы вредоносной программы. Эта информация записывается только для того, чтобы операторы понимали, что происходит.

На финальном этапе ключ реестра устанавливается на значение:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session
Manager\BootExecute = “autocheck autochk *”
```

Затем служба RwDrv останавливается и удаляется. Важно, что ключу системного реестра Windows присваивается значение этой строки, так как руткит UEFI ищет именно эту строку для ее изменения и исполнения своего компонента в процессе запуска Windows. Мы более подробно расскажем об этом, когда будем описывать руткит UEFI и его компоненты.

Технический анализ LoJax

Инструмент для дампа, патчинга и записи в SPI флеш-память кастомизирован для конкретного образа прошивки и его нельзя использовать на любой системе. При этом полный модуль UEFI выделить можно. Первое, что мы сделали после получения этого модуля – изучили данные телеметрии, чтобы узнать, встречался ли он ранее. В этом нам пришлось полагаться на новый сканер UEFI, который может сканировать системную прошивку. Мы установили, что модуль UEFI группы Sednit был установлен в системе как минимум один раз, а это означает, что данный руткит действительно применяется in-the-wild.

Пока не установлено, как вредоносные инструменты были доставлены в скомпрометированные системы. Скорее всего, для этого использовались другие программы – например, XAgent. Инструменты для дампа и записи обнаружены в одной и той же системе, но в разное время – вероятно, операторы работали в несколько этапов. Сначала они выгрузили прошивку на целевой машине, убедились, что инструмент для внесения корректировок в программу работает без сбоев, а потом заново загрузили его и уже на самом деле пропатчили прошивку. Мы обнаружили только одну версию инструмента для дампа и записи, но есть возможность, что существуют иные версии для других прошивок с уязвимостями, которые они смогли найти.

На рисунке ниже представлен обзор работы UEFI руткита вплоть до загрузки ОС. Сначала драйвер SecDxe DXE загружается диспетчером DXE. Таким образом настраивается функция уведомления по группе событий `EFI_EVENT_GROUP_READY_TO_BOOT`. Когда прошивка готова выбрать устройство загрузки и запустить загрузчик ОС, вызывается функция уведомления. Она выполняет три действия:

- загружает встроенный драйвер NTFS DXE для предоставления доступа и возможности записи в разделы NTFS
- пишет два файла в раздел NTFS Windows: rpcnetp.exe и autoche.exe
- меняет ключ реестра ‘HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\ Session Manager\BootExecute’: до: ‘autocheck autochk *’; после: ‘autocheck autoche *’ .

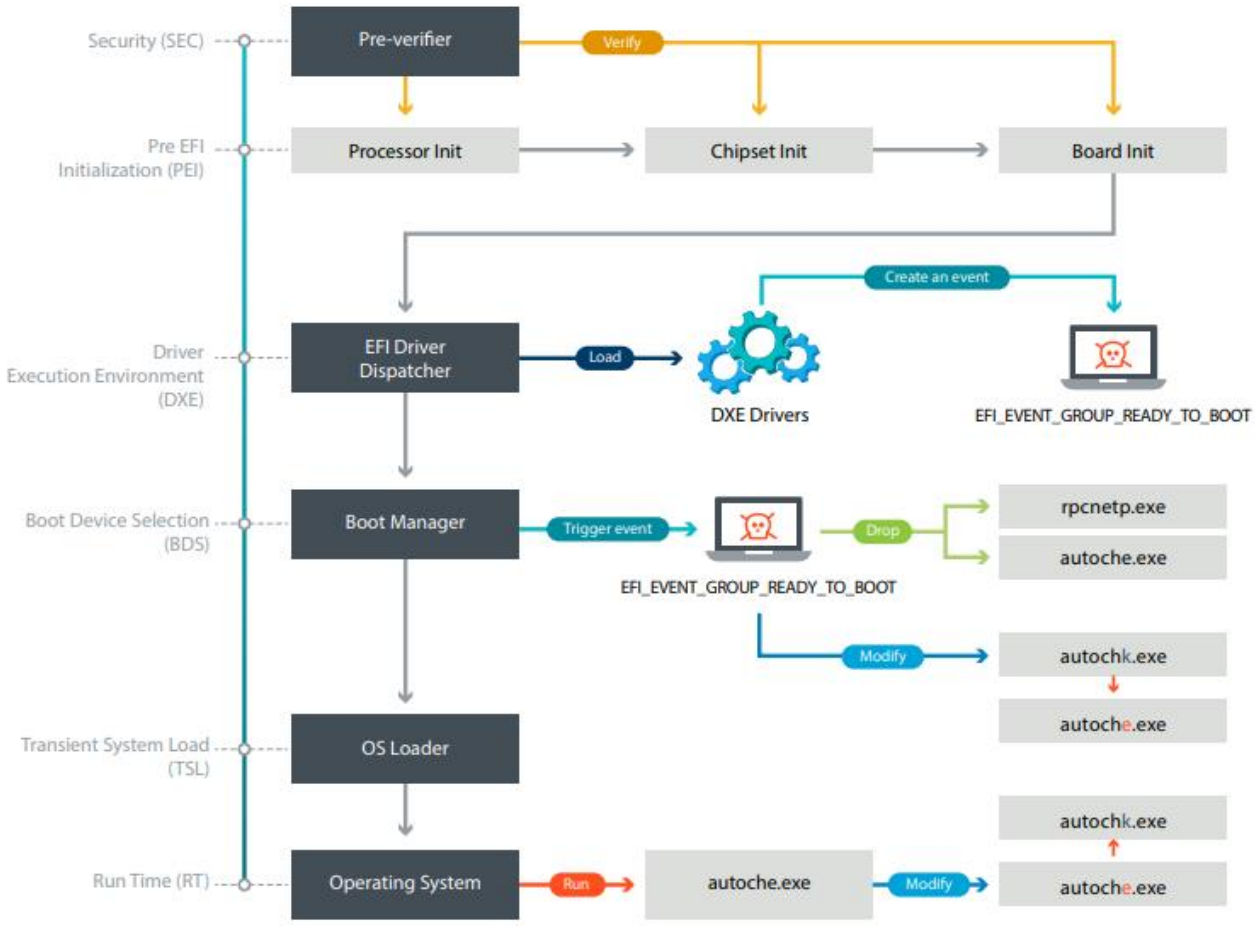

Рисунок 14. Процесс загрузки системы, зараженной руткитом UEFI

SecDxe: Вредоносный драйвер DXE

В данном разделе мы раскроем последовательность событий, происходящих в скомпрометированной системе. Начнем с описания руткита, а затем следуем по цепочке событий вплоть до финальных компонентов, разворачиваемых на уровне операционной системы.

UEFI руткит группы Sednit – это драйвер DXE с GUID идентификатором `682894B5-6B70-4EBA-9E90-A607E5676297`.

Он не подписан, поэтому не может быть запущен на системе с включенной защитой Secure Boot. После развертывания в одном из томов прошивки DXE Foundation загружает его при каждом старте системы.

SecDxe – небольшой драйвер DXE, который, в основном, делает две вещи. Он устанавливает протокол, определяемый по GUID `832d9b4d-d8d5-425f-bd52-5c5afb2c85dc`, который никогда не используется. Затем он создает связанное с функцией уведомления событие. Функция уведомления настраивается на вызов, по сигналу на группу `EFI_EVENT_GROUP_READY_TO_BOOT`. Сигнал на эту группу событий приходит от менеджера загрузки, когда он готов к выбору устройства для загрузки.

```
__int64 __fastcall fnCreateEventEx(EFI_TPL NotifyTpl_1, EFI_EVENT_NOTIFY NotifyFunction_, __int64 NotifyContext_, EFI_EVENT *Event_1)
{
  int *v4; // r8
  int *v5; // r9
  __int64 result; // rax
  EFI_TPL NotifyTpl; // [rsp+50h] [rbp+8h]
  EFI_EVENT_NOTIFY NotifyFunction; // [rsp+58h] [rbp+10h]
  __int64 NotifyContext; // [rsp+60h] [rbp+18h]
  EFI_EVENT *Event; // [rsp+68h] [rbp+20h]

  Event = Event_1;
  NotifyContext = NotifyContext_;
  NotifyFunction = NotifyFunction_;
  NotifyTpl = NotifyTpl_1;
  if ( fnRetZero() && !Event )
    sub_19CC("c:\\edk2\\MdePkg\\Library\\UefiLib\\UefiNotTiano.c", &byte_40[117], "ReadyToBootEvent != ((void *) 0)");
  if ( gEfiSystemTable->Hdr.Revision >= 0x20000 )
  {
    if ( NotifyFunction )
      result = (gEfiBootServices->CreateEventEx)(
                 EFI_EVENT_NOTIFY_SIGNAL,
                 NotifyTpl,
                 NotifyFunction,
                 NotifyContext,
                 &gEfiEventReadyToBootGuid,
                 Event);
    else
      result = (gEfiBootServices->CreateEventEx)(
                 EFI_EVENT_NOTIFY_SIGNAL,
                 NotifyTpl,
                 fnDefaultNotifyFunction,
                 NotifyContext,
                 &gEfiEventReadyToBootGuid,
                 Event);
  }
  else
  {
    if ( fnRetZero() && sub_19E0(0x80000000i64) )
      sub_19B4(0x80000000i64, "EFI1.1 can't support ReadyToBootEvent!", v4, v5);
    if ( fnRetZero() )
      sub_19CC("c:\\edk2\\MdePkg\\Library\\UefiLib\\UefiNotTiano.c", (&dword_B8 + 1), "((BOOLEAN)(0==1))");
    result = EFI_UNSUPPORTED;
  }
  return result;
}
```

Рисунок 15. Результат прохода декомпилятора Hex-Rays по процессу создания события

Функция уведомления использует вредоносное поведение UEFI руткита Sednit. Она пишет компоненты в файловую систему NTFS Windows. Как правило, прошивка UEFI единолично работает с разделом системы EFI, поэтому драйвер NTFS обычно не включен. Только файловые системы на FAT поддерживаются как разделы для загрузки. Поэтому UEFI прошивка не обязательно идет в комплекте с драйверами NTFS. По этой причине SecDxe имеет свой встроенный NTFS драйвер. Этот драйвер загружается первым и соединяется с дисковым устройством. То есть устанавливает `EFI_SIMPLE_FILE_SYSTEM_PROTOCOL` на дисковые устройства с NTFS разделами, таким образом включая к ним файловый доступ.

Теперь, когда все готово для записи файлов в разделы Windows, SecDxe сбрасывает `rpcnetp.exe` и `autoche.exe`. Затем `rpcnetp.exe` устанавливается в `%WINDIR%\SysWOW64` на 64-битных версиях Windows или `%WINDIR%\System32` на 32-битных версиях. А `autoche.exe` устанавливается в `%WINDIR%\SysWOW64`. На следующем рисунке показан процесс, отвечающий за запись этих файлов на диск.

```
EfiSimpleFileSystemProtocol->OpenVolume(EfiSimpleFileSystemProtocol, Root);
v2 = (*Root)->Open(*Root, WindowsDirHandle, WindowsDir, 1ui64, 0x10ui64);
if ( !v2 )
{
  if ( (*WindowsDirHandle)->Open(*WindowsDirHandle, SystemDirHandle, SysWOW64Dir, 1ui64, 0x10ui64) )
  {
    if ( !(*WindowsDirHandle)->Open(*WindowsDirHandle, SystemDirHandle, System32Dir, 1ui64, 0x10ui64) )
    {
      if ( (*SystemDirHandle)->Open(*SystemDirHandle, NewHandle, L"rpcnetp.exe", 1ui64, 0x20ui64) )
      {
        (*SystemDirHandle)->Open(*SystemDirHandle, NewHandle, L"rpcnetp.exe", 0x8000000000000003ui64, 0x20ui64);
        (*NewHandle)->Write(*NewHandle, &RpcnetpFileSize, &gRpcnetp_exe);
      }
      (*NewHandle)->Close(*NewHandle);
    }
  }
  else
  {
    if ( (*SystemDirHandle)->Open(*SystemDirHandle, NewHandle, L"rpcnetp.exe", 1ui64, 0x20ui64) )
    {
      (*SystemDirHandle)->Open(*SystemDirHandle, NewHandle, L"rpcnetp.exe", 0x8000000000000003ui64, 0x20ui64);
      (*NewHandle)->Write(*NewHandle, &RpcnetpFileSize, &gRpcnetp_exe);
    }
    (*NewHandle)->Close(*NewHandle);
  }
  v2 = (*WindowsDirHandle)->Open(*WindowsDirHandle, SystemDirHandle, System32Dir, 1ui64, 0x10ui64);
  if ( !v2 )
  {
    if ( (*SystemDirHandle)->Open(*SystemDirHandle, NewHandle, L"autoche.exe", 1ui64, 6ui64) )
    {
      (*SystemDirHandle)->Open(*SystemDirHandle, NewHandle, L"autoche.exe", 0x8000000000000003ui64, 6ui64);
      (*NewHandle)->Write(*NewHandle, &AutocheFileSize, &gAutoche_exe);
    }
    v2 = (*NewHandle)->Close(*NewHandle);
  }
}
```

Рисунок 16. Результат прохода декомпилятора Hex-Rays по процессу записи файлов на диск

Затем SecDxe открывает `%WINDIR%\System32\config\SYSTEM`, файл с бэкапом набора ключей реестра `HKLM\SYSTEM`. Он парсит файл, пока не находит `'autocheck autochk *'` и не заменяет `'k'` в `'autochk'` на `'e'`. В результате `'HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\BootExecute'` меняется на `'autocheck autoche *'`. При следующей загрузке Windows autoche.exe будет запущен вместо `autochk.exe`.

Драйвер NTFS от Hacking Team

Как уже раньше обсуждалось, в модуль SecDxe встроен драйвер NTFS. Есть серьезные доказательства того, что операторы Sednit не стали писать собственный драйвер, а скомпилировали копию обнародованного драйвера NTFS DXE от Hacking Team.

Их драйвер NTFS в качестве ядра использует опесорсный проект ntfs-3g. Это просто обертка, чтобы заставить его работать как драйвер UEFI DXE. Сам по себе файл INF с информацией о сборке драйвера Hacking Team перечисляет имена файлов проекта ntfs-3g. В строках кода SecDxe NTFS драйвера также перечисляются многие из этих имен файлов:

- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\inode.c`
- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\volume.c`
- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\bootsect.c`
- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\unistr.c`
- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\attrib.c`
- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\mft.c`
- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\index.c`
- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\cache.c`
- `c:\edk2\NtfsPkg\NtfsDxe\ntfs\misc.c`

- c:\edk2\NtfsPkg\NtfsDxe\ntfs\dir.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\runlist.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\logfile.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\uefi_io.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\ntfsinternal.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\mst.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\lcnalloc.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\compress.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\bitmap.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\collate.c
- c:\edk2\NtfsPkg\NtfsDxe\ntfs\security.c

Интересно отметить, что путь к проекту совпадает с тем, что можно обнаружить у vector-edk, проекта Hacking Team по разработке EFI. В vector-edk есть субпроект `NtfsPkg` с абсолютно идентичной схемой директорий. Файлы исходного кода проекта ntfs-3g находятся по тому же адресному пути. И хотя сами по себе пути ничем не примечательны, мы считаем, что это не простое совпадение.

Сравнивая утекший в сеть исходный код с тем, что мы получили на выходе декомпилятора Hex-Rays, становится очевидно, что это один и тот же проект. На рисунке ниже представлен пример сравнения функции `NtfsDriverBindingStart`, взятой из `vector-edk/NtfsPkg/NtfsDxe/Ntfs.c`. Комментарии из оригинального кода HT убраны для лучшего восприятия. Логика и последовательность вызовов функций одинаковы. Оба проекта даже используют одну переменную (LockedByMe) для сохранения заблокированного состояния.

```
ControllerHandle_1 = ControllerHandle;
EfiDriverBindingProtocol = This;
LockedByMe = 0;
if ( fnNtfsAcquireLockOrFail() >= 0 )
  LockedByMe = 1;
Status = fnInitlalizeUnicodeCollationSupport(EfiDriverBindingProtocol->DriverBindingHandle);
if ( Status >= 0 )
{
  v4 = EFI_OPEN_PROTOCOL_GET_PROTOCOL;
  Status = (gEfiBootServices->OpenProtocol)(
             ControllerHandle_1,
             &gEfiBlockIoProtocolGuid,
             &EfiBlockIoProtocol,
             EfiDriverBindingProtocol->DriverBindingHandle,
             ControllerHandle_1,
             v4);
  if ( Status >= 0 )
  {
    LODWORD(v6) = EFI_OPEN_PROTOCOL_BY_DRIVER;
    Status = (gEfiBootServices->OpenProtocol)(
               ControllerHandle_1,
               &gEfiDiskIoProtocolGuid,
               &EfiDiskIoProtocol,
               EfiDriverBindingProtocol->DriverBindingHandle,
               ControllerHandle_1,
               v6);
    if ( Status >= 0 )
    {
      Status = fnNtfsAllocateVolume(ControllerHandle_1,
          EfiDiskIoProtocol, EfiBlockIoProtocol);
      if ( Status < 0 )
      {
        LODWORD(v7) = 4;
        Status = (gEfiBootServices->OpenProtocol)(
                   ControllerHandle_1,
                   &gEfiSimpleFileSystemProtocolGuid,
                   0i64,
                   EfiDriverBindingProtocol->DriverBindingHandle,
                   ControllerHandle_1,
                   v7);
        if ( Status < 0 )
          (gEfiBootServices->CloseProtocol)(
            ControllerHandle_1,
            &gEfiDiskIoProtocolGuid,
            EfiDriverBindingProtocol->DriverBindingHandle,
            ControllerHandle_1);
      }
    }
  }
}
if ( LockedByMe )
  fnNtfsReleaseLock(v3);
return Status;
}
```

```
Status = NtfsAcquireLockOrFail ();
if (!EFI_ERROR (Status)) {
  LockedByMe = TRUE;
}

Status = InitlalizeUnicodeCollationSupport (This->DriverBindingHandle);
if (EFI_ERROR (Status)) {
  goto Exit;
}
Status = gBS->OpenProtocol (
               ControllerHandle,
               &gEfiBlockIoProtocolGuid,
               (VOID **) &BlockIo,
               This->DriverBindingHandle,
               ControllerHandle,
               EFI_OPEN_PROTOCOL_GET_PROTOCOL
               );
if (EFI_ERROR (Status)) {
  goto Exit;
}
Status = gBS->OpenProtocol (
               ControllerHandle,
               &gEfiDiskIoProtocolGuid,
               (VOID **) &DiskIo,
               This->DriverBindingHandle,
               ControllerHandle,
               EFI_OPEN_PROTOCOL_BY_DRIVER
               );
if (EFI_ERROR (Status)) {
  goto Exit;
}
Status = NtfsAllocateVolume (ControllerHandle, DiskIo, BlockIo);
if (EFI_ERROR (Status)) {
  Status = gBS->OpenProtocol (
               ControllerHandle,
               &gEfiSimpleFileSystemProtocolGuid,
               NULL,
               This->DriverBindingHandle,
               ControllerHandle,
               EFI_OPEN_PROTOCOL_TEST_PROTOCOL
               );
  if (EFI_ERROR (Status)) {
    gBS->CloseProtocol (
           ControllerHandle,
           &gEfiDiskIoProtocolGuid,
           This->DriverBindingHandle,
           ControllerHandle
           );
  }
}

Exit:
  if (LockedByMe) {
    NtfsReleaseLock ();
  }
```

Рисунок 17. Сравнение результатов на выходе декомпилятора Hex-Rays NTFS драйвера Sednit (слева) и NTFS драйвера HT (справа)

Выше показан код разработчиков из Hacking Team, отсутствующий в открытых исходниках ntfs-3g. Как упоминалось в разделе `ReWriter_binary`, в процессе парсинга по файловой системе прошивки исполняемый файл пытается удалить драйвер AMI NTFS. Мы хотели разобраться, почему его удаляют вместо того, чтобы использовать.

Мы проанализировали драйвер и обнаружили, что он может выполнять только операции чтения. Так как запись в файловой системе недоступна, разработчики не могли его использовать в своих целях. Также вероятно, что операторы Sednit столкнулись с трудностями, связанными с тем, что в прошивке уже есть другой драйвер NTFS, поэтому они решили просто удалить его. Вдобавок к реализации возможности выполнять чтение и запись, драйвер Hacking Team не соблюдает права доступа к файлам. К примеру, он может переписать файл только для чтения, не вызывая какую-либо ошибку.

К этому моменту мы уже описали различные операции по компрометации системы, выполняемые руткитом UEFI. Мы также обсудили причины, по которым считаем, что операторы Sednit использовали исходный код vector-edk от Hacking Team для разработки своего драйвера NTFS для записи файлов в NTFS разделы Windows. Далее в работе мы представим анализ компонентов, доставляемых SecDxe.

autoche.exe vs. autochk.exe

Вредоносный `autoche.exe` используется для обеспечения персистентности мини-агента `rpcnetp.exe`. Как видно из следующего рисунка, он использует нативные обращения к Windows API для создания службы.

```
if ( NtOpenKey(&KeyHandle, 0xF003Fu, &ObjectAttributes) < 0 )
{
  NtCreateKey(&KeyHandle, KEY_ALL_ACCESS, &ObjectAttributes, 0u, 0u, 0u, 0u);
  RtlInitUnicodeString(&ValueName, L"DisplayName");
  RtlInitUnicodeString(&v5, L"Remote Procedure Call (RPC) Net")
  if ( NtSetValueKey(KeyHandle, &ValueName, 0u, 1u, v5.Buffer, v5.MaximumLength) >= 0 )
  {
    RtlInitUnicodeString(&ValueName, L"ObjectName");
    RtlInitUnicodeString(&v5, L"LocalSystem");
    if ( NtSetValueKey(KeyHandle, &ValueName, 0u, 1u, v5.Buffer, v5.MaximumLength) >= 0 )
    {
      RtlInitUnicodeString(&ValueName, L"ErrorControl");
      Data = 1;
      if ( NtSetValueKey(KeyHandle, &ValueName, 0u, 4u, &Data, 4u) >= 0 )
      {
        RtlInitUnicodeString(&ValueName, L"ImagePath");
        v19 = NtCreateFile(&FileHandle, 1u, &v24, &IoStatusBlock, 0u, 128u, 1u, 1u, 1u, 0u, 0u);
        RtlInitUnicodeString(&v5, L"C:\\Windows\\SysWOW64\\rpcnetp.exe");
        if ( v19 < 0 )
          RtlInitUnicodeString(&v5, L"C:\\Windows\\System32\\rpcnetp.exe");
        if ( NtSetValueKey(KeyHandle, &ValueName, 0u, 2u, v5.Buffer, v5.MaximumLength) >= 0 )
        {
          RtlInitUnicodeString(&ValueName, L"Start");
          v20 = 2;
          if ( NtSetValueKey(KeyHandle, &ValueName, 0u, 4u, &v20, 4u) >= 0 )
          {
            RtlInitUnicodeString(&ValueName, L"Type");
            v21 = 16;
            NtSetValueKey(KeyHandle, &ValueName, 0u, 4u, &v21, 4u);
```

Рисунок 18. Вредоносный autoche .exe настраивает персистентность rpcnetp.exe

Необходимо отметить, что имя службы совпадает с тем, которое использует легитимный агент Computrace. После создания службы он восстанавливает предыдущее значение ключа реестра `BootExecute`.

```
NtClose(FileHandle);
RtlInitUnicodeString(&v28, L"\\REGISTRY\\MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Session Manager");
ObjectAttributes.Length = 24;
ObjectAttributes.RootDirectory = 0;
ObjectAttributes.Attributes = 512;
ObjectAttributes.ObjectName = &v28;
ObjectAttributes.SecurityDescriptor = 0;
ObjectAttributes.SecurityQualityOfService = 0;
NtOpenKey(&v23, 0xF003Fu, &ObjectAttributes);
*SourceString = 'u\0a';
v8 = 'o\0t';
v9 = 'h\0c';
v10 = 'c\0e';
v11 = ' \0k';
v12 = 'u\0a';
v13 = 'o\0t';
v14 = 'h\0c';
v15 = ' \0k';
v16 = '*';
v17 = 0;
RtlInitUnicodeString(&ValueName, L"BootExecute");
RtlInitUnicodeString(&v5, SourceString);
NtSetValueKey(v23, &ValueName, 0u, 7u, SourceString, 0x2Au);
return NtTerminateProcess(0xFFFFFFFF, 0);
```

Рисунок 19. Вредоносный autoche.exe восстанавливает оригинальное значение ключа реестра BootExecute

Так как процесс происходит во время загрузки Windows, пользователь вряд ли заметит изменение значения ключа `BootExecute`. Нужно отметить, что в autoche .exe заметны некоторые сходства с модулем autochk.exe в Computrace, например, применяемые вызовы API и регистрация служб, но остальное довольно сильно отличается. Модуль Computrace больше, и он восстанавливает оригинальный исполняемый файл `autochk.exe` вместо изменения ключа реестра. Он также отвечает за внедрение мини-агента на диск, в то время как у LoJax это делает UEFI руткит.

rpcnetp.exe

В то время как мини-агент `rpcnetp.exe` может быть внедрен UEFI руткитом, есть вероятность, что в большинстве случаев, когда мы находили троянизированную версию LoJack, мини-агент не использовал этот компонент. Вероятно, что они исходили из оппортунистических соображений и устанавливали UEFI руткит, только когда имели такую возможность, и в наиболее интересных для них организациях.

В ходе расследования мы обнаружили различные версии мини-агента LoJax. В списке индикаторов компрометации приведены их хеши и соответствующие домены/IP-адреса. Как мы уже говорили, все найденные образцы были троянизированной версией одного и того же старого агента Computrace, скомпилированного в 2008 году.

Мы ни разу не видели, чтобы агент LoJax загружал и устанавливал дополнительные модули, но знаем, что такой функционал существует. Так как лучшие качества LoJax – скрытность и персистентность, его точно можно использовать для обеспечения доступа к ключевым ресурсам.

Предотвращение атаки и ликвидация последствий

Для предотвращения атаки необходима комплексная экосистема, состоящая из множества действующих компонентов.

Рекомендуем включить функцию Secure Boot, это базовая защита от атак на прошивку UEFI.

Как и в случае с программным обеспечением, прошивка UEFI должна всегда своевременно обновляться. Посетите сайт производителя материнской платы, чтобы убедиться, что у вас последняя доступная версия.

Вам также стоит убедиться, что на всех ваших системах установлены современные чипсеты с Platform Controller Hub (начиная с чипсетов Intel Series 5 и далее). Это обеспечит работу механизмов безопасности против «уязвимости состояния гонки», которая, как мы указали, присутствует в платформе.

Другая часть безопасности прошивки в руках вендоров UEFI/BIOS. Механизмы безопасности, предоставляемые платформой, должны быть правильно сконфигурированы системной прошивкой, чтобы действительно обеспечивать ее защиту. Поэтому прошивка должна быть изначально построена с пониманием мер безопасности. К счастью, все больше исследователей безопасности обращает внимание на безопасность прошивки, привлекая внимание вендоров. Также заслуживает упоминания CHIPSEC, фреймворк с открытым исходным кодом, выполняющим оценку низкоуровневой безопасности, что позволяет определить, правильно ли сконфигурирована платформа.

Ликвидация последствий компрометации через UEFI прошивку является трудной задачей. Нет простых способов провести очистку системы от таких угроз, а также нет специальных продуктов по безопасности, которые могли бы все исправить. В описываемом здесь случае для удаления руткита необходимо перепрошить SPI флеш-память. Это нетривиальная задача, для среднестатистического пользователя она не подходит. Обновление прошивки UEFI может убрать руткит, если будет переписана вся область SPI флеш-памяти. Если перепрошивка UEFI невозможна, единственное решение – замена материнской платы зараженной системы.

Выводы

UEFI руткит – один из наиболее опасных и мощных инструментов злоумышленников по причине высокой персистентности и невосприимчивости к переустановке ОС и замене жесткого диска, а также исключительной трудности обнаружения и удаления. Хотя UEFI образ системы трудно изменить, мало какие решения позволяют просканировать модули UEFI и определить среди них вредоносные. Более того, очистка UEFI прошивки означает ее перепрошивку, не совсем обычную операцию, которую не сможет выполнить обычный пользователь. Эти преимущества объясняют, почему кибергруппы, не ограниченные в ресурсах, продолжат атаки на UEFI системы.

По любым вопросам о данной работе обращайтесь по адресу threatintel@eset.com.

Хотим выразить признательность авторам проекта opensecuritytraining.info. Курс Introduction to BIOS & SMM очень помог нам, когда мы анализировали взаимодействия с флеш-чипом SPI.

Глоссарий

Смотрите спецификации Intel для более детального описания сокращений.
— BIOS_CNTL: BIOS Control Register
— BIOSWE: BIOS Write Enabled
— BLE: BIOS Lock Enabled
— FADDR: Flash Address
— FDATAX: Flash Data from FDATA0 to FDATAN
— FDBC: Flash Data Byte Count
— FGO: Flash Cycle Go
— HSFC: Hardware Sequencing Flash Control
— HSFS: Hardware Sequencing Flash Status
— IOCTL: Input/Output Control
— PCH: Platform Controller Hub
— RCBA: Root Complex Base Address Register

— RCRB: Root Complex Register Block
— SCIP: SPI Cycle in Progress
— SMI: System Management Interrupt
— SMM: System Management Mode
— SMM_BWP: SMM BIOS Write Protect Disable
— SPI: Serial Peripheral Interface

Индикаторы компрометации

ReWriter_read.exe

Детектирование ESET
`Win32/SPIFlash.A`
SHA-1
`ea728abe26bac161e110970051e1561fd51db93b`

ReWriter_binary.exe

Детектирование ESET
`Win32/SPIFlash.A`
SHA-1
`cc217342373967d1916cb20eca5ccb29caaf7c1b`

SecDxe

Детектирование ESET
`EFI/LoJax.A`
SHA-1
`f2be778971ad9df2082a266bd04ab657bd287413`

info_efi.exe

Детектирование ESET
`Win32/Agent.ZXZ`
SHA-1
`4b9e71615b37aea1eaeb5b1cfa0eee048118ff72`

autoche.exe

Детектирование ESET
`Win32/LoJax.A`
SHA-1
`700d7e763f59e706b4f05c69911319690f85432e`

Мини-агент EXE

Детектирование ESET
`Win32/Agent.ZQE`
`Win32/Agent.ZTU`
SHA-1
`1771e435ba25f9cdfa77168899490d87681f2029`
`ddaa06a4021baf980a08caea899f2904609410b9`
`10d571d66d3ab7b9ddf6a850cb9b8e38b07623c0`
`2529f6eda28d54490119d2123d22da56783c704f`
`e923ac79046ffa06f67d3f4c567e84a82dd7ff1b`
`8e138eecea8e9937a83bffe100d842d6381b6bb1`
`ef860dca7d7c928b68c4218007fb9069c6e654e9`
`e8f07caafb23eff83020406c21645d8ed0005ca6`
`09d2e2c26247a4a908952fee36b56b360561984f`
`f90ccf57e75923812c2c1da9f56166b36d1482be`

АНТИВИРУСНАЯ ЗАЩИТА БИЗНЕС-КЛАССА

Имена доменов командных серверов

secao[.]org
ikmtrust[.]com
sysanalyticweb[.]com
lxwo[.]org
jflynci[.]com
remotepx[.]net
rdsnets[.]com
rpcnetconnect[.]com
webstp[.]com
elaxo[.]org

IP-адреса командных серверов

185.77.129[.]106
185.144.82[.]239
93.113.131[.]103
185.86.149[.]54
185.86.151[.]104
103.41.177[.]43
185.86.148[.]184
185.94.191[.]65
86.106.131[.]54

Мини-агент DLL

Детектирование ESET
Win32/Agent.ZQE
SHA-1
397d97e278110a48bd2cb11bb5632b99a9100dbd
Имена доменов командных серверов
elaxo.org
IP-адреса командных серверов
86.106.131[.]54