ESET Whitepaper // September 2018

LOJAX

Erstes UEFI-Rootkit in freier Wildbahn gefunden – Sednit-Gruppe erweitert Arsenal

ENJOY SAFER TECHNOLOGY™

INHALT

1. ZUSAMMENFASSUNG

Sednit, auch bekannt als APT28, Sofacy, Strontium und Fancy Bear, ist seit mindestens 2004 aktiv und war seitdem bereits mehrfach in die Schlagzeilen. Man nimmt an, dass einige hochkarätige Angriffe auf das Konto der Gruppe gehen. So vermuten mehrere Sicherheitsunternehmen [1] sowie das amerikanische Sicherheitsministerium [2], dass der Angriff auf das Democratic National Committee (DNC) kurz vor den Präsidentschaftswahlen in den USA im Jahr 2016 Sednit zuzuschreiben ist. Zudem soll die Gruppe hinter den Angriffen auf den französischen TV-Sender TV5Monde [3], die Anti-Doping-Agentur WADA [4], die deutsche Regierung 2016 und 2017 [22] sowie viele andere mehr stecken. Im Verlauf erweiterte Sednit sein Arsenal immer mehr, wobei wir einige der eingesetzten Werkzeuge bereits ausführlich dokumentiert haben. Diese Veröffentlichung soll jedoch vor allem die erste Verwendung eines UEFI-Rootkits in freier Wildbahn durch Sednit beleuchten.

Zentrale Punkte dieses Whitepapers:

- Schon zu Beginn des Jahres 2017 wurden trojanisierte Versionen eines früheren Userland-Agenten der Anti-Diebstahl-Software LoJack von Absolute Software außerhalb von Forschungslaboren identifiziert. Wir gaben dieser Modifikation den Namen LoJax. LoJack hat in den letzten Jahren viel Aufmerksamkeit auf sich gezogen – verwendet es doch ein UEFI/BIOS-Modul um dauerhauft auf Geräten bleiben zu können.
- Aufgrund der Tatsache, dass Sednit-Rootkits zusammen mit LoJax-Samples auftauchen und weil einige C&C-Server, die diese Trojaner verwenden, Teil einer bereits bekannten Sednit-Netzwerk-Infrastruktur sind, gehen wir mit hoher Sicherheit davon aus, dass das Rootkit von Sednit entwickelt wurde.
- Neben dem LoJax-Agenten konnten wir weitere Tools identifizieren, die die UEFI-Firmware eines Systems lesen können. In einem Fall war ein Tool sogar in der Lage, den SPI-Flash-Speicher eines Systems auszulesen, zu patchen und zu überschreiben. Ziel ist es dabei, ein schädliches UEFI-Modul auf Systemen mit unzureichendem oder falsch konfiguriertem Schutz des SPI-Flash-Speichers zu installieren.
- Dieses UEFI-Modul wiederum platziert den LoJax-Agenten im System und kann so als das erste echte Sednit UEFI-Rootkit angesehen werden. Es befindet sich in der Firmware des Systems und kann so eine Neuinstallation des Betriebssystems sowie den Austausch der gesamten Festplatte überstehen.
- Unseren Erkenntnissen zufolge wurde es erfolgreich in mindestens einem SPI-Flash-Speicher installiert und ist so das erste, in freier Wildbahn identifizierte UEFI-Rootkit.

Sollten Sie Fragen zur vorliegenden Veröffentlichung haben, kontaktieren Sie uns unter *threatintel@eset.com*

2. EINLEITUNG

Bei Sednit handelt es sich um eine ressourcenstarke APT-Gruppe, die seit mindestens 2004 sowohl Einzelpersonen als auch Organisationen auf der ganzen Welt ins Visier nimmt und für ihre Angriffe eine Vielzahl an Malware-Familien zum Einsatz bringt. Eine Übersicht über die am häufigsten genutzten Tools finden sich in unserem Whitepaper zu Sednit [5]. Wir beobachten die Gruppe seit Jahren und veröffentlichten bereits mehrfach Berichte zu ihren Aktivitäten, darunter Zero-Day-Angriffe [6] und komplett neu entwickelte Malware wie Zebrocy [7]. Die hier beschriebene Komponente spielt jedoch in einer komplett anderen Liga.

Berichte über UEFI-Rootkits gab es schon häufiger, darunter „rkloader", das durch den Datenleak beim Hacking Team bekannt wurde [8] oder „DerStarke", ein macOS EFI/UEFI Boot Implantat, das in den Vault7 Leaks beschrieben wird [9]. Die Existenz solcher Tools war durchaus seit längerem bekannt – bis heute gab es aber keinen Bericht über die Entdeckung einer solchen Attacke außerhalb von Forschungslaboren.

Im Verlauf unserer Forschung stießen wir nun jedoch auf Firmware, auf der das LoJax UEFI-Modul platziert worden war und konnten zudem noch die komplette Toolchain nachvollziehen, mit der dies durchgeführt wurde. Tatsächlich verwendete Sednit schon 2013 und 2014 das DownDelph-Bootkit, um eine ihrer ersten Backdoors, DownDelph, dauerhaft in den Opfersystemen zu verankern. Obwohl sie grundsätzlich dem aktuellen Rootkit nicht unähnlich sind, sind solche Bootkits mit der neuen UEFI-Implementierung komplett wirkungslos geworden. Die hier beschriebene Komponente hebt sich daher auch in zentralen Punkten von ihnen ab, die im folgenden näher erläutert werden sollen.

Vorliegende Veröffentlichung unterteilt sich in drei Kapitel. Das erste beschäftigt sich mit bisherigen Forschungsergebnissen zu LoJack/Computrace und deren böswilliger Nutzung. Das zweite Kapitel legt dar, wie wir Schritt für Schritt das UEFI-Rootkit ausfindig machen konnten. Das dritte Kapitel gibt schließlich Einblick in die verschiedenen LoJax-Komponenten und wie sie sich so fest im System einnisten, dass sie selbst eine Neuinstallation von Windows oder den Austausch der Festplatte überstehen.

Attribuierung

In der Vergangenheit wurden mehrfach Vermutungen angestellt, wo die Sednit-Gruppe geopolitisch verortet werden könnte. Wie wir jedoch bereits im Whitepaper zu Sednit von 2016 [5] verdeutlicht haben, werden wir dies nicht tun. Wie damals erläutert, ist eine wissenschaftlich fundierte Attribuierung im Rahmen unserer Möglichkeiten nicht durchführbar. Was wir als „Sednit-Gruppe" bezeichnen, ist lediglich eine Ansammlung von Software und damit verbundener Netzwerk-Infrastruktur, die sich nicht mit absoluter Sicherheit einer einzigen Organisation zuschreiben lassen.

Typische Ziele

Im Rahmen unserer Forschungen was es uns möglich, verschiedene LoJax-Samples zu analysieren. Auf Basis von Telemetrie-Daten und weiteren, bereits identifizierten Sednit-Tools können wir davon ausgehen, dass LoJax im Vergleich zu anderer Malware der Gruppe eher selten verwendet wird. Ziele waren zumeist Regierungsorganisationen auf dem Balkan sowie in Mittel- und Osteuropa.

3. BISHERIGE FORSCHUNGSERGEBNISSE ZU COMPUTRACE/ LOJACK

Bei LoJack handelt es sich um eine Anti-Diebstahl-Software der Absolute Software Cooperation. Frühere Versionen waren unter dem Namen Computrace bekannt. Wie dieser Name schon richtig andeutet, ermöglicht dieser Agent dem Rechner, mit einem Command & Control (C&C) -Server zu kommunizieren. Der Nutzer kann so bei Verlust oder Diebstahl den Standort des Gerätes feststellen.

Dieser Abschnitt soll die Architektur früherer LoJack-Versionen beschreiben. Das macht dahingehend Sinn, als dass nur ältere Versionen der Software trojanisiert worden sind. Absolute Software berichtete im Mai 2018 [10], dass die unten erläuterten Schwachstellen in aktuellen Versionen ihrer Agents geschlossen worden seien.

Computrace erregte vor allem wegen seines ungewöhnlichen Persistenzmechanismus die Aufmerksamkeit von Sicherheitsexperten. Aufgabe der Software ist es, Hardware vor Diebstahl zu schützen – es muss also Neuinstallationen des Betriebssystems und sogar den Tausch der Festplatte überstehen. Hierfür nutzt es ein UEFI/BIOS-Modul, welches bereits Teil der Firmware vieler Laptops verschiedener Hersteller ist und lediglich durch den Nutzer aktiviert werden muss. Dies wird mithilfe der in Abbildung 1 dargestellten BIOS-Option erledigt.

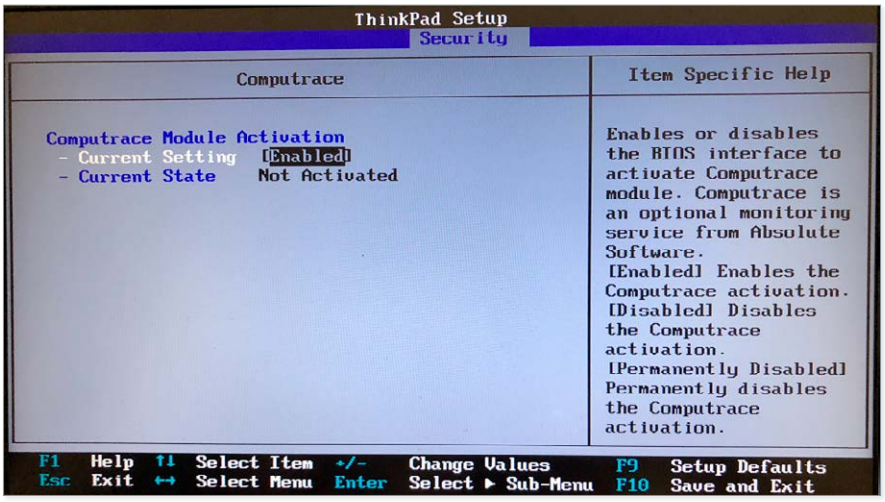

Abbildung 1 // Aktivierung von Computrace im BIOS

Einer der ersten Forschungsberichte über die Implementierung von LoJax wurde 2009 veröffentlicht [11]. Er beschreibt detailliert die Architektur des Produktes und wie das Modul in der Lage ist, den Userland-Agenten im System zu platzieren, um von dort aus den Webserver von Absolute Software zu kontaktieren. Eine Übersicht über den Ablauf findet sich in Abbildung 2.

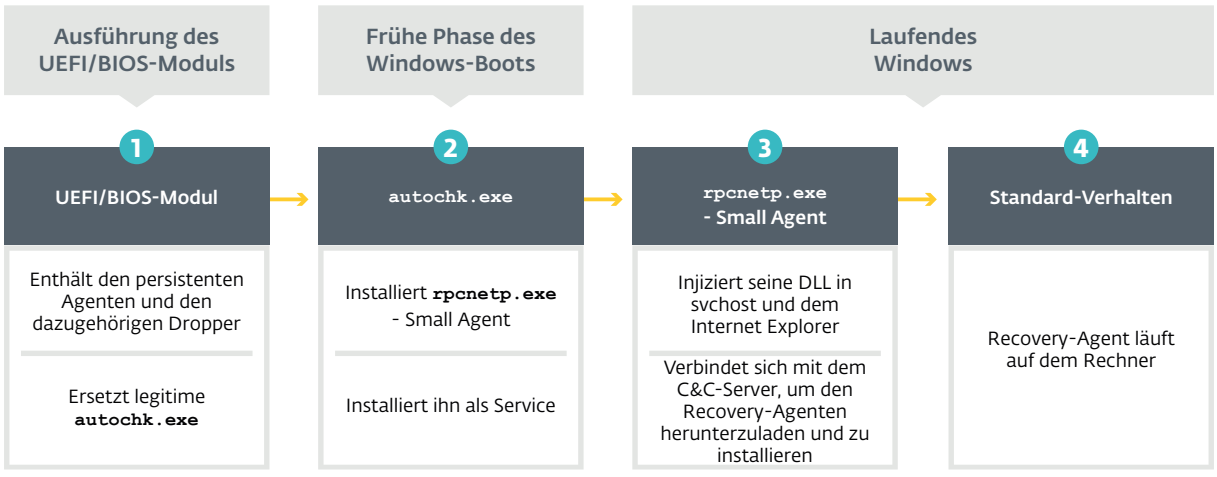

Abbildung 2 // LoJacks Persistenzmechanismus (ca. 2008)

Die verschiedenen Schritte sehen dabei folgendermaßen aus:

1. Während der Rechner hochfährt, wird das UEFI/BIOS-Modul ausgeführt (sofern es aktiviert ist) und versucht, eine FAT/FAT32/NTFS-Partition zu finden. Mithilfe eines NTFS-Treibers erstellt es dann eine Kopie von `autochk.exe` und überschreibt die Originaldatei mithilfe eines Droppers, der für die Installation der Userland-Agent-Komponente zuständig ist. `Autochk.exe` wird zu Beginn der Windows-Initialisierung ausgeführt und soll Fehlfunktionen der Festplatte identifizieren.
2. Die Ausführung der modifizierten `autochk.exe` dient jedoch dazu, den Small Agent `rpcnetp.exe` zu installieren und als Service hinzuzufügen, sodass er bei jedem Hochfahren ausgeführt wird. Im Anschluss wird die ursprüngliche Version von `autochk.exe` wiederhergestellt.
3. Beim Small Agent `rpcnetp.exe`, handelt es sich um eine kleine ausführbare Datei, die vor allem sicherstellen soll, dass der Main Agent läuft. Wenn nicht, versucht sie sich mit dem C&C-Server von Absolute Software zu verbinden, um ihn erneut herunterzuladen und auszuführen. Der Small Agent erstellt hierfür zunächst eine Kopie von sich selbst und verändert seinen PE-Header so, dass er zu einer Dynamic Link Library (DLL) wird. Diese DLL wird dann in den Speicher geladen, startet dort einen `svchost.exe`-Prozess und fügt die DLL ein. Dadurch wird ein `iexplore.exe`-Prozess des Internet Explorers gestartet und die DLL auch hier eingefügt. Letztgenannter Prozess wird genutzt, um über das Internet zu kommunizieren. Das Einbringen von Code in fremde Prozesse, wie es der Computrace Small Agent hier durchführt, ist eigentlich typisch für Malware und für legitime Software eher ungewöhnlich.
4. Der nun komplett ausgestattete Agent läuft jetzt auf dem System und implementiert die verschiedenen Tracking- und Recovery-Funktionen von Computrace.

Diese Prozessbeschreibung wurde 2014 zusammen mit dem Netzwerkprotokoll der Kommunikation zwischen Small Agent und C&C-Server veröffentlicht [12]. Besonders bedenklich ist, dass jeder Angreifer mit Zugriff auf den Kontrollserver den Small Agent dazu bringen kann, jede Art von Code herunterzuladen und auszuführen. Angreifer haben jedoch noch weitere Möglichkeiten, mit dem Small Agent direkt zu kommunizieren – insbesondere über den Mechanismus, mit dem der Small Agent die Adresse des C&C-Servers in Erfahrung bringt. Diese Information wird in einem Konfigurationsfile, welches fest in die .exe einprogrammiert ist, gespeichert.

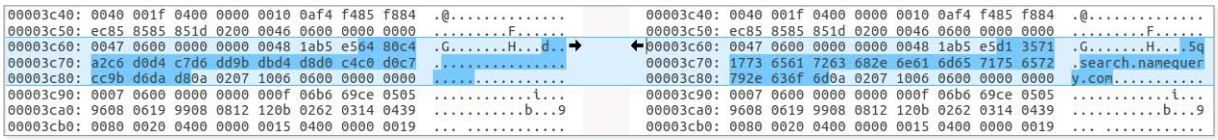

Abbildung 3 // Verschlüsseltes LoJack-Konfigurationsfile links, teilweise entschlüsselt rechts

Abbildung 3 zeigt das verschlüsselte und das teilweise entschlüsselte Konfigurationsfile des LoJack Small Agent. Die „Verschlüsselung" ist eine einfache XOR-Operation mit einem One-Byte-Schlüssel. Dieser Schlüssel 0xB5 ist für alle untersuchten Small Agents gleich. Wie auch in Abbildung 3 ersichtlich, ist der C&C-Domänenname klar lesbar. Die vier Bytes davor geben die IP-Adresse des C&C-Servers an. Da keine Validierung des Inhalts des Konfigurationsfiles durchgeführt wird, können Angreifer mit Schreibrechten auf `%WINDIR%` ihren Inhalt verändern, z.B. um den Small Agent dazu zu bringen, sich anstelle des legitimen mit einem gekaperten C&C-Server zu verbinden. Kennt ein Angreifer das Netzwerkprotokoll, kann er so einen Small Agent dazu bringen, jede Art von Code herunterzuladen und auszuführen.

Auch wenn diese Risiken bereits vor langer Zeit identifiziert worden sind, konnte bis zu unserer aktuellen Entdeckung keine Nutzung der Sicherheitslücke festgestellt werden.

Aus LoJack wird LoJax

In einem Blogpost beschrieben Arbor Networks im Mai 2018 einige trojanisierte Versionen des LoJack Small Agent `rpcnetp.exe`. Die Konfigurationseinstellungen dieser Samples waren so manipuliert worden, dass sie mit einem gefälschten C&C-Server kommunizieren anstelle des legitimen Kontrollservers von Absolute Software. Einige Domains, die in den LoJax Samples identifiziert werden konnten, sind eingehend bekannt: Sie wurden bereits Ende 2017 als C&C-Domains der berüchtigten Sednit First-Stage-Backdoor SedUploader verwendet. Abbildung 4 zeigt die modifizierten Konfigurationen eines solchen LoJax Small Agent.

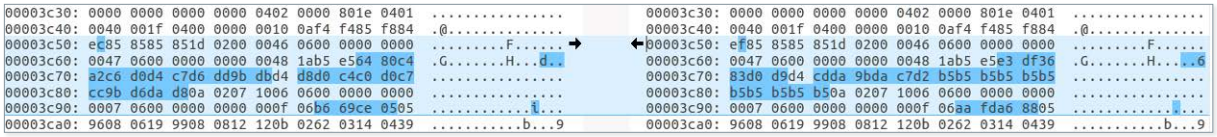

Abbildung 4 // Legitimes Konfigurationsfile links, modifiziertes rechts

Die Unterschiede zwischen legitimem und trojanisiertem Agent sind so klein, dass Abbildung 4 schon fast alle enthält. Alle von uns identifizierten Samples trojanisieren das exakt gleiche Sample des Computrace Small Agent `rpcnetp.exe`. Alle besitzen den gleichen Kompilierungszeitstempel und weichen nur um einige Dutzend Bytes vom Original ab. Neben den Modifikationen an der Konfigurationsdatei wurden u.a. die zeitlichen Abstände zwischen den Verbindungen zum C&C-Server verändert.

Zum Zeitpunkt der Veröffentlichung des Blogeintrags hatten wir bereits verschiedene LoJax-Agenten mit unterschiedlichen Zielen auf dem Balkan und in Mittel- und Westeuropa identifiziert, wussten aber nicht, wie diese in die betroffenen Systeme gelangt waren. Die einfachste Antwort auf diese Frage wäre natürlich gewesen, dass eine der bekannten Sednit-Backdoors sie installiert hätte. Schließlich handelte es sich bei LoJack um ein bekanntes und bei vielen Security-Anbietern als ungefährlich eingestuftes Tool. Das heißt, dass ein Malwarescanner den Small Agent vermutlich nicht als bösartig einstufen würde. Was aber, wenn die Gefahr noch größer ist und die Malware versucht, LoJack nachzuahmen, um so bis zur Firmware des Systems vorzudringen?

4. AUF DER JAGD NACH EINER LOW-LEVEL-KOMPONENTE

Im Verlauf unserer Nachforschungen waren wir in der Lage, LoJax-Kampagnen mit Zielen auf dem Balkan und Mittel- bzw. Osteuropa aufzudecken. In allen fanden sich Elemente anderer Sednit-Malware, darunter:

- SedUploader, eine First-Stage-Backdoor,
- XAgent, Sednits „Flaggschiff"-Backdoor,
- sowie Xtunnel, ein Netzwerk-Proxy-Tool, welches jede Art von Traffic zwischen C&C-Server und Endpoint umleiten kann.

Obwohl wir in den meisten Systemen, die durch LoJax kompromittiert worden waren, Spuren von Sednit-Tools fanden, gab es auch einige, auf denen nur LoJax selbst zu finden war. Wir können daraus schließen, dass LoJax in manchen Fällen als Stand-alone-Tool verwendet wird – vermutlich als zusätzliche Backdoor, um den Zugang zum Netzwerk für die Angreifer wiederherzustellen, sollte er einmal gekappt werden.

Da XAgent standardmäßig verwendet wird, um zusätzliche Module auf ein bereits infiziertes System aufzuspielen, ist man geneigt anzunehmen, dass LoJax-Samples auf dieselbe Art und Weise eingespielt werden und dass kein weiterer Mechanismus im Spiel ist. Das würde bedeuten, dass LoJack nur vom Small Agent inspiriert worden wäre. Wir merkten jedoch schnell, dass der Einfluss ein ganzes Stück weiter ging.

RWEverything-Treiber (RwDrv) und info_efi.exe

Einen ersten Hinweis liefert ein von den Angreifern entwickeltes Tool, das bei Ausführung Informationen über Low-Level-Systemeinstellungen in ein Textfile schreibt. Dieses Tool wurde zusammen mit LoJax-Samples entdeckt. Folgende Abbildung zeigt einen Ausschnitt aus dem Logfile, das durch das Tool „`info_efi.exe`" erstellt wird.

```
Get SMBIOS..
SMBIOS:
^@^X^@^@^A^BRê^C^@■Ð    |^@^@^@^@■^G^D^F^@^@Phoenix Technologies LTD^@6.00^@05/19/2017^@^@^A^[^A^@^A^B^C
^DUM^S■ÍÛW^MÓ\■^I~-(f^F^@^@UMware, Inc.^@UMware Virtual Platform^@None^@UMware-
                    ^@^@^B^D^B^@^A^B^C^D^@^@^@^@^@^A^@Intel Corporation^@440BX Desktop Reference Platf
orm^@None^@None^@^@^C^U^C^@^A^A^B^C^D^C^C^C^C4^R^@^@^@^@^@^@No Enclosure^@N/A^@None^@No Asset Tag^@^@^D*
^D^@^A^C^B^Bé^F  ^@ÿû■^O^C^B^@^@u¸^KA^D^U^@^U^@ÿÿ^@^@^@^A^A^@$^@^B^@CPU #000^@GenuineIntel^@Intel(R) Cor
e(TM) i5-7400 CPU @ 3.00GHz^@^@^E.^E^@^C^D^C^C^O^L^@^X^E^B^O^F^@^G^@^H^@        ^@
^@^K^@^L^@^M^@^N^@^O^@^P^@^Q^@^R^@^S^@^T^@^D^@^@^F^L^F^@^Aÿ^@^P^A
```

Abbildung 5 // Auszug aus Logfile, generiert durch info_efi.exe

Um die entsprechenden Informationen auszulesen, nutzt das Tool einen Treiber namens `RwDrv.sys`. Dieser gehört zu einem Tool namens RWEverything, welches frei im Netz verfügbar ist [14] und Low-Level-Einstellungen des Systems auslesen kann, inklusive PCI Express, Speicher, PCI Option ROMs und anderen. Da dieser Treiber zu einer legitimen Software gehört, ist er auch mit einem gültigen Code-Signing-Zertifikat ausgestattet.

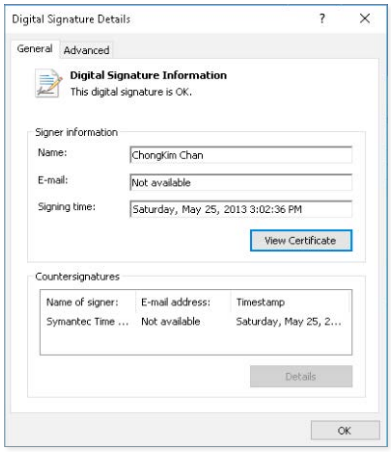

Abbildung 6 // Code-Signing-Zertifikat von RwDrv.sys

Mithilfe der zur RWEverything-Software gehörenden GUI kann sich der Nutzer alle Daten komfortabel anzeigen lassen.

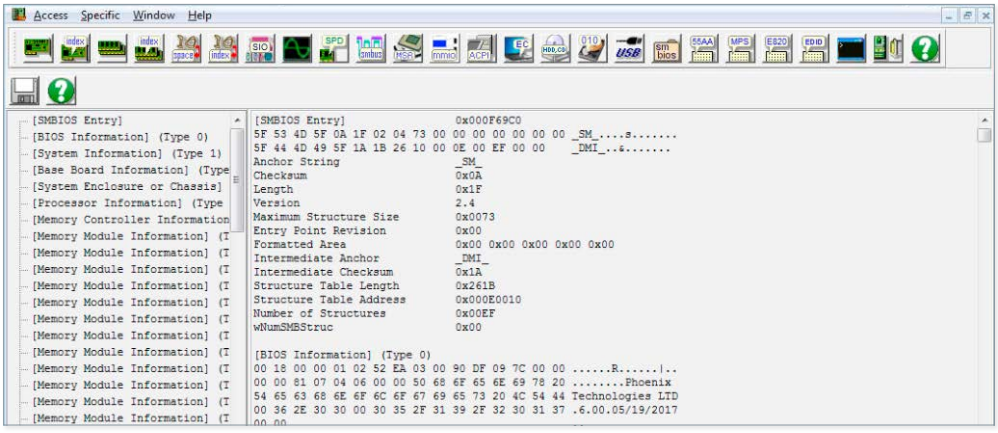

Abbildung 7 // Screenshot der RWEverything-GUI

Die Entdeckung des `info_efi`-Tools war für uns ein erster Hinweis darauf, dass ein LoJax UEFI-Modul tatsächlich existieren könnte: Für ein Update der Firmware eines System benötigt man Informationen über den Hersteller, die Versionsnummer etc. Um bekannte Schwachstellen, die Userland-Prozessen Zugriff auf

den SPI-Flash-Speicher geben, ausnutzen zu können, benötigen die Angreifer so viele Informationen wie möglich über die vom System genutzte Hardware. Info_efi liefert genau diese Informationen.

Einen letzten Hinweis auf der Suche nach Sednits erstem UEFI Rootkit gab uns die Entdeckung zweier weiterer Tools: eines, welches den SPI-Flash-Speicher ausliest und eines, welches ihn mit neuen Daten füllt.

Auslesen des SPI-Flash-Speichers

Zunächst stießen wir auf `ReWriter_read.exe`, die den kompletten Code für das Auslesen des SPI-Flash-Speichers mithilfe von `RwDrv.sys` liefert. Damit der Treiber die notwendigen Aktionen ausführen kann, muss das Auslesetool die korrekten IOCTL-Codes verschicken. Obwohl `RwDrv.sys` viele verschiedene IOCTL-Codes unterstützt, verwenden sowohl Auslese- als auch Schreibtool nur vier davon.

Tabelle 1 **Durch RwDrv.sys unterstützte IOCTLs**

IOCTL-Code	Beschreibung
0x22280c	Schreibt in speichergemappten I/O Bereich
0x222808	Liest speichergemappten I/O Bereich aus
0x222840	Liest ein Dword aus einem gegebenen PCI-Konfigurations-Register aus
0x222834	Schreibt ein Byte in ein gegebenes PCI-Konfigurations-Register

`ReWriter_read` erstellt zunächst einen Service mit dem eingebetteten Kernel-Treiber `RwDrv.sys` und loggt Informationen zur UEFI/BIOS-Konfiguration, nämlich die Werte der drei Felder im BIOS Control Register (BIOS_CNTL): BIOS Lock Enable (BLE), BIOS Write Enable (BIOSWE) und SMM BIOS Write Protect Disable (SMM_BWP). `ReWrite_read` selbst nutzt diese Werte tatsächlich nicht. Warum sie dennoch sehr wichtig für das Tool sind, erläutert der folgende Absatz.

Als nächstes hat das Tool die Aufgabe, die Region Base Adresse des BIOS im SPI-Flash-Speicher sowie seine Größe zu lesen. Diese Information befindet sich im SPI Host Interface Register „BIOS Flash Primary Region". Alle Host Interface-Register sind im Root Complex Register Block (RCRB) speichergemappt, dessen Base Adresse aus dem entsprechenden PCI-Konfigurations-Register ausgelesen werden kann. ReWriter erhält diese Adresse, indem es `ReWriter_read` verwendet und den korrekten Offset liest (0xF0 in unserem Fall). Sind BIOS-Adresse und -Größe bekannt, liest das Auslesetool die relevanten Informationen aus dem SPI-Flash-Speicher und schreibt es in eine Datei auf der Festplatte. Abbildung 8 illustriert den Leseprozess im SPI-Flash-Speicher. Erläuterungen zu den Abkürzungen finden Sie im *Glossar*.

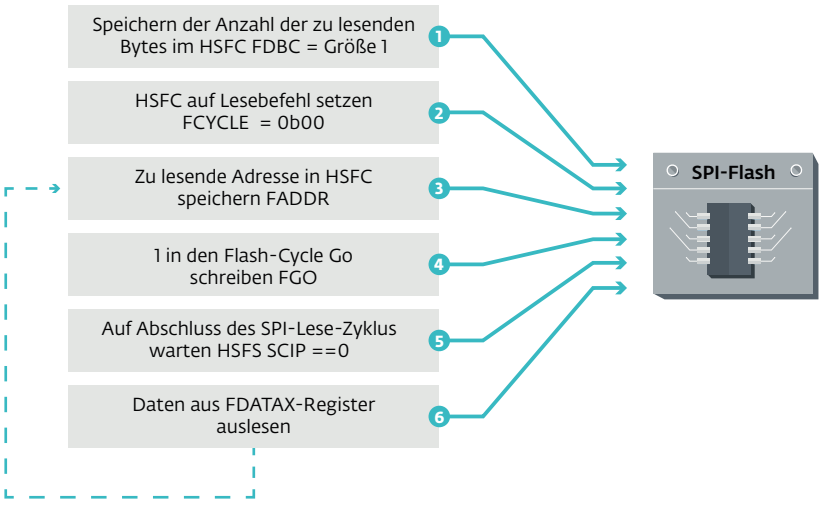

Abbildung 8 // Sequenz der Operationen, um SPI-Flash-Speicher auszulesen

Bis auf die ersten beiden Schritte, die nur ein einziges Mal ausgeführt werden, werden die Operationen wiederholt, bis alle Daten aus dem Speicher ausgelesen worden sind. [15] beschreibt diesen Prozess genauer. ReWriter_read validiert dann die Größe des ausgelesenen Images. Es parst den Flash-Deskriptor des Images, um den Speicherbereich des BIOS, des Gigabit Ethernets (GbE) und der Management Engine (ME) in Erfahrung zu bringen. Mithilfe dieser Informationen kann das Auslesetool die Größe des gesamten SPI-Flash-Speichers berechnen. Ist die Größe dieselbe wie die Größe der BIOS Flash Primary Region Registers, wird das Image als passend angenommen.

Patchen der UEFI-Firmware

Das nächste Element ist eine `ReWriter_binary.exe` genannte Datei. Sie enthält die Informationen, mit deren Hilfe wir abschließend nachweisen konnten, dass Sednit bis auf die Firmware hinunter angreift. Diese Datei enthält den Code, um das erstellte UEFI-Image zu patchen und die trojanisierte Version zurück in den SPI-Flash-Speicher zu schreiben. Der folgende Abschnitt soll diese Binärdatei im Detail beschreiben.

Wurde der Inhalt des Flash-Speichers erfolgreich ausgelesen und validiert, wird das schädliche UEFI-Modul dem Image hinzugefügt. Hierfür muss das UEFI-Image zunächst schrittweise geparst werden, um die nötigen Informationen für diese Aufgabe zu erhalten.

Die im UEFI-Image gespeicherten Daten werden im Firmware File System (FFS) abgelegt, welches eigens dafür gedacht ist, Firmware-Images zu speichern. Laufwerke enthalten Dateien, die per GUID identifiziert werden. Jede Datei ist dabei typischerweise in mehrere Sektionen aufgeteilt, von denen jede eine PE/COFF-.exe, die das jeweilige UEFI-Image darstellt, enthält. Der Screenshot von UEFITool [16], einer Open Source Software zur Bearbeitung von UEFI-Firmware-Images, in Abbildung 9 soll helfen, das Layout nachzuvollziehen.

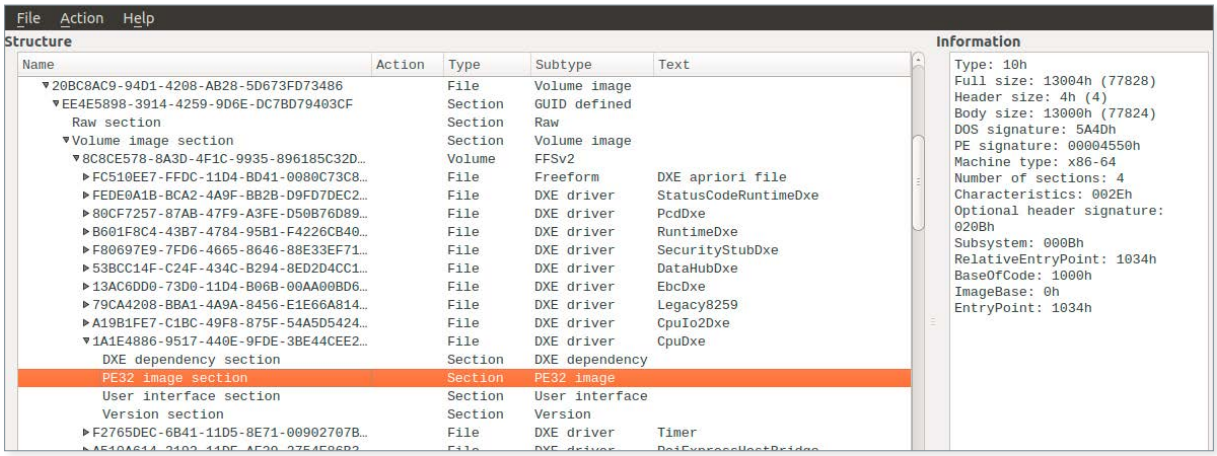

Abbildung 9 // Beispiel eines UEFI-Firmware-Images in UEFITool

`ReWriter_binary` parst alle Firmware-Laufwerke, die es in der BIOS-Region im SPI-Flash-Speicher findet und sucht nach bestimmten Dateien:

- 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;
```

Abbildung 10 // Output des Hex-Ray-Decompilers für die Routine, die die Firmware-Laufwerke parst

Ip4Dxe und NtfsDxe sind DXE-Treiber. In der UEFI-Firmware sind DXE-Treiber PE/COFF-Images, die entweder die Hardware abstrahieren oder Services erstellen, die durch andere DXE-Treiber oder UEFI-Anwendungen genutzt werden können. Solche Treiber werden von der DXE Foundation durch den DXE-Dispatcher (DXE Core) zu Beginn des Boot-Prozesses identifiziert und geladen. Danach sind alle Services, die durch UEFI-Anwendungen bereitgestellt werden (z.B. der OS-Loader) verfügbar. Normalerweise werden alle DXE-Treiber im gleichen Laufwerk gespeichert. Der DXE-Dispatcher kann sich jedoch in einem separaten befinden.

`ReWriter_binary` benötigt Ip4Dxe nur als Indikator dafür, dass das Laufwerk, welches gerade geparst wird, dasjenige mit den DXE-Treibern ist. Wie später noch genauer beschrieben wird, ist ein solches Laufwerk dann Kandidat für die Installation des gefälschten DXE-Treibers. Weiterhin sucht ReWriter_binary nach DXE Core und fügt das entsprechende Laufwerk ebenfalls als möglichen Kandidaten für das Rootkit hinzu. Der verfügbare Speicherplatz auf den Laufwerken wird festgestellt und dient später dazu sicherzustellen, dass genug Platz für den schädlichen Treiber ist.

NtfsDxe ist der AMI NTFS DXE-Treiber. Befindet er sich auf dem Firmware-Laufwerk, wird sein Speicherort vermerkt und später verwendet, um die Datei vom Laufwerk zu entfernen. Im Abschnitt zum UEFI-Rootki erläutern wir, warum das Tool diesen Treiber entfernt.

Die Informationen in Bezug auf das SmiFlash-Image werden gespeichert, aber nicht weiter in der Malware verwendet. Interessanterweise ist dieses Image aber verwundbar [17]. Wir nehmen deshalb an, dass die Entwickler bei Sednit aktuell daran arbeiten, hierfür einen Exploit zu entwickeln. Hiermit wäre es ihnen möglich, den SPI-Flash-Speicher selbst auf gut konfigurierten Systemen, die im Moment noch sicher sind, zu kompromittieren. Aktuell kann das Tool lediglich die BIOS-Region schlecht konfigurierter oder relativ alter Systeme (Motherboards älter als Versionen von 2008 ohne Platform Controller Chipsets) infizieren.

Nachdem die benötigten Metadaten ausgelesen wurden, patcht `ReWriter_binary` das ausgelesene UEFI-Image mit seinem eigenen DXE-Treiber. Es erstellt hierfür zunächst die Datei-Header-Struktur (EFI_FFS_FILE_HEADER). Dann wählt es das Ziellaufwerk entsprechend des Speicherortes von IpDxe und DXE Core und dem verfügbaren Speicherplatz aus. `ReWriter_binary` fügt zudem einen komprimierten Abschnitt mit dem PE-Image und einen User-Interface-Abschnitt ein, die den menschenlesbaren Namen der Datei, SecDxe, beinhaltet. Der komprimierte Abschnitt wird an den File-Header angehängt und an das Ende des Laufwerks geschrieben – dort, wo sich freier Speicherplatz befindet.
Abbildung 11 zeigt die Dateistruktur in UEFITool.

Abbildung 11 // SecDxe (Darstellung in UEFITool)

Zu guter Letzt wird der NtfsDxe-Treiber, sofern er sich noch im Image befindet, entfernt. Da Firmware-Dateisysteme Dateien und ihren Inhalt sequenziell speichern, ist das recht einfach erledigt:

- das Tool ermittelt den Abstand zum freien Speicherplatz am Ende des Laufwerks,
- das NtfsDxe-Image wird durch 0xFF Bytes überschrieben,
- der nachfolgende Teil des Firmware-Laufwerks wird ab dem Offset, an dem sich NtfsDxe befand, kopiert,
- der Rest des Dateisystems wird mit 0xFF Bytes ausgefüllt (also freiem Speicher).

Schreiben der gepatchten Firmware in den SPI-Flash-Speicher

Ist das Firmware-Image erfolgreich modifiziert, muss es zurück in den SPI-Flash-Speicher geschrieben werden. Bevor wir jedoch näher auf diesen Prozess eingehen, wollen wir kurz einige BIOS-Schutzmechanismen gegenüber unerlaubtem Schreibzugriff erläutern. Andere existierende Mechanismen, wie BIOS Range Write Protection, werden ausgelassen, da sie nicht von der `ReWriter_binary` geprüft werden.

Es gibt einige Schutzmechanismen, mit denen das BIOS vor unberechtigten Schreibzugriffen geschützt werden soll. Diese sind jedoch nicht von vornherein aktiviert. Die Firmware ist dafür zuständig, sie passend zu konfigurieren. Die Konfiguration lässt sich im BIOS Kontrollregister (BIOS_CNTL) nachvollziehen. In diesem Register ist das BIOS-Write-Enable-Bit (BIOSWE) enthalten, welches auf 1 gesetzt sein muss, um ins BIOS schreiben zu können. Ein weiteres Bit dient dazu, BIOSWE zu schützen: das BIOS Lock Enable (BLE). Dieser Mechanismus soll bei Aktivierung dafür sorgen, dass das BIOSWE-Bit auf 0 gesetzt ist. Die Implementierung ist jedoch relativ leicht angreifbar. Wird ein Request gesendet, das BIOSWE-Bit auf 1 zu setzen, geschieht dies auch. Erst dann löst der Vorfall eine Unterbrechung des Systemmanagements (SMI) aus, dessen Handler dann dafür verantwortlich ist, das BIOSWE-Bit zurück auf 0 zu setzen.

Diese Art der Implementierung birgt mehrere Probleme: Zunächst ist die Implementierung des SMI-Handlers den Entwicklern der Firmware überlassen. Implementiert die Firmware den Handler nicht, ist das BLE-Bit komplett nutzlos, da es dann keine Routine gibt, die das BIOSWE Bit zurück auf 0 setzt. Zweitens liegt eine Race Condition-Schwachstelle vor [18], mit der der Mechanismus umgangen werden kann, selbst wenn der SMI-Handler korrekt implementiert wurde. Um die Schwachstelle auszunutzen, muss ein Angreifer einen Thread starten, der das BIOSWE-Bit kontinuierlich auf 1 setzt, während ein anderer Thread Daten in den SPI-Flash-Speicher schreibt. Einer Veröffentlichung von Kallenberg und Wojtczuk zufolge [19] funktioniert diese Art Angriff sowohl auf Mehrkern-Prozessoren als auch auf Einzelkern-Prozessoren, wenn dort Hyper-Threading aktiviert ist.

Um diese Schwachstelle zu beheben, wurde dem BIOS_CNTL in der Platform Controller Hub (PCH)-Familie der Intel-Chipsets ein weiterer Schutzmechanismus hinzugefügt. Wenn seine Konfigurationsbits gesetzt werden, stellt SMM BIOS Write Project Disable (SMM_BWP) sicher, dass die BIOS-Region nur dann beschreibbar ist, wenn alle Kerne im System Management Modus (SMM) laufen und das BIOSWE-Bit auf 1 gesetzt ist. Dies schützt das System vor der oben erwähnten Race Condition-Schwachstelle. Allerdings muss auch SMM_BWP in

der Firmware aktiviert sein. Ist die Firmware unvollständig oder falsch konfiguriert, besteht das Problem weiterhin.

`ReWriter_binary` liest den Inhalt des BIOS-Kontrollregisters und wählt den passenden Pfad aus. Es prüft zunächst, ob BIOSWE gesetzt ist. Wenn ja, tritt es in die Schreibphase ein. Ist BIOSWE deaktiviert, überprüft es den Wert des BLE-Bits. Ist dieses nicht gesetzt, schaltet es das Bit um und beginnt mit dem Schreiben der gepatchten Firmware. Ist BLE aktiviert, sorgt es dafür, dass SMM_BWP deaktiviert ist und nutzt die Race Condition von oben. Ist SMM_BWP gesetzt, scheitert der Prozess. Abbildung 12 illustriert diesen Prozess.

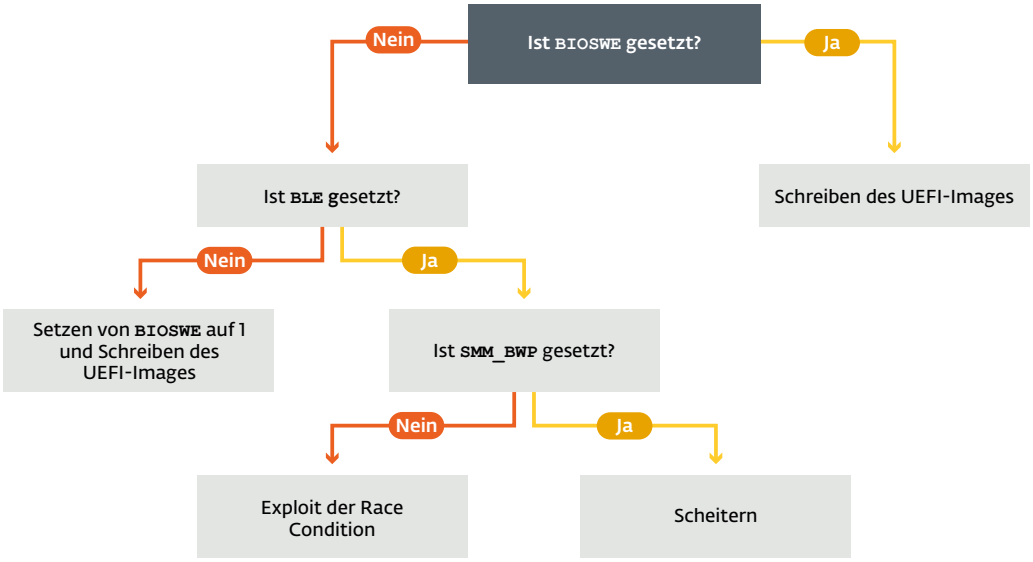

Abbildung 12 // Entscheidungsbaum des Schreibprozesses

Unter der Annahme, dass die Implementierung des `ReWriter_binary`, welches wir untersuchten, dieselbe war, mit dem das UEFI_Rootkit aufgespielt wurde, können wir sagen, dass entweder die Firmware den BIOS-Schreibschutz nicht korrekt konfiguriert hatte oder dass der Rechner des Opfers ein Chipset verwendete, welches älter als PCH war. Wäre das System modern und korrekt konfiguriert gewesen, hätte `ReWriter_binary` die UEFI-Firmware nicht erfolgreich modifizieren können. Wir haben allerdings auch beobachtet, dass die Angreifer während der Analyse der UEFI-Firmware nach dem verwundbaren SmiFlash UEFI-Image suchten. Dies legt die Vermutung nahe, dass sie bereits mit weiter entwickelten Technologien zum Umgehen der BIOS-Abwehrmechanismen experimentieren [17].

Analog zur oben beschriebenen Lese-Operation scheint die folgende Sequenz dazu zu dienen, in den SPI-Flash-Speicher zu schreiben:

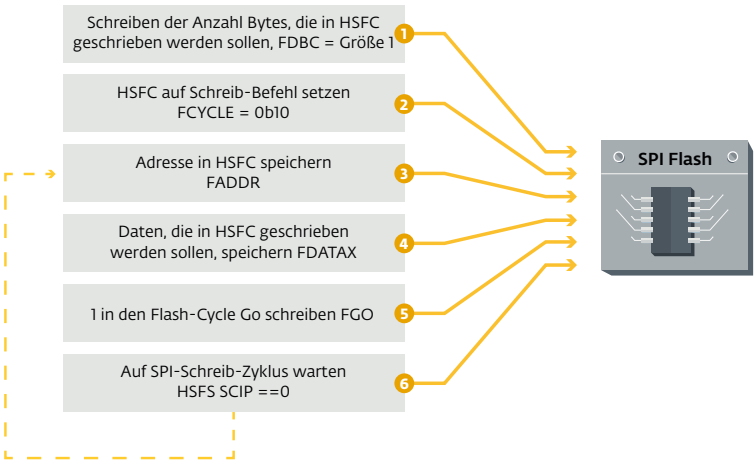

Abbildung 13 // Ablauf des Schreibprozesses auf SPI-Flash-Speicher

Bis auf die ersten beiden Schritte, die nur ein einziges Mal ausgeführt werden, werden die Operationen wiederholt, bis alle Daten aus dem Speicher ausgelesen worden sind.

Ist der Schreibprozess abgeschlossen, wird der Inhalt des SPI-Flash-Speichers erneut in der Datei `image.bin` gespeichert. Derselbe Integritätscheck, welcher durch `ReWriter_read` durchgeführt wurde, wird nun auf dem neuen Image durchgeführt. Dann wird das ausgelesene Image mit dem gepatchten im Speicher verglichen. Sind Bytes unterschiedlich, wird vermerkt, an welcher Adresse dies vorkommt. Ob sie sich unterscheiden oder nicht, hat jedoch keinen Einfluss auf die Malware. Die Information wird lediglich als Referenz für die Angreifer gespeichert.

Zu guter Letzt wird der Registry-Key gesetzt:

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

Dann wird der RwDrv-Service angehalten und deinstalliert. Es ist von zentraler Bedeutung, dass der Eintrag in der Registry auf diesen String gesetzt wird, weil das UEFI-Rootkit nach genau diesem String sucht, um ihn zu verändern und seine Aufgabe während des Startens von Windows auszuführen. Weitere Informationen zur Modifikation der Windows-Registry folgen im Kapitel zum UEFI-Rootkit und seinen Aufgaben.

5. LOJAX – TECHNISCHE ANALYSE

Obwohl das Tool für das Auslesen, Patchen und Beschreiben des SPI-Flash-Speichers an ein bestimmtes Firmware-Image angepasst ist und deshalb nicht problemlos auf jedem anderen System verwendet werden kann, lässt sich das komplette UEFI-Modul aus ihm extrahieren. Unser erster Schritt, um an das Modul heranzukommen, war eine Sichtung unserer Telemetriedaten. War es schon einmal aufgetaucht? Da es sich hier um ein UEFI-Modul handelt, verließen wir uns auf den neuen ESET UEFI-Scanner, um bis auf Firmware-Ebene herunterzukommen. Mithilfe der so erhaltenen Telemetriedaten konnten wir mindestens einen Fall eindeutig identifizieren, bei dem ein Sednit UEFI-Modul auf einem System platziert worden war. Das UEFI-Rootkit war also tatsächlich zum ersten Mal „in freier Wildbahn“ außerhalb von Forschungslaboren zum Einsatz gekommen.

Wir können nicht sicher nachvollziehen, wie die Tools auf das infizierte System gelangten, nehmen aber an, dass sie durch ein anderes Tool, vermutlich XAgent, platziert wurden. Da Auslese- und Schreibwerkzeuge zwar auf demselben System, aber zu verschiedenen Zeiten entdeckt wurden, gehen wir davon aus, dass die Angreifer in zwei Schritten vorgingen. Zunächst platzierten sie die Firmware auf dem Ziel-Rechner um sicherzugehen, dass ihr Patchingtool einwandfrei funktionieren würde. Dann luden sie es erneut hoch und patchten die Firmware tatsächlich. Obwohl wir jeweils nur eine Version des Schreib- und des Auslesetool fanden, ist es möglich, dass verschiedene Versionen für verschiedene, durch die Tools identifizierte Firmware existieren.

Abbildung 14 gibt einen allgemeinen Überblick über den Workflow des UEFI-Rootkits vor dem Start des Betriebssystems. Zunächst wird der SecDxe-Treiber durch den DXE-Dispatcher geladen. Es setzt einen Notify-Funktionscallback auf die `EFI_EVENT_GROUP_READY_TO_BOOT`-Event-Gruppe. Sobald die Firmware ein Boot-Device auswählen und den OS-Loader starten will, wird die Funktion aufgerufen. Der Callback führt dabei drei verschiedene Aktivitäten aus:

- er lädt den integrierten NTFS DXE-Treiber, um auf NTFSPartitionen zugreifen zu können
- er legt zwei Dateien in der Windows NTFS-Partition ab: `rpcnetp.exe` und `autoche.exe`
- er modifiziert den Registry-Key `‘HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\BootExecute’`:
  - Ausgangsversion: `‘autocheck autochk *’`
  - Modifikation: `‘autocheck autoche *’`.

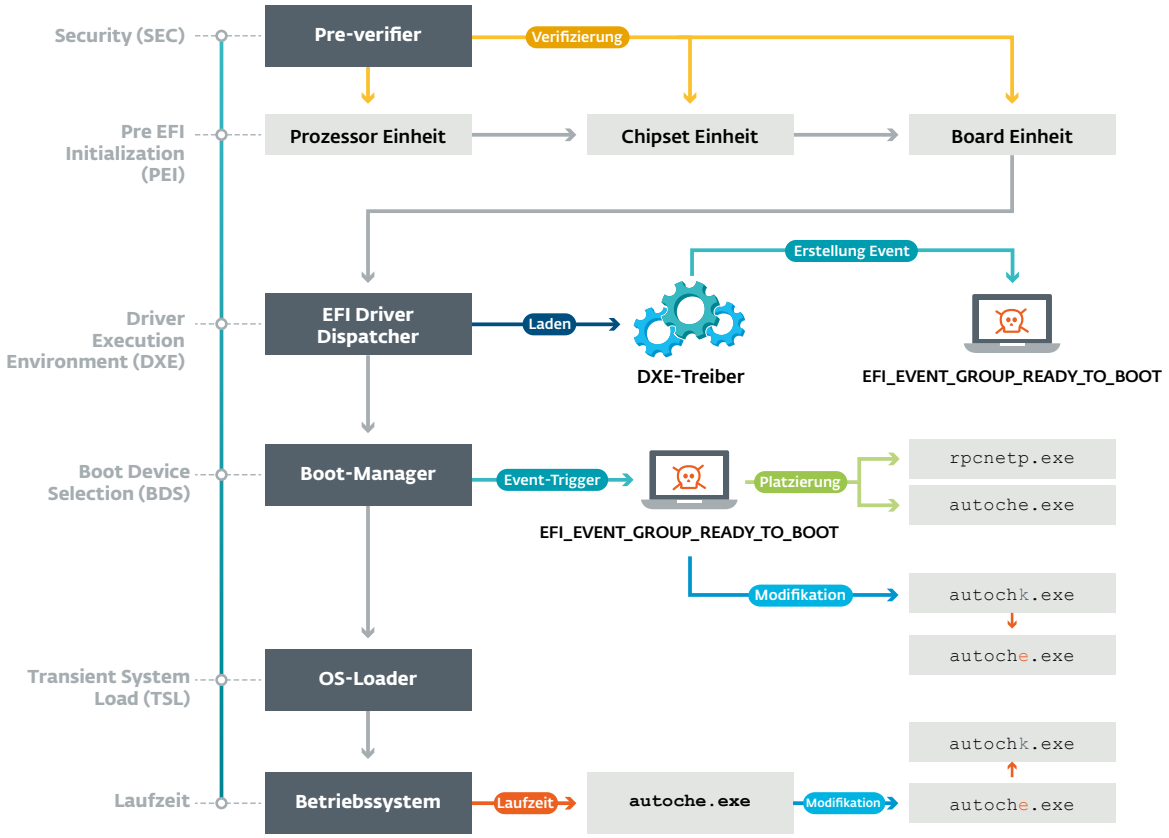

Abbildung 14 // Boot-Prozess eines mit dem UEFI-Rootkit infizierten Systems

SecDxe: Der modifizierte DXE-Treiber

Nachdem wir nun die Details, die zum Deployment des UEFI-Rootkits nötig sind, besprochen haben, soll das folgende Kapitel die Vorgänge auf einem infizierten System erläutern. Wir beginnen mit der Beschreibung des Rootkits selbst und erläutern dann die Event-Kette bottom-up bis auf Betriebssystemebene.

Sednits UEFI-Rootkit ist ein DXE-Treiber mit der GUID `682894B5-6B70-4EBA-9E90-A607E5676297`. Da er unsigniert ist, wird er nicht auf einem System, auf dem Secure Boot aktiviert ist, ausgeführt. Ist er einmal auf einem der Firmware-Laufwerke gelaufen, lädt ihn die DXE Foundation bei jedem Hochfahren neu.

SecDxe ist ein kleinerer DXE-Treiber mit zwei Hauptaufgaben. Er installiert ein Protokoll mit der GUID `832d9b4d-d8d5-425f-bd52-5c5afb2c85dc`, welches aber niemals verwendet wird. Danach erstellt er ein Event, welches mit einer Notify-Funktion verbunden ist. Diese Funktion wird dann aufgerufen, wenn die `EFI_EVENT_GROUP_READY_TO_BOOT`-Eventgruppe aufgerufen wird, was wiederum immer dann geschieht, wenn der Boot-Manager ein Gerät auswählt, von dem gebootet werden soll.

```
__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;
}
```

Abbildung 15 // Durch Decompiler ausgegebene Hex-Rays für die Event-Routine

Die Notify-Funktion implementiert das schädliche Verhalten des Sednit UEFI-Rootkits. Sie schreibt die Aufgaben in das NTFS-Dateisystem von Windows. Da UEFI-Firmware normalerweise nur auf der Partition arbeitet, auf der EFI liegt, ist ein NTFS-Treiber im Allgemeinen nicht enthalten. Nur FAT-basierte Systeme werden als Boot-Partitionen unterstützt. So ist es nicht zwingend notwendig, dass die UEFI-Firmware einen NTFS-Treiber enthält. SecDxe bringt ebenfalls seinen eigenen NTFS-Treiber mit. Der Treiber wird zunächst geladen und mit dem Laufwerk verbunden. Dann installiert es ein `EFI_SIMPLE_FILE_SYSTEM_PROTOCOL` auf Laufwerken mit NTFS-Partition und ermöglicht so dateibasierten Zugriff.

Da nun die Windows-Partition soweit vorbereitet ist, platziert SecDxe `rpcnetp.exe` und `autoche.exe`. Im Anschluss wird `rpcnetp.exe` auf `%WINDIR%\SysWOW64` (für 64-bit Windows) bzw. auf `%WINDIR%\System32` (für 32-bit Windows) installiert. `Autoche.exe` wiederum wird auf `%WINDIR%\SysWOW6 4` installiert. Abbildung 16 verdeutlicht die Routine, mit der die Dateien abgelegt werden.

```
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);
  }
}
```

Abbildung 16 // Durch Decompiler ausgegebene Hex-Rays für den Schreibprozess auf der Platte

SecDxe öffnet nun `%WINDIR%\System32\config\SYSTEM`, die Datei, welche für den Registry-Abschnitt `HKLM\SYSTEM` verantwortlich ist. Es analysiert die Datei, bis es `'autocheck autochk *'` findet und ersetzt das `'k'` von `'autochk'` durch `'e'`. So wird `'HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\BootExecute'` auf `'autocheck autoche *'` gesetzt. Beim nächsten Start von Windows wird `autoche.exe` anstelle von `autochk.exe` gestartet.

Der NTFS-Treiber von Hacking Team

Wie oben angesprochen, nutzt SecDxe einen NTFS-Treiber. Wir gehen davon aus, dass Sednit keinen eigenen Treiber entwickelt hat, sondern eine Kopie des geleakten NTFS DXE-Treibers von Hacking Team kompilierte.

Dieser NTFS-Treiber nutzt das Open Source-Projekt ntfs-3g, einen Wrapper, der dafür sorgt, dass er als UEFI DXE-Treiber fungieren kann. Daher listet die INF-Datei mit Informationen zum Treiber des Hacking-Teams Dateinamen aus dem Projekt ntfs-3g auf. Die SecDxe-Treiber-Strings listen ebenfalls viele dieser Dateinamen.

- `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`

Ein weiterer interessanter Punkt ist, dass der Projektpfad derselbe ist wie der, der in vector-edk gefunden wurde, dem geleakten EFI-Development-Projekt von Hacking Team. Vector-edk besitzt dabei ein Unterprojekt NtfsPkg mit genau demselben Verzeichnislayout. Die Dateien mit dem ntfs-3g-Quellcode liegen auf demselben Pfad. Auch wenn die Pfade generisch sind, gehen wir davon aus, dass dies kein Zufall ist.

Vergleicht man den geleakten Quellcode mit den durch den Decompiler herausgegebenen Hex-Rays, wird eindeutig klar, dass es sich um dasselbe Projekt handelt. In Abbildung 17 vergleichen wir die Funktion `NtfsDriverBindingStart` aus `vector-edk/NtfsPkg/NtfsDxe/Ntfs.c`. Kommentare im Code von HT wurden für die leichtere Lesbarkeit entfernt. Logik und Anordnung der Funktionsaufrufe sind exakt gleich. Beide Projekte verwenden die gleiche Variable (LockedByMe) um den Status des Locks zu fixieren.

```
ControllerHandle_1 = ControllerHandle;
EfiDriverBindingProtocol = This;
LockedByMe = 0;
if ( fnNtfsAcquireLockOrFail() >= 0 )
  LockedByMe = 1;
Status = fnInitializeUnicodeCollationSupport(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 = InitializeUnicodeCollationSupport (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 ();
  }
```

Abbildung 17 //
Vergleich der Hex-Rays (Decompiler-Output) von Sednits NTFS-Treiber (links) und dem NTFS-Treiber von Hacking Team

Obiger Vergleich zeigt Code der Hacking-Team-Entwickler und ist im ntfs-3g-Quellcode nicht enthalten.

Wie im Kapitel zu `ReWriter_binary` angedeutet, versucht die .exe, den AMI NTFS-Treiber zu löschen. Uns stellte sich die Frage, warum der Treiber gelöscht wird, anstatt ihn für andere Zwecke zu nutzen. Die Antwort war recht einleuchtend: Da das Schreiben in das Dateisystem nicht unterstützt wird, konnten die Angreifer ihn nicht für ihre Zwecke verwenden. Es ist auch wahrscheinlich, dass Sednit auf einige Probleme gestoßen ist, weil ein anderer NTFS-Treiber bereits in der Firmware vorhanden war. Also entschieden sie sich einfach, sie zu entfernen. Zusätzlich zum Implementieren von Lese- und Schreibvorgängen erzwingt der Treiber von Hacking Team keine Dateiberechtigungen. Zum Beispiel ist es möglich, eine schreibgeschützte Datei zu überschreiben, ohne einen Fehler zu verursachen.

Wir haben nun einen Überblick über die verschiedenen Prozesse, die das UEFI-Rootkit ausführt, sowie die Gründe für die Verwendung der vector-edk von HT für den Bau des NTFS-Treibers gegeben. Die folgenden Kapitel sollen die Ergebnisse unserer Analyse des Payloads, den SecDxe platziert, wiedergeben.

autoche.exe vs. autochk.exe

Die bösartige `autoche.exe` wird verwendet, um den Small Agent `rpcnetp.exe` fest in der Firmware zu verankern. Wie in Abbildung 18 zu sehen ist, werden native Windows-API-Aufrufe genutzt, um diesen Dienst zu erstellen.

```
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);
```

Abbildung 18 // Prozess, mit dem autoche.exe die Persistenz von rpcnetp.exe sicherstellt

Hierbei ist anzumerken, dass der Name des Service derselbe wie der, den der legitime Computrace-Agent verwendet. Ist der Service einmal aufgesetzt, setzt er den `BootExecute`-Registry-Key auf seinen ursprünglichen Wert.

```
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);
```

Abbildung 19 // Wiederherstellung des ursprünglichen Registry-Wertes von BootExecute durch autoche.exe

Dieser Prozess läuft während des Hochfahrens von Windows ab, sodass der Nutzer nicht merkt, dass der `BootExecute`-Registrywert verändert wird. Hierbei sollte beachtet werden, dass `autochk.exe` zwar einige Ähnlichkeiten mit Computraces autochk.exe aufweist, z.B. API-Aufrufe und Einträge in die Registry, sich ansonsten aber sehr davon unterscheidet. Das Modul von Computrace ist um einiges größer und stellt die originale autochk.exe wieder her, anstatt den Registry-Eintrag zu ändern. Zudem speichert es den Small Agent selbst auf der Festplatte, während dies bei LoJax durch das UEFI-Rootkit übernommen wird.

rpcnetp.exe

Auch wenn der Small Agent `rpcnetp.exe` durch das UEFI-Rootkit platziert werden kann, wird diese Komponente vermutlich in den meisten Fällen des trojanisierten LoJax Small Agent nicht verwendet. Es erscheint wahrscheinlich, dass die Malware-Entwickler opportunistisch handelten und das Rootkit nur zu installieren versuchten, wenn es überhaupt möglich war – und dies auch nur bei lohnenswerten Zielen. Im Verlauf unserer Nachforschungen konnten wir mehrere Varianten des LoJax Small Agent identifizieren. Im IOC-Abschnitt am Ende dieses Papiers finden sich die Hashes und dazugehörigen schädlichen Domains/IP-Adressen. Wie oben bereits erwähnt, handelte es sich bei allen von uns gefundenen LoJax Small Agent Samples um trojanisierte Versionen desselben alten Computrace Small Agents von 2008.

Zwar konnten wir nicht beobachten, dass der LoJax-Agent weitere Module heruntergeladen oder installiert hätte, wissen aber mit Sicherheit, dass dies möglich ist. LoJax' zentrale Funktionalität ist es, sich unsichtbar und fest im System einzunisten. So ist es prädestiniert dafür, anderen Modulen Zugriff auf zentrale Ressourcen in einem System zu ermöglichen.

6. VORBEUGUNG UND GEGENMASSNAHMEN

Abschließend stellt sich die Frage, wie Systeme vor solchen Angriffen zu schützen sind. Hierfür bedarf es des komplexen Zusammenspiels vieler Faktoren. Zunächst einmal sollte als erste Verteidigungslinie Secure Boot aktiviert sein, da dann jede Firmware-Komponente, die durch die Firmware geladen wird, auf ihre Signatur überprüft wird. Nicht oder fehlerhaft signierte Software wird nicht zugelassen. Wir empfehlen daher ausdrücklich die Aktivierung dieses Features, um die Integrität der Firmware sicherzustellen und sich vor Angriffen auf die UEFI-Firmware zu schützen.

Wie jede andere Software auch, sollte die UEFI-Firmware regelmäßig geupdatet werden. Auf der Website des Herstellers Ihres Motherboards können Sie ermitteln, ob Sie die aktuellste Version nutzen.

Außerdem sollten Sie sicherstellen, dass alle Systeme in Ihrem Netzwerk moderne Chipsätze mit Platform Controller Hub verwenden (verbaut seit Intel Series 5). So schließen Sie die oben erläuterte Race Condition-Schwachstelle [18].

Für weitere Schutzmechanismen sind die Hersteller des UEFI/BIOS verantwortlich. Die auf der Plattform vorhandenen Sicherheitsmechanismen müssen korrekt durch die System-Firmware konfiguriert sein. Daher muss Firmware immer mit ihrer eigenen Absicherung im Hinterkopf gebaut werden. Glücklicherweise beschäftigen sich immer mehr Sicherheitsforscher mit der Firmware-Sicherheit und tragen so dazu bei, diesen Bereich zu verbessern und das Bewusstsein bei Firmware-Herstellern zu erhöhen. Auch sollte CHIPSEC [19] nicht unerwähnt bleiben – ein Open Source Framework, welches die Sicherheit auf Systemebene untersucht und Ihnen hilft, die korrekte Konfiguration Ihrer Plattform zu prüfen.

War eine Attacke auf die UEFI-Firmware hingegen erfolgreich, sind hochkomplexe Gegenmaßnahmen erforderlich. Selbst Security-Produkte können hier nicht helfen. Im beschriebenen Fall müsste der SPI-Flash-Speicher komplett neu bespielt werden, um das Rootkit zu entfernen – eine Aufgabe, die der Durchschnittsnutzer nicht selbst durchführen sollte. Eventuell kann ein Upgrade der UEFI-Firmware helfen, sofern es die gesamte BIOS-Region auf dem SPI-Flash-Speicher ersetzt.
Die einzige echte Alternative für das Flashen des SPI-Flash-Speichers ist der Austausch des Motherboards.

7. FAZIT

UEFI-Rootkits sind eines der gefährlichsten Werkzeuge im Arsenal von Angreifern, überstehen sie doch sowohl die Neuinstallation des Betriebssystems als auch den Austausch der Festplatte. Zudem sind sie extrem schwer zu entdecken und zu entfernen. Zwar ist es eine relativ komplexe Aufgabe, das UEFI-Image eines Systems zu modifizieren, doch es existieren nur sehr wenige Lösungen, die die UEFI-Firmware eines Systems analysieren und schädliche Module identifizieren. Ist ein System betroffen, kann die UEFI-Firmware nur durch komplettes Überschreiben mit einem exakt passenden Image gesäubert werden – eine für den

Durchschnittsnutzer kaum praktikable Lösung. Entsprechend werden vor allem Angreifer mit ausreichenden Ressourcen besonders lohnenswerte Ziele auch weiterhin für solche Angriffe ins Visier nehmen.

8. DANKSAGUNG

Unser Dank gilt den Herausgebern von http://opensecuritytraining.info/ für das umfangreiche und informative Material, das sie der Community bereitstellen. Vor allem der Kurs ‚Introduction to BIOS & SMM' [20] war sehr nützlich für uns, um die Interaktionen mit dem SPI-Flash-Chip zu untersuchen.

9. GLOSSAR

Detailinformationen finden Sie in den Intel-Spezifikationen [21].

- **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

10. QUELLEN

1 D. Alperovitch, „Bears in the Midst: Intrusion into the Democratic National Committee," Crowdstrike, 15. Juni 2016. [Online]. Available: *https://www.crowdstrike.com/blog/bears-midst-intrusion-democratic-national-committee/*.

2 US Department of Justice, Juli 2018. [Online]. Verfügbar unter: *https://assets.documentcloud.org/documents/4598895/DOJ-Russia-DNC-Hack-Indictment.pdf*.

3 G. Corera, „How France's TV5 was almost destroyed by ‚Russian hackers'," BBC, 10. Oktober 2016. [Online]. Verfügbar unter: *https://www.bbc.com/news/technology-37590375*.

4 L. Matsakis, „Hack Brief: Russian Hackers Release Apparent IOC Emails in Wake of Olympics Ban," WIRED, 1. Januar 2018. [Online]. Verfügbar unter: *https://www.wired.com/story/russian-fancy-bears-hackers-release-apparent-ioc-emails/*.

5 ESET Research, „En Route with Sednit," ESET, 2016. [Online]. Verfügbar unter: *http://www.welivesecurity.com/wp-content/uploads/2016/10/eset-sednit-full.pdf*.

6 ESET Research, „Sednit adds two zero-day exploits using ‚Trump's attack on Syria' as a decoy," ESET, 9. Mai 2017.[Online].Verfügbarunter:*https://www.welivesecurity.com/2017/05/09/sednit-adds-two-zero-day-exploits-using-trumps-attack-syria-decoy/*.

7 ESET Research, „Sednit update: Analysis of Zebrocy," ESET, 24. April 2018. [Online]. Verfügbar unter: https://www.welivesecurity.com/2018/04/24/sednit-update-analysis-zebrocy/.

8 P. Lin, „Hacking Team Uses UEFI BIOS Rootkit to Keep RCS 9 Agent in Target Systems," Trend Micro, 13. Juli 2015. [Online]. Verfügbar unter: https://blog.trendmicro.com/trendlabs-security-intelligence/hacking-team-uses-uefi-bios-rootkit-to-keep-rcs-9-agent-in-target-systems/.

9 WikiLeaks, „DerStarke 2.0," [Online]. Verfügbar unter: https://wikileaks.org/ciav7p1/cms/page_13763820.html.

10 Absolute, „Absolute Response to Arbor Research," Mai 2018. [Online]. Verfügbar unter: https://www.absolute.com/en-gb/resources/faq/absolute-response-to-arbor-research.

11 A. Ortega and A. Sacco, „Deactivate the Rootkit: Attacks on BIOS anti-theft," Core Security Technologies, 24. Juli 2009. [Online]. Verfügbar unter: https://www.blackhat.com/presentations/bh-usa-09/ORTEGA/BHUSA09-Ortega-DeactivateRootkit-PAPER.pdf.

12 V. Kamlyuk, S. Belov and A. Sacco, „Absolute Backdoor Revisited," BlackHat, Juni 2014. [Online]. Verfügbar unter: https://www.blackhat.com/docs/us-14/materials/us-14-Kamluk-Computrace-Backdoor-Revisited-WP.pdf.

13 ASERT team, „Lojack Becomes a Double-Agent," 1. Mai 2018. [Online]. Verfügbar unter: https://asert.arbornetworks.com/lojack-becomes-a-double-agent/.

14 „RWEverything Read & Write Everything," [Online]. Verfügbar unter: http://rweverything.com/.

15 A. Matrosov and E. Rodionov, „UEFI Firmware Rootkits: Myths and Reality," Black Hat Asia, 2017. [Online]. Verfügbar unter: https://www.blackhat.com/docs/asia-17/materials/asia-17-Matrosov-The-UEFI-Firmware-Rootkits-Myths-And-Reality.pdf.

16 „GitHub repository for UEFITool," [Online]. Verfügbar unter: https://github.com/LongSoft/UEFITool.

17 Cylance, „Researchers Disclose Vulnerabilities in GIGABYTE BRIX Systems," [Online]. Verfügbar unter: https://threatvector.cylance.com/en_us/home/gigabyte-brix-systems-vulnerabilities.html.

18 Carnegie Mellon University SEI CERT, „Vulnerability Note VU#766164, Intel BIOS locking mechanism contains race condition that enables write protection bypass," [Online]. Verfügbar unter: https://www.kb.cert.org/vuls/id/766164.

19 C. Kallenberg and R. Wojtczuk, „Speed Racer: Exploiting an Intel Flash Protection Race Condition," Januar 2015. [Online]. Verfügbar unter: https://bromiumlabs.files.wordpress.com/2015/01/speed_racer_whitepaper.pdf.

20 J. Butterworth, „Advanced x86: Introduction to BIOS & SMM," 2014. [Online]. Verfügbar unter: http://opensecuritytraining.info/IntroBIOS.html.

21 Intel, „Intel 7 Series / C216 Chipset and Family Platform Controller Hub (PCH)," Juni 2012. [Online]. Verfügbar unter: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/7-series-chipset-pch-datasheet.pdf.

11. IOCs

ReWriter_read.exe

ESET-Bezeichnung

Win32/SPIFlash.A

SHA-1

`ea728abe26bac161e110970051e1561fd51db93b`

ReWriter_binary.exe

ESET-Bezeichnung

Win32/SPIFlash.A

SHA-1

`cc217342373967d1916cb20eca5ccb29caaf7c1b`

SecDxe

ESET-Bezeichnung

EFI/LoJax.A

SHA-1

`f2be778971ad9df2082a266bd04ab657bd287413`

info_efi.exe

ESET-Bezeichnung

Win32/Agent.ZXZ

SHA-1

`4b9e71615b37aea1eaeb5b1cfa0eee048118ff72`

autoche.exe

ESET-Bezeichnung

Win32/LoJax.A

SHA-1

`700d7e763f59e706b4f05c69911319690f85432e`

Small agent EXE

ESET-Bezeichnung

Win32/Agent.ZQE

Win32/Agent.ZTU

SHA-1

`1771e435ba25f9cdfa77168899490d87681f2029`

`ddaa06a4021baf980a08caea899f2904609410b9`

`10d571d66d3ab7b9ddf6a850cb9b8e38b07623c0`

`2529f6eda28d54490119d2123d22da56783c704f`

`e923ac79046ffa06f67d3f4c567e84a82dd7ff1b`

`8e138eecea8e9937a83bffe100d842d6381b6bb1`

`ef860dca7d7c928b68c4218007fb9069c6e654e9`

```
e8f07caafb23eff83020406c21645d8ed0005ca6
09d2e2c26247a4a908952fee36b56b360561984f
f90ccf57e75923812c2c1da9f56166b36d1482be
```

C&C-Server Domain-Bezeichnung

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

C&C-Server-IP-Adressen

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

Small agent DLL

Hier finden sich die DLLs, für die wir keine entsprechende .exe finden konnten.

ESET-Bezeichnungen

Win32/Agent.ZQE

SHA-1

```
397d97e278110a48bd2cb11bb5632b99a9100dbd
```

C&C-Server Domain-Bezeichnung

elaxo[.]org

C&C-Server-IP-Adressen

86.106.131[.]54