Dieser Artikel erschien ursprünglich in c't 18/2011, Seite 178

Tatort Internet

S02E04: Operation am offenen Herzen

Frank Boldewin - *03.10.2011*

Es ist einer der wenigen Samstagnachmittage, an denen die Sonne scheint. Ich überlege gerade, wie viel Fleisch ich wohl für den geplanten Grill-Abend einkaufen soll, als mein Handy klingelt. Mit nervöser Stimme eröffnet mir Hans, dass er sich anscheinend 'nen Virus eingefangen habe.

Als ich zurückfrage, wie er denn darauf komme, gerät der sonst so selbstsichere Maschinenbaustudent ins Stottern: „Äh, naja, ich hab mir doch gestern einen neuen Rechner gekauft, inklusive Microsoft Office. Allerdings nur als 30-Tage-Testversion und ich hatte kein Geld mehr für die Vollversion. Und da dachte ich …“, „… da lad ich mir mal eben einen Crack runter und spar mir das Geld“, vervollständige ich den Satz. „Und jetzt passieren auf einmal seltsame Dinge auf deinem System, stimmt's?“

„Genau, aber woher weißt Du…?“, wundert sich Hans. Ich denk insgeheim: „Weil jeden Tag ein Dummer aufsteht“, während ich ihm ruhig erkläre, dass er keineswegs der Erste ist, der sich mit solchen Problemen an mich wendet.

Dann zieh ich ihm die Fakten Stück für Stück aus der Nase: Nach dem Start des angeblichen Cracks verschwand die ausführbare Datei wie von Zauberhand vom Desktop. Sonst passierte nichts – jedenfalls nichts Sichtbares. Doch seitdem signalisierte der Router quasi permanenten Internet-Verkehr, obwohl keine Anwendungen gestartet waren, die Traffic hätten produzieren können. Nach einem Neustart war kurz Ruhe und dann ging das Geblinke der Router-LED erneut los. Da Hans noch etwas bei mir gut hat, schwing ich mich auf mein Fahrrad und mach mich auf den Weg zu ihm.

Ein erster Blick auf das System und mein siebter Sinn sagen mir, dass ich hier an der Oberfläche nicht fündig werde, sondern deutlich tiefer bohren muss. Zu dumm, dass ich mein Memory-Dump-Analyse-System gestern im Büro gelassen habe. Also werd ich wohl den Rechner direkt analysieren müssen: lokales Kernel-Debugging – quasi eine Operation am offenen Herzen.

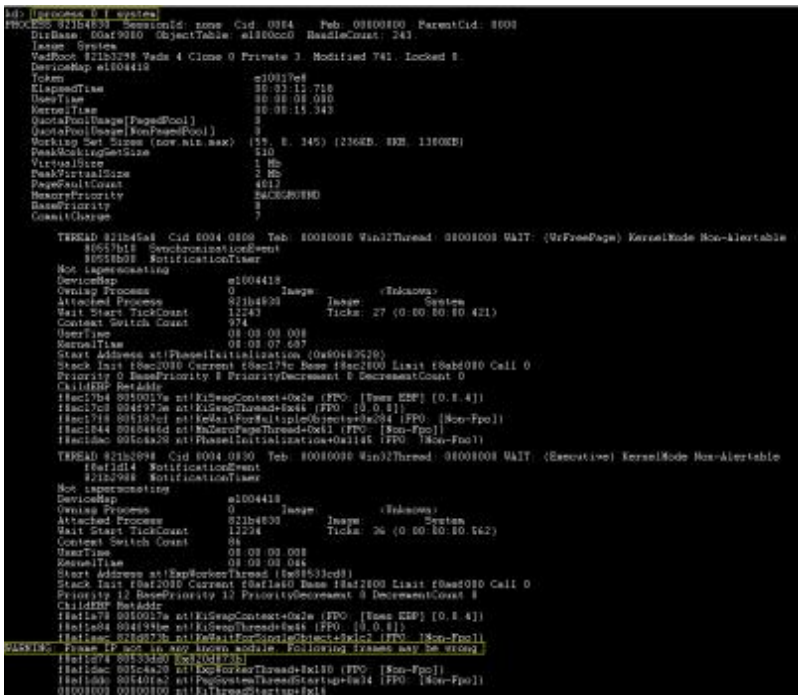
„No risk, no fun!“, schießt mir durch den Kopf, während ich in Gedanken die Ärmel hochkremple. Ganz real installiere ich von meinem schreibgeschützten USB-Stick

Sklavenarbeit

Ein Interrupt-Handler stellt dazu via `ExQueueWorkItem` beziehungsweise `IoQueueWorkItem` ein sogenanntes Work-Item mit einem Zeiger auf eine Callback-Routine in die Warteschlange. Kommt das Work-Item an die Reihe, entfernt einer der System Worker Threads es aus der Warteschlange und führt dann die Callback-Routine aus, die die anstehende Arbeit für den Handler erledigt. Diese Threads laufen immer unter dem System-Prozess mit der Prozess-ID 4.

English version

Genau solche Work-Items legt TDL über den Umweg sogenannter Asynchronous Process Calls (APCs) an. Mal sehen, ob ich auf Hansens PC wirklich welche finde. Dazu lasse ich mir mit `!process 0 f system` möglichst detailliert Thread- und Stack-Zustände des System-Prozesses anzeigen.



[http://www.heise.de/security/artikel/Tatort-Internet-Operation-am-offenen-Herzen-1338967.html?artikelseite=2;view=zoom;zoom=2]

[http://www.heise.de/security/artikel/Tatort-Internet-Operation-am-offenen-Herzen-1338967.html?artikelseite=2;view=zoom;zoom=2]

Ganz tief unten

Ich hangel mich weiter und schau mir nun mit `dt _driver_object` das Treiber-Objekt an, das laut Device-Object bei `0x82166c40` liegt. Wieder erscheint ein ungültiger Typ 0 – richtig wäre 4. Immerhin der Name ist auf den korrekten Unicode-String "`\Driver \atapi`" gesetzt und auch die Funktionen wie `DriverInit` und `DriverStartIO` zeigen auf den Original-Windows-Atapi-Treiber. Doch das „`MajorFunction Array`“, das die I/O-Operationen des Treibers wie `IRP_MJ_READ` und `IRP_MJ_WRITE` enthält, liegt erneut im bereits einschlägig bekannten Adressbereich `0x820dxxxx` – höchst verdächtig. Die schau ich mir doch mal genauer an.

English version

```
kd> dt device_object 0x821feaf0
nt!_DEVICE_OBJECT
   +0x000 Type             : 0n3
   +0x002 Size             : 0x234
   +0x004 ReferenceCount   : 0n0
   +0x008 DriverObject     : 0x82166c40 _DRIVER_OBJECT
   +0x00c NextDevice       : 0x82144040 _DEVICE_OBJECT
   +0x010 AttachedDevice   : 0x8217cab8 _DEVICE_OBJECT
   +0x014 CurrentIrp       : (null)
   +0x018 Timer            : (null)
   +0x01c Flags            : 0x5050
   +0x020 Characteristics  : 0n100
   +0x024 Vpb              : (null)
   +0x028 DeviceExtension  : 0x8217eba8 Void
   +0x02c DeviceType       : 7
   +0x030 StackSize        : 1 ''
   +0x034 Queue            : __unnamed
   +0x05c AlignmentRequirement : 1
   +0x060 DeviceQueue      : _KDEVICE_QUEUE
   +0x074 Dpc              : _KDPC
   +0x094 ActiveThreadCount : 0
   +0x098 SecurityDescriptor : 0xe101aaa0 Void
   +0x09c DeviceLock       : _KEVENT
   +0x0ac SectorSize       : 0
   +0x0ae Spare1           : 1
   +0x0b0 DeviceObjectExtension : 0x8217ed28 _DEVOBJ_EXTENSION
   +0x0b4 Reserved         : (null)
kd> dt driver_object 0x82166c40
nt!_DRIVER_OBJECT
   +0x000 Type             : 0n4
   +0x002 Size             : 0n168
   +0x004 DeviceObject     : 0x8201cb90 _DEVICE_OBJECT
   +0x008 Flags            : 4
   +0x00c DriverStart      : 0xf84d2000 Void
   +0x010 DriverSize       : 0x17480
   +0x014 DriverSection    : 0x821d9c28 Void
   +0x018 DriverExtension  : 0x82166ca0 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\atapi"
   +0x024 HardwareDatabase : 0x80fe6fd8 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xf84e7517        long  atapi!GsDriverEntry+0
   +0x030 DriverStartIo    : 0xf84d97c6        void  atapi!IdePortStartIo+0
   +0x034 DriverUnload     : 0xf84e3204        void  atapi!IdePortUnload+0
   +0x038 MajorFunction    : [28] 0x820d8446       long  +ffffffff820d8446
```

[http://www.heise.de/security/artikel/Tatort-Internet-Operation-am-offenen-Herzen-1338967.html?artikelseite=3;view=zoom;zoom=4]

[http://www.heise.de/security/artikel/Tatort-Internet-Operation-am-offenen-Herzen-1338967.html?artikelseite=3;view=zoom;zoom=4]

Dazu muss ich den Code gar nicht allzu genau studieren. Schon bei einer oberflächlichen Analyse wird mir klar, was hier passiert. Das mit dem Befehl `uf` erzeugte Disassembly zeigt dank Symbol-Tabelle viele bekannte Aufrufe der File System Runtime Library wie `FsRtlProcessFileLock` oder `FsRtlAllocatePool`. Darüber hinaus finden sich in den Funktionen typische Return-Codes wie `0xC0000123`

Routinemäßig überprüfe ich noch, ob sich die Usermode-Komponente ebenfalls ins Windows-API einklinkt. Das Kommando `!chkimg` gleicht die aktuellen Bibliotheksfunktionen mit dem Microsoft Symbol Server ab und entdeckt solche Hooks dabei sehr zuverlässig. Beim Check der System-Bibliotheken ntdll.dll und mswsock.dll meldet es Fehler, die ziemlich sicher auf Hooks zurückzuführen sind. Ich gehe ihnen allerdings jetzt nicht mehr weiter nach.

English version

```
kd> !chkimg -d ntdll.dll
    7c91deb6-7c91debe  5 bytes - ntdll!ZwProtectVirtualMemory
      [ b8 89 00 00 00:e9 4f 21 50 84 ]
    7c91ea32-7c91ea36  5 bytes - ntdll!NtWriteVirtualMemory (+0xb7c)
      [ b8 15 01 00 00:e9 d3 15 51 84 ]
    7c91eaec-7c91eaf0  5 bytes - ntdll!KiUserExceptionDispatcher (+0xba)
      [ 8b 4c 24 04 8b:e9 1b 15 4f 84 ]
15 errors : ntdll.dll (7c91deb6-7c91eaf0)
kd> !chkimg -d mswsock.dll
    719b405f-719b4063  5 bytes - mswsock!WSPCloseSocket
      [ 6a 44 68 d8 41:e9 a8 bf 0e 90 ]
    719b4342-719b4346  5 bytes - mswsock!WSPRecv (+0x2e3)
      [ 6a 44 68 b8 44:e9 c5 bc 48 8f ]
    719b5847-719b584b  5 bytes - mswsock!WSPSend (+0x1505)
      [ 6a 40 68 80 59:e9 c0 a7 50 8f ]
15 errors : mswsock.dll (719b405f-719b584b)
```

[http://www.heise.de/security/artikel/Tatort-Internet-Operation-am-offenen-Herzen-1338967.html?artikelseite=4;view=zoom;zoom=9]

[http://www.heise.de/security/artikel/Tatort-Internet-Operation-am-offenen-Herzen-1338967.html?artikelseite=4;view=zoom;zoom=9]

Die zusammengetragenen Fakten reichen mir, um mit den Dateinamen und markanten Windbg-Ausgaben via Google zu bestätigen, dass es sich hier tatsächlich um den neuesten Abkömmling der TDL-Rootkit-Familie handelt. Das TDL4-Rootkit ist derzeit das einzige Schadprogramm, das sich auch auf einem Windows 7 64 Bit im Kernel einnisten kann.

Boot-Spielchen

Eigentlich soll bei einem 64-Bit-Windows der Patchguard verhindern, dass Treiber in den Kernel geladen werden, die keine gültige Signatur tragen. Deshalb knipst TDL4 diese Schutzfunktion aus, indem es in einer sehr frühen Boot-Phase dem System vorgaukelt, im Systemrestore-Modus „WinPE“ zu booten. Denn der aktiviert Patchguard nicht.