Bundesamt für Verfassungsschutz

Köln, 06. Dezember 2019

BfV Cyber-Brief

Nr. 01/2019

- Hinweis auf aktuelle Angriffskampagne -



Kontakt:
Bundesamt für Verfassungsschutz
Cyberabwehr
☎ 0221/792-2600

© maxsim/Fotolia.com

Bedrohung der deutschen Wirtschaft durch Cyberangriffe der Gruppe WinNTI

Die Cyberabwehr des Bundesamtes für Verfassungsschutz warnt Wirtschaftsunternehmen vor Cyberangriffen durch die Cyberangreifergruppe WinNTI

Sachverhalt

Mitte 2019 wurde in den Medien über Cyberangriffe auf mehrere deutsche DAX-Konzerne berichtet, die durch die Cyberangreifergruppe WinNTI verübt worden sein sollen.

Nach Erkenntnissen der Cyberabwehr des Bundesamtes für Verfassungsschutz starteten die Angriffe vermutlich bereits im Jahre 2016. Durchgeführt wurden die Angriffe mit verschiedenen Varianten der gleichnamigen Malware WinNTI in der Version 3. Diese Version der Schadsoftware ist bereits seit 2013 im Einsatz, die neuesten Varianten wurden durch den Akteur im Jahr 2017 entwickelt.

Die Cyberabwehr des Bundesamtes für Verfassungsschutz geht von einer anhaltenden Angriffswelle durch den Akteur auf die deutsche Wirtschaft aus. Vor diesem Hintergrund sollen die wesentlichen Analyseergebnisse der Öffentlichkeit zur Verfügung gestellt werden. Dieser Cyber-Brief enthält aktuelle Detektionsregeln und technische Indikatoren (Indicators of Compromise), um eine mögliche Infektion durch WinNTI feststellen zu können.[1] Außerdem wird die Funktionsweise der aktuellsten Variante der WinNTI-Malware dargestellt.

Technische Analyse

Bei einem Angriff auf ein Unternehmensnetzwerk infiziert Winnti typischerweise mehrere Systeme und bewegt sich so von wenigen an das Internet angeschlossenen Systemen (internet-facing hosts) durch das gesamte Opfernetzwerk. Eine typische Infektion auf einem System erfolgt durch einen mehrstufigen Prozess und beginnt mit einem sog. DLL-Search-Order-Hijacking.

1 Die Cyberabwehr des Bundesamtes für Verfassungsschutz bedankt sich bei der **Deutschen Cyber-Sicherheitsorganisation GmbH (DCSO)** für die Prüfung der Detektionsregeln und die Bereitstellung zusätzlicher Indikatoren.

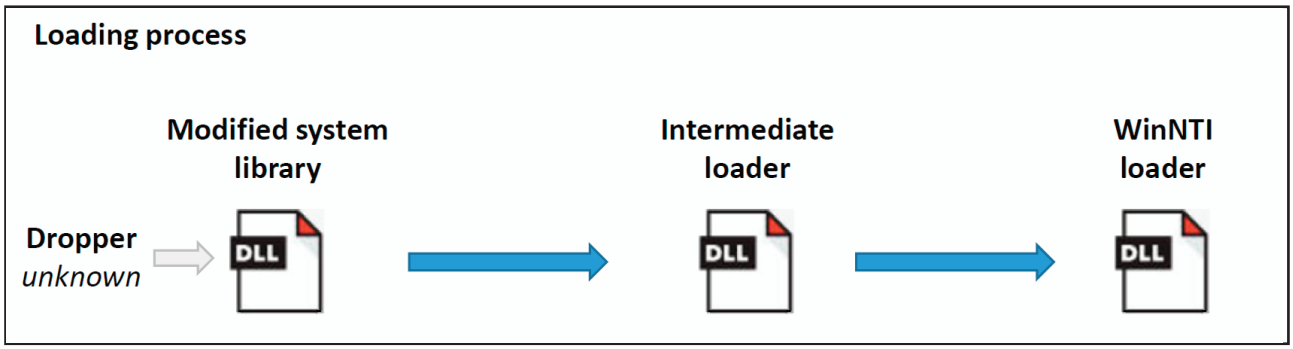


Abbildung 1: Prozesskette der Ausführung

Dabei wird eine legitime Systembibliothek durch den Angreifer modifiziert ***(modified system library)*** und auf dem Opfersystem platziert. Beim Start des Betriebssystems auf dem Zielrechner importiert ein legitimes, mit Systemrechten startendes Programm diese modifizierte Systembibliothek, was die Ausführungskette anstößt. Die modifizierte Systembibliothek verfügt über einen zusätzlichen Eintrag im eigenen Importverzeichnis zwecks Import einer Funktion, einer ebenfalls vom Akteur im System platzierten weiteren Datei ***(intermediate loader)***. Diese Bibliothek des Akteurs enthält Informationen, wo sich die letzte Stufe des Ausführungsprozesses des Akteurs ***(WinNTI loader)*** befindet und führt diese mit dem Windows-Dienstprogramm rundll32.exe aus. Bei einigen Varianten wird das Laden des ***WinNTI loaders*** unmittelbar über die ***modified system library*** veranlasst.

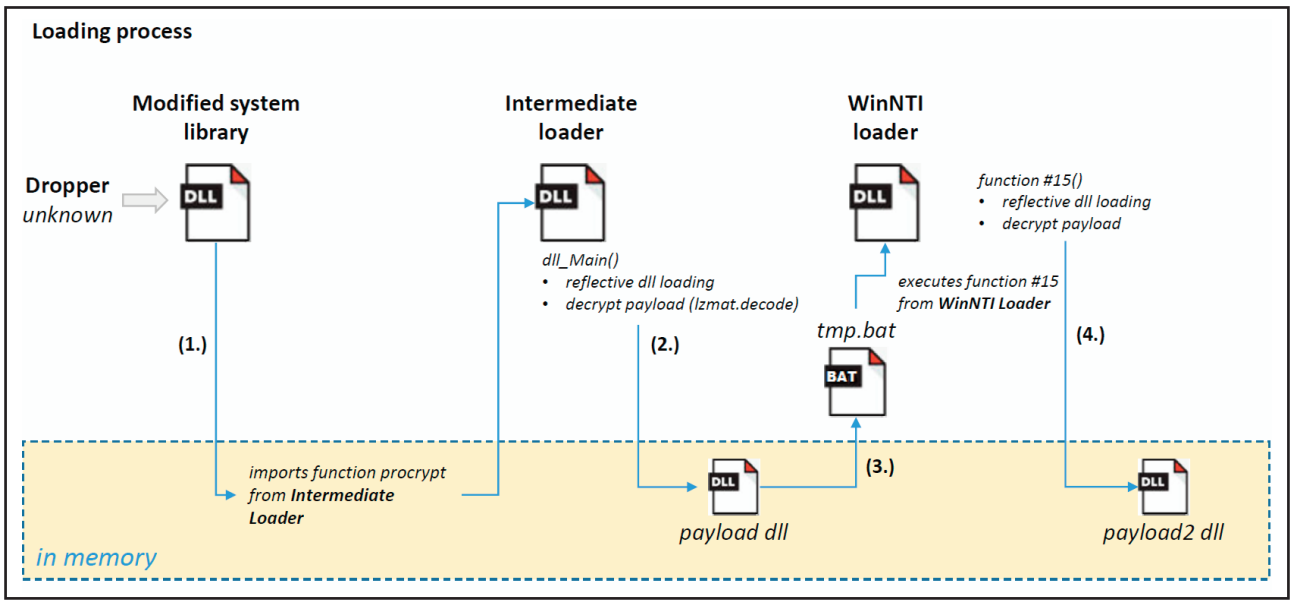


Abbildung 2: Detaillierter Ladeprozess der Malware

Der genaue Ablauf ist dabei wie folgt (siehe Abb. 2):

(1.) Zur Durchführung des DLL-Search-Order-Hijacking wird eine im gleichen Verzeichnis abgelegte, modifizierte Systembibliothek von einem legitimen Programm vorrangig geladen ***(modified system library)***, welche die Funktion procrypt [2] aus dem ***intermediate loader*** importiert. Dies veranlasst den Windows-Loader für ausführbare Dateien dazu, den intermediate loader in den Prozessspeicher des legitimen Programmes zu laden und die zugehörige Methode DllMain() mit dem Flag DLL_PROCESS_ATTACH aufzurufen (üblicher Vorgang zum Laden von DLLs).

2 Die vom intermediate loader exportierte Funktion kann auch anders heißen. Sie enthält Code, der h.E. nicht zur Ausführung kommt.

(2.) Die Funktion DllMain() des intermediate loaders dekryptiert zunächst die enthaltene ***intermediate payload dll*** und lädt diese anschließend per reflective dll loading in den Arbeitsspeicher. Im Anschluss wird die einzige von der ***intermediate payload dll*** exportierte Funktion aufgerufen (hier: lzmat_decode).

(3.) Die Funktion lzmat_decode der ***intermediate payload dll*** schreibt daraufhin ein Batch-File auf die Festplatte, das in Folge bei Ausführung die Funktion mit dem Ordinal 15 im finalen ***WinNTI Loader*** aufruft (Ordinalwert kann abweichen).

(4.) Die Funktion mit dem Ordinal 15 des Winnti Loaders dekryptiert zwei mit dem Hostnamen des Zielsystems via AES CFB verschlüsselte Datenbereiche. Hierbei handelt es sich um Shellcode für reflective dll loading und um die ***WinNTI payload dll*** (siehe Abbildung 2 ***payload2 dll***). Letztere wird mit Hilfe des Shellcodes in den Speicher geladen und dessen einzig exportierte Funktion wird aufgerufen.

Damit ist der Ladeprozess abgeschlossen und die Malware geht in den Ausführungsprozess.

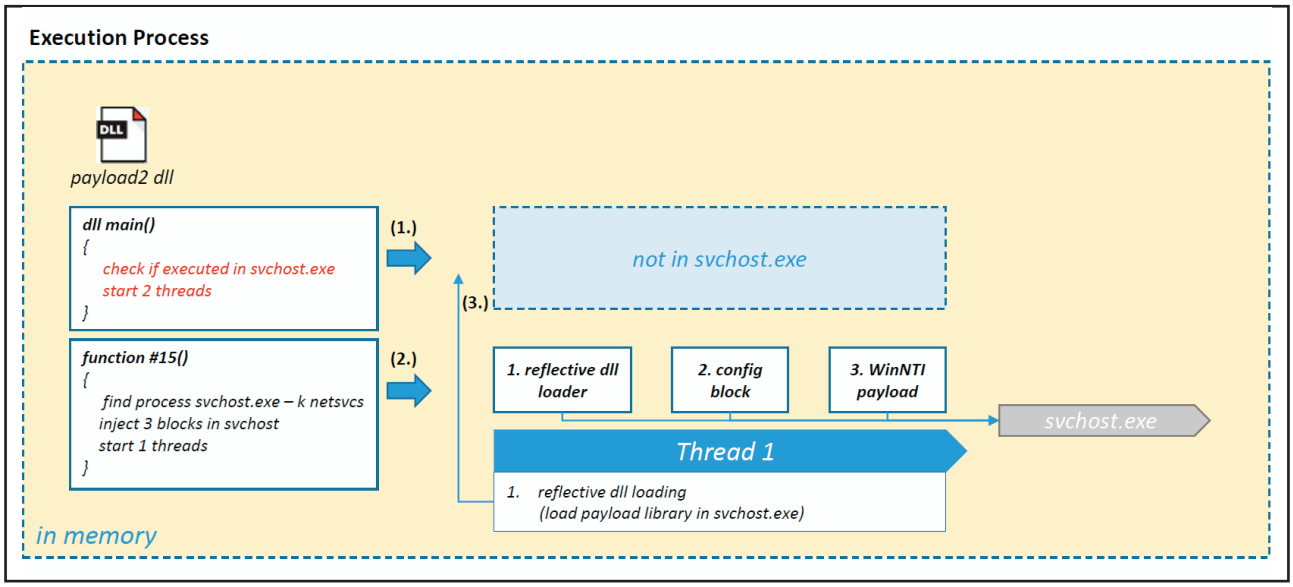


Abbildung 3: Ausführungsprozess der Malware - Teil I

Der Ausführungsprozess der Malware sieht wie folgt aus (siehe Abb. 3):

(1.) Beim erstmaligen Laden der ***WinNTI payload-dll*** (siehe Abbildung 3 ***payload 2 dll***) im Arbeitsspeicher wird zuerst die DllMain()-Funktion ausgeführt, die prüft, ob die DLL in einem Prozess mit dem Modulnamen ***svchost.exe*** ausgeführt wird. Zu diesem Zeitpunkt ist das nicht der Fall.

(2.) Bei der nachfolgenden Ausführung der Funktion mit dem Ordinal 15 erfolgt zunächst die Suche nach dem svchost.exe-Prozess, welcher mit dem Parameter –k netsvcs gestartet wurde, dann werden in diesen drei Blöcke mittels VirtualAllocEx injiziert:

1. einen reflective dll loading shellcode
2. einen config block
3. die WinNTI payload dll (umfasst u.a. Remote Access Tool – RAT)

Im Anschluss wird mittels CreateRemoteThread der reflective dll loading Shellcode im Zielprozess svchost.exe zur Ausführung gebracht.

(3.) Bei der diesmaligen Ausführung des WinNTI Payload im svchost.exe-Prozess wird erneut die DllMain()-Funktion aufgerufen.

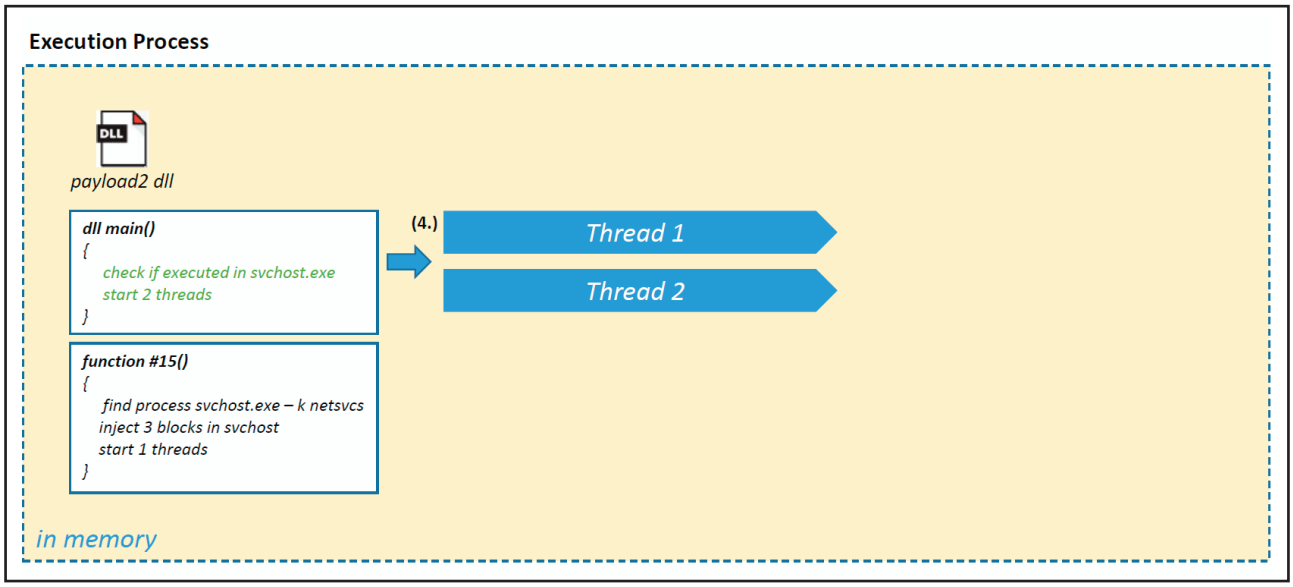


Abbildung 4: Ausführungsprozess der Malware - Teil II

(4.) Die erneute Prüfung, ob die WinNTI ***payload-dll*** im svchost.exe-Prozess ausgeführt wurde, kommt nun zu einem positiven Ergebnis und startet daraufhin zwei Haupt-Threads.

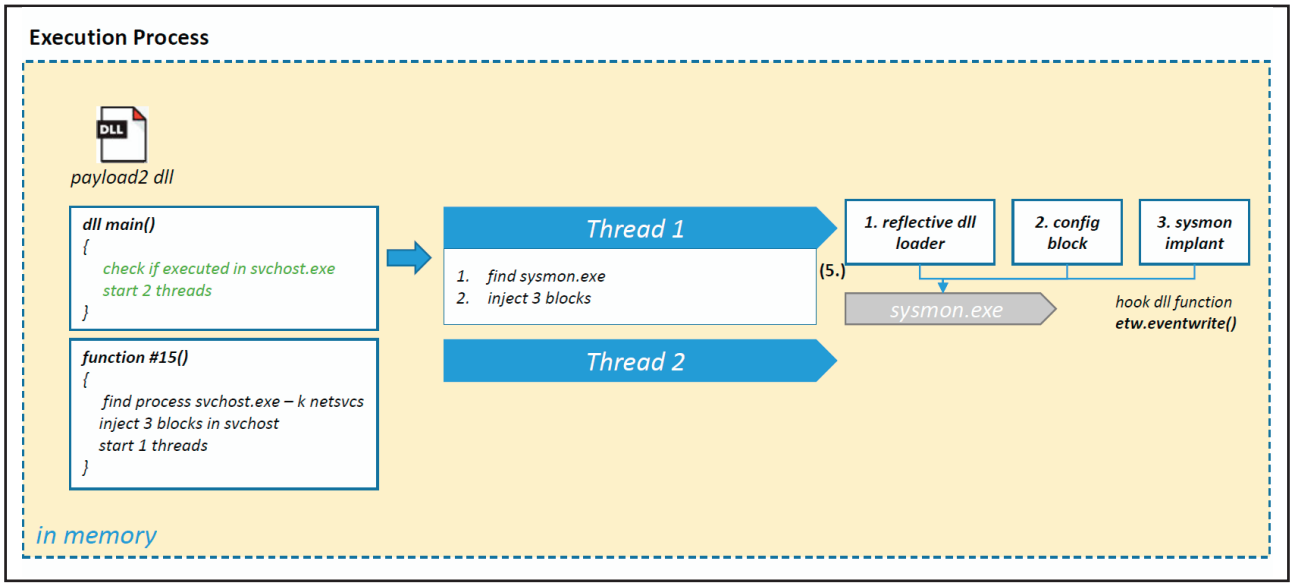


Abbildung 5: Ausführungsprozess der Malware - Teil III

(5.) Der erste Haupt-Thread sucht nach dem sysmon.exe-Prozess (Windows System Monitor) und – falls vorhanden – injiziert drei Blöcke in den Prozess:

1. einen reflective dll loader
2. einen config block
3. das sysmon implant der Malware

Das ***sysmont implant***, das mit Hilfe des reflective dll loaders geladen wird, führt ein API-Hooking der Funktion Advapi32.dll!EventWrite durch (siehe Abbildung 6). Die Realisierung basiert wahrscheinlich auf einer Detours-Implementierung einer Drittbibliothek.

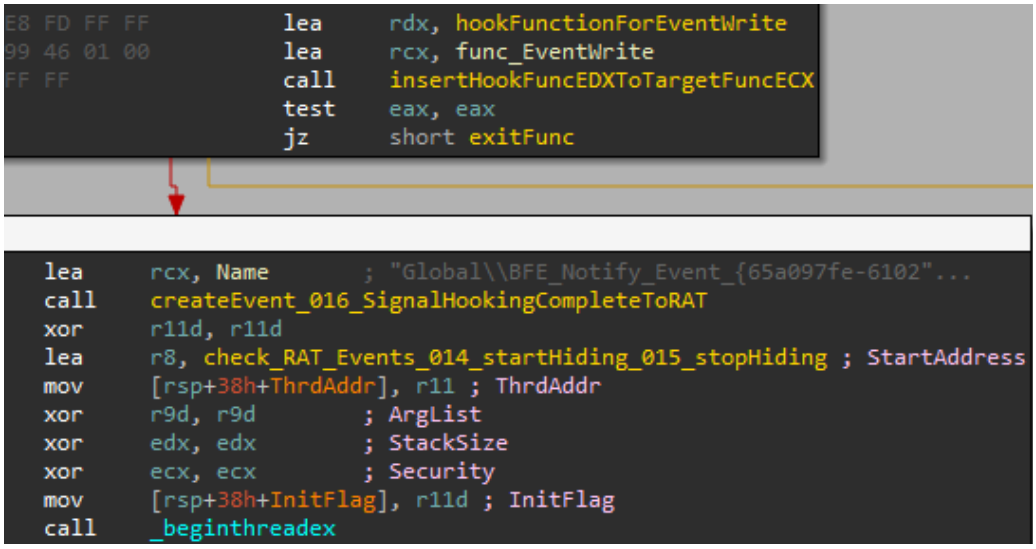


Abbildung 6: API-Hooking der Funktion Advapi32.dll!EventWrite

Letztendlich wird auch der Funktionsprolog von ntdll.dll!EtwEventwrite mit einem Sprungbefehl überschrieben. Die neu eingefügte Funktion kann, abhängig vom Zustand einer globalen Variable, bestimmte Malware-Funktionalitäten aus dem Event Tracing von Sysmon ausblenden. Das erfolgreiche API-Hooking wird dem RAT mittels der Erzeugung des Events **Global\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411016}** signalisiert (Erkennungsmerkmal für einen kompromittierten Sysmon-Prozess). Die Events **Global\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411014}** und **Global\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411015}** dienen dem RAT zur An- bzw. Ausschaltung der Eventfilterung. Sie werden jeweils nur kurzzeitig verwendet und dienen eher nicht als verlässliches Merkmal einer Infektion.

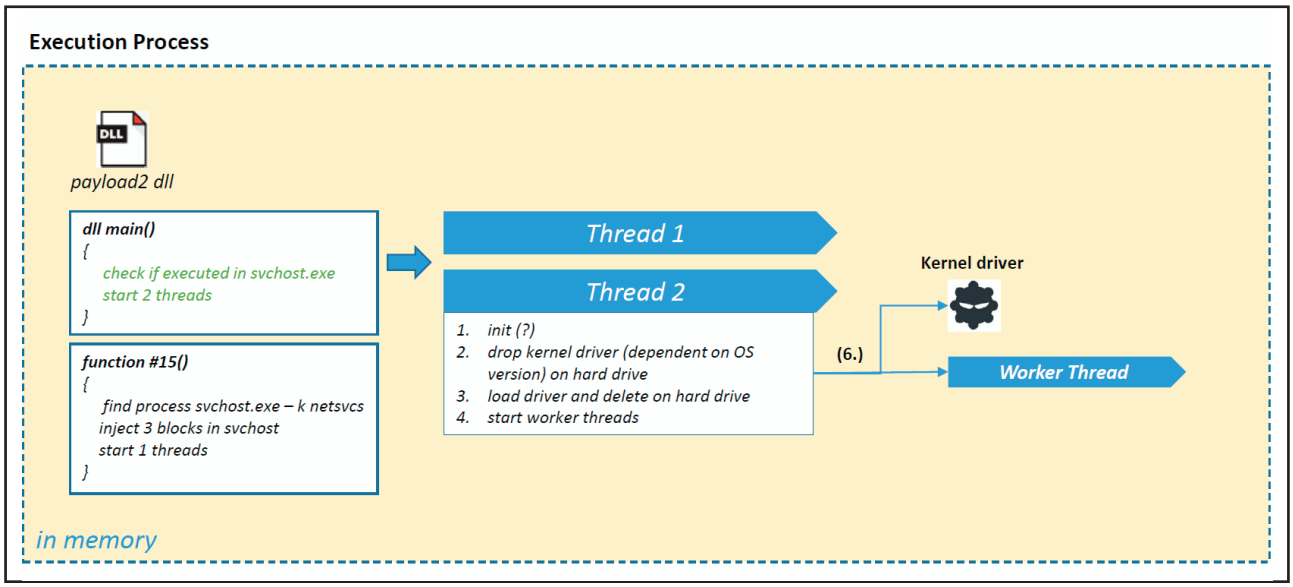


Abbildung 7: Ausführungsprozess der Malware - Teil IV

(6.) Der zweite Haupt-Thread, der gestartet wurde, erzeugt zunächst das Event **Global\BFE_Notify_Event_{7D00FA3C-FBDC-4A8D-AEEB-3F55A4890D2A}** (Erkennungsmerkmal für ein laufendes WinNTI RAT). Mutmaßlich kann das RAT über dieses Event bereits bestehende, vorherige Infektionen desselben Systems über eine erneute Infektion informieren (ggf. Update/CleanUp).

(7.) Im Anschluss schreibt das RAT – je nach Betriebssystem-Version – einen Kernel Treiber der Malware auf die Festplatte, führt diesen aus und löscht ihn wieder von der Festplatte. Zudem werden weitere User Mode Worker Threads gestartet.

Der Kernel Treiber der Malware hat die Aufgabe, CPIP Network Device Interface Specification (NDIS) Protocol Handler zu hooken und damit eingehende TCP Pakete abzufangen. Diese werden in ein Null-Device (\\Device\\Null oder \\.\Nul) geschrieben, wo sie vom Worker Thread ausgelesen und durch die Malware verarbeitet werden können. Es kommen u.a. die IoControlCodes 0x15E007 und 0x156003 zum Einsatz (custom functions 0x800, 0x801). Dadurch sollen verdeckte Kommunikationskanäle für die Malware etabliert werden, die durch andere Komponenten des Betriebssystems nicht registriert werden.

Mögliche Erhöhung der Sichtbarkeit einer Infektion

Die Unterdrückung der Protokollierung von Windows Events mittels Sysmon Implant greift in der oben beschriebenen Malware-Variante nur beim Einsatz der 32-Bit Version von Sysmon. Die Umbenennung der Sysmon-Executable kann dessen Kompromittierung bereits aushebeln. Alternativ kann das Event **Global\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411015}** manuell erzeugt werden, um die Eventfilterung durch das Sysmon Implant zu unterbinden. Dies kann ggf. hilfreich sein, um den Ausbreitungsgrad und Aktivität einer etwaig vorhandenen Infektion mit dieser WinNTI-Variante zu erforschen (Scoping & Containment).

Darüber hinaus kann es empfehlenswert sein, Hauptspeicherabbilder (memory dumps) von als besonders gefährdet eingestuften Servern mit Hilfe von Werkzeugen der Hauptspeicherforensik (z.B. volatility) auf das Vorhandensein von WinNTI Kerneltreibern (Plugins Modules und Moddump) und den Hauptevents des RAT (Plugin Handles) zu testen. Entsprechende Befehlsbeispiele und weitere Detektionsregeln können den Anhängen entnommen werden.

Handlungsempfehlung

Die Cyberabwehr des Bundesamtes für Verfassungsschutz empfiehlt die Prüfung des eigenen Unternehmens- oder Organisationsnetzwerks mit den nachfolgenden Detektionsregeln.

Darüber hinaus bieten wir Ihnen – insbesondere beim Anschlagen der beigefügten Detektionsregeln – zusätzliche Hintergrundinformationen an. Hierzu stehen wir Ihnen unter folgenden Kontaktdaten gerne zur Verfügung:

Tel.: 0221-792-2600 oder

E-Mail: poststelle@bfv.bund.de

Cyberabwehr BfV

Abbildung nach MITRE ATT&CK[3] Schema

No.	Tactics	Techniques
T1133	Initial Access	External Remote Services
T1059	Execution	Command-Line Interface
T1106	Execution	Execution through API
T1085	Execution	Rundll32
T1035	Execution	Service Execution
T1204	Execution	User Execution
T1038	Persistence	DLL Search Order Hijacking
T1215	Persistence	Kernel Modules and Extensions
T1205	Persistence	Port Knocking
T1108	Persistence	Redundant Access
T1009	Defense Evasion	Binary Padding
T1116	Defense Evasion	Code Signing
T1140	Defense Evasion	Deobfuscate/Decode Files or Information
T1089	Defense Evasion	Disabling Security Tools
T1038	Defense Evasion	DLL Search Order Hijacking
T1480	Defense Evasion	Execution Guardrails
T1107	Defense Evasion	File Deletion
T1112	Defense Evasion	Modify Registry
T1205	Defense Evasion	Port Knocking
T1055	Defense Evasion	Process Injection
T1108	Defense Evasion	Redundant Access
T1014	Defense Evasion	Rootkit
T1085	Defense Evasion	Rundll32
T1045	Defense Evasion	Software Packing
T1099	Defense Evasion	Timestomp
T1043	Command and Control	Commonly Used Port
T1090	Command and Control	Connection Proxy
T1094	Command and Control	Custom Command and Control Protocol
T1024	Command and Control	Custom Cryptographic Protocol
T1008	Command and Control	Fallback Channels
T1205	Command and Control	Port Knocking
T1219	Command and Control	Remote Access Tools

3 URL: https://attack.mitre.org

Detektionsregeln

Yara Rules (especially for memory scanning)

```
rule cb2_01
{
  strings:
    $e1 = „Global\\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411015}” ascii nocase
    $e2 = „Global\\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411014}” ascii nocase
    $e3 = „Global\\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411016}” ascii nocase
    $e4 = „\\BaseNamedObjects\\{B2B87CCA-66BC-4C24-89B2-C23C9EAC2A66}” wide
    $e5 = „BFE_Notify_Event_{7D00FA3C-FBDC-4A8D-AEEB-3F55A4890D2A}” nocase
  condition:
    (any of ($e*))
}
rule cb2_02
{
  strings:
    $a1 = „IPSecMiniPort” wide fullword
    $a2 = „ndis6fw” wide fullword
    $a3 = „TCPIP” wide fullword
    $a4 = „NDIS.SYS” ascii fullword
    $a5 = „ntoskrnl.exe” ascii fullword
    $a6 = „\\BaseNamedObjects\\{B2B87CCA-66BC-4C24-89B2-C23C9EAC2A66}” wide
    $a7 = „\\Device\\Null” wide
    $a8 = „\\Device” wide
    $a9 = „\\Driver” wide
    $b1 = { 66 81 7? ?? 70 17 }
    $b2 = { 81 7? ?? 07 E0 15 00 }
    $b3 = { 8B 46 18 3D 03 60 15 00 }
  condition:
    (6 of ($a*)) and (2 of ($b*))
}
rule cb2_03
{
  strings:
    $b1 = { 0F B7 ?? 16 [0-1] (81 E? | 25) 00 20 [0-2] [8] 8B ?? 50 41 B9 40 00 00 00 41 B8 00
10 00 00 }
    $b2 = { 8B 40 28 [5-8] 48 03 C8 48 8B C1 [5-8] 48 89 41 28 }
    $b3 = { 48 6B ?? 28 [5-8] 8B ?? ?? 10 [5-8] 48 6B ?? 28 [5-8] 8B ?? ?? 14 }
    $b4 = { 83 B? 90 00 00 00 00 0F 84 [9-12] 83 B? 94 00 00 00 00 0F 84 }
```

```
    $b5 = { (45 | 4D) (31 | 33) C0 BA 01 00 00 00 [10-16] FF 5? 28 [0-1] (84 | 85) C0 }
  condition:
    (4 of ($b*))
}
rule cb2_04
{
  strings:
    $b1 = { 4C 8D 41 24 33 D2 B9 03 00 1F 00 FF 9? F8 00 00 00 48 85 C0 74 }
    $b2 = { 4C 8B 4? 08 BA 01 00 00 00 49 8B C? FF D0 85 C0 [2-6] C7 4? 1C 01 00 00 00 B8 01
00 00 00 }
    $b3 = { 8B 4B E4 8B 53 EC 41 B8 00 40 00 00 4? 0B C? FF 9? B8 00 00 00 EB }
  condition:
    (2 of ($b*))
}
rule cb2_05
{
  strings:
    $a1 = „-k netsvcs” ascii
    $a2 = „svchost.exe” ascii fullword
    $a3 = „%SystemRoot%\\System32\\ntoskrnl.exe” ascii
    $a4 = „Global\\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411015}” ascii
    $a5 = „Global\\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411014}” ascii
    $a6 = „Global\\BFE_Notify_Event_{65a097fe-6102-446a-9f9c-55dfc3f411016}” ascii
    $a7 = „cmd.exe” wide
    $a8 = „,XML” wide
    $a9 = „\\rundll32.exe” wide
    $a10 = „\\conhost.exe” wide
    $a11 = „\\cmd.exe” wide
    $a12 = „NtQueryInformationProcess” ascii
    $a13 = „Detours!” ascii fullword
    $a14 = „Loading modified build of detours library designed for MPC-HC player
(http://sourceforge.net/projects/mpc-hc/)” ascii
    $a15 = „CONOUT$” wide fullword
    $a16 = { C6 0? E9 4? 8? 4? 05 [2] 89 4? 01 }
  condition:
    (12 of ($a*))
}
```

Registry Keys

```
HKLM\SOFTWRE\Microsoft\Ole\
        lpValueName: GUID (create)
```

lpData: XXXXX-XXXXX-XXXXX-XXXXX-XXXXX[4] (read/write)

HKLM\System\CurrentControlSet\Services\tmpXXXX[5]

lpValueName: Type

lpData: 0x1 (write)

lpValueName: ErrorControl

lpData: 0x1 (write)

lpValueName: Start

lpData: 0x3 (write)

Detection with Volatility

Plugin Handles

Examples: Search memdump for major RAT and MD5-PID events

vol.py -f $img –profile=$prof handles -t Event | grep –i "7D00FA3C-FBDC"

vol.py -f $img –profile=$prof handles -t Event | grep -Pe „\{[0-9a-zA-Z]{32}-[0-9]{1,}\}"

Offset(V)	Pid	Handle	Access	Type	Details
0xffffe00XXXXXXXXX	XXX	0xXXXX	0x1f0003	Event	BFE_Notify_Event_{7D00FA3C-FBDC-4A8D-AEEB-3F55A4890D2A}
0xfffffa800XXXXXXX	XXX	0xd64	0x1f0003	Event	{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX-XXX}

Plugin Modules

Offset(V)	Name	Base	Size	File
0xffffe00XXXXXXXXX	tmpXXXX[6].tmp	0xfffff80XXXXXXXXX	0xe000	\??\C:\Windows\TEMP\tmpXXXX.tmp

WinNTI Kernel Driver Type 2a (Example 1)

0xffffe00XXXXXXXXX	tmpXXXX[6].tmp	0xfffff80XXXXXXXXX	0xe000	\??\C:\Windows\TEMP\tmpXXXX.tmp

WinNTI Kernel Driver Type 2b (Example 2)

0xfffffa800XXXXXXX	NtXXXX[7].tmp	0xfffff8800XXXXXXX	0xd000	\??\C:\Windows\TEMP\NtXXXX.tmp

Plugin Malfind

WinNTI RAT DLL, reflectively loaded inside svchost.exe –k netsvcs (obfuscated PE header)

Process: svchost.exe Pid: XXX Address: 0x180000000

4 Derived at runtime from MAC-Address of first network adapter, a random number (1-5 digits) and the hostname (GetComputerNameA)
5 Four random digits, conforming to random filename of dropped kernel driver. Registry keys are deleted after call to ntdll.dll!NtLoadDriver
6 Four random digits based on a call to kernel32.dll!GetTempFileNameA
7 Four random alphanumerical chars which likely base on a call to kernel32.dll!GetTempFileNameA

```
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x180000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x180000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x180000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x180000030  00 00 00 00 00 00 00 00 00 00 00 00 e8 00 00 00   ................
```

Injected WinNTI RAT config structure (Example 1)

```
Process: svchost.exe Pid: XXX Address: 0x50ba020000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x50ba020000  00 00 43 c0 50 00 00 00[8] 00 00 00 00 00 00 00 00   ..C.P...........
0x50ba020010  41 64 76 61 70 69 33 32 2e 64 6c 6c 00 00 00 00   Advapi32.dll....
0x50ba020020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x50ba020030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

Injected WinNTI RAT config structure (Example 2)

```
Process: svchost.exe Pid: XXX Address: 0xb70000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00b70000  00 00 82 02 00 00 00 00[9] 00 00 00 00 00 00 00 00   ................
0x00b70010  41 64 76 61 70 69 33 32 2e 64 6c 6c 00 00 00 00   Advapi32.dll....
0x00b70020  00 00 00 00 47 6c 6f 62 61 6c 5c 7b 46 43 43 33   ....Global\{FCC3
0x00b70030  30 45 34 30 45 35 41 43 44 35 41 36 44 35 31 42   0E40E5ACD5A6D51B
```

Injected WinNTI Reflective DLL Loading Shellcode Type 2a

```
Process: svchost.exe Pid: XXX Address: 0x50ba4a0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x50ba4a0000  48 89 5c 24 18 55 56 57 41 54 41 55 41 56 41 57   H.\$.UVWATAUAVAW
0x50ba4a0010  48 8d 6c 24 d9 48 81 ec 90 00 00 00 45 33 f6 48   H.l$.H......E3.H
0x50ba4a0020  8b f9 48 85 c9 75 07 33 c0 e9 9d 05 00 00 c7 45   ..H..u.3.......E
0x50ba4a0030  fb 40 00 00 00 41 bf 01 00 00 00 44 38 71 24 0f   .@...A.....D8q$.
```

8 Virtual address (VA) of injected WinNTI Payload DLL
9 see Fn. 6

Injected WinNTI Reflective DLL Loading Shellcode Type 2b

```
Process: svchost.exe Pid: XXX Address: 0xc00000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00c00000  40 56 48 81 ec a0 00 00 00 48 8b f1 48 85 c9 75   @VH......H..H..u
0x00c00010  0b 33 c0 48 81 c4 a0 00 00 00 5e c3 48 89 9c 24   .3.H......^.H..$
0x00c00020  b8 00 00 00 4c 89 bc 24 80 00 00 00 45 33 ff c7   ....L..$....E3..
0x00c00030  44 24 54 40 00 00 00 44 38 79 24 0f 84 a5 00 00   D$T@...D8y$.....
```

Injected WinNTI Payload DLL (obfuscated PE header)

```
Process: svchost.exe Pid: XXX Address: 0x50c0430000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x50c0430000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x50c0430010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x50c0430020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x50c0430030  00 00 00 00 00 00 00 00 00 00 00 00 e8[10] 00 00 00    ................
```

Intermediate Loader

```
Process: vmtoolsd.exe Pid: XXX Address: 0x9ae30e0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x9ae30e0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x9ae30e0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x9ae30e0020  00 00 00 00 00 00 00 00 14 21 0e e3 9a 00 00 00   .........!......
0x9ae30e0030  00 00 00 00 00 00 00 00 00 00 00 00 e0[11] 00 00 00    ................
```

Hash Values of Malicious Files / Code Blocks

Intermediate Loader (disk)

```
Type: PE32+ executable (DLL) (console) x86-64, for MS Windows
ssdeep (file): 3072:3ZvhT4Xd7ncWKby0T+SQ0IYevsxtjg9RfnJHarO:3LT4tVKO0wLsxt0TnJHaO
ssdeep (.text): 768:zRWRzPlgivs6/lR/T4XxMJefllEHWcVDkPKbgB:S2ivhT4Xd7EWchkPKby
ssdeep (.data): 1536:5Q0PgGT9YX/sLPdK0skw7KjgrrdqsE7ynJHarO7:5Q0IYevsxtjg9RfnJHarO
imphash: 1fb46361b3762772e68127b42d1b1d5e
```

10 e_lfanew, Pointer to PE-Header
11 e_lfanew

Reflective DLL Loading Shellcode Type 1 (used by Intermediate Loader and Loader, disk and memory)

Codesize: 4.410 Bytes

MD5: 25c735f0e64464e8c75db3d225912add

SHA1: ec8fd561551db21c86766296611c1d8df9bf98c5

ssdeep:
48:sKuCvM5L7NuPFi6YaLC8DNx+xlWEsOQGSmY0X1BHT5Hp5iwjS9d6ybxnAOmq/a7D:srCvk3NuH7LC4qlWST1B8Ma427
a7D

Intermediate Loader Payload DLL (memory only)

ssdeep (file): 1536:B6Lf7rVA8vhTjRmIeYQv9jB0dMSI/qe9lD9:QLfrvhTjRNeYA9ieSbGlD9

ssdeep (.text):
768:466RwzXvMmLrVAhu5ljDhTb/YWD8ChD/1gIeYQhtbpY8B0z5MSuN/:46Lf7rVA8vhTjRmIeYQv9jB0dMSI/

WinNTI Payload DLL (Decrypted PE, unloaded/injected state, memory only)

ssdeep: 12288:iUCXzbtTwr9ZnO7CMXvXD03WvR+WZj1EusOLw4owntX4SncgcP:ODbtTOnO7CMX7WeIWZgO7owtIScj

Reflective DLL Loading Shellcode Type 2a (loads injected WinNTI Payload DLL, calls DllMain, memory only)

Codesize: 2.048 Bytes

MD5: 119d144147662013ee85e8ee00024cc4

SHA1: 715a1b53556be0f51951547b86ec8d38a74ec7d9

SHA256: bd1cde125389590f75b808a27401de15b03f70795311881c5da3e079a44e39ef

ssdeep: 48:FyaxW8RrvmX2EJtzXFurCXgj9e0tQ380Fon/keb5B7003/s:tepfzFiCwj9eVM0IkebX0Es

Reflective DLL Loading Shellcode Type 2b (loads injected WinNTI Payload DLL, calls DllMain, memory only)

MD5: 42560fde33e1e5f83e61bcdfa77b5b9c

SHA1: 29fee2e1138592a3c3167176849dee3f193bf4a8

SHA256: 5aa25bb6795f0e72176b6d7b5f9808c8c4685ce4ca1ab34e0ce4e41eaf19ad61

ssdeep:
48:/D7DxQaGZDz5b546czuXZUa0Gr2z44uLGswLBaZalxIJegXGplDYriXhwaul:3DxPGZTMzOmnG6zqLGsYBaMlCJegW3
YD

Sysmon Implant (Decrypted PE, memory only)

Codesize: 90.112 Bytes

MD5: 3bb87749da36ebd1a564ee85e9f0fff0

SHA1: 8a2356303356e2850a15401ee8b5727b152e200b

SHA256: 806df629a0e58a70b4936bb9a28eafe555ff4ce190039bb26215782a93cff4cb

ssdeep:
1536:vGzAkyjIOsTCT2IP+W0k+0X4a3Ro1MeAJhN9tdN9VtdNz9Tl1caSQZ/26XvX:vGzAkyE3TCqk+pIgMeAJhN9tdN9V
tdNn

imphash: f3c01ba3a71e1e0ef157c3b8cb0ad625

MD5 (code section): 5e5dd13d6986f521c24e816f3a0880cc

SHA1 (code section): 9a3ca3a368fee2f2f9d824e6d8ffd1ef2ed62c72

SHA256 (code section): 28afc1eb9d37322257c9ee628b82ca1e44af29e2e40f28d70ee544a63113638f

ssdeep (code section):
768:TiDxzGr+GAJxxtgyZiCcJ5Ev7AT5sFlloZ8RBT2I/HqhPO0i1+i5X4aFV/O3wds:6GzAkyjIOsTCT2IP+W0k+0X4a3
RolMe

Kernel Driver Type 1 (temporarily dropped to disk, deleted after loading)

Filesize: 47.104 Bytes

MD5: 1801319eb2b82016ae6a33ee18fcc3ad

SHA1: 7bbed9fbff45b15dbf5cedfa3636a3caad65650f

SHA256: ebdb8cfc3207b411a4d898489c8825cb2187221a473f2fbf7a43cbf637f2fe57

ssdeep: 768:jZh+oyCeGqt/P76bbwYCmKGqV+VNQNDBKTW1/bz2vTvQtCK:jiCeB/Gbbi0qV6QNBK+QTvQQK

imphash: c22f9228e1c400cb179800b69544162b

Kernel Driver Type 2a (temporarily dropped to disk, deleted after loading, Example 1)

Filesize: 44.624 Bytes

MD5: 50f624b3fb6ca04f352e0463a43df86f

SHA1: 3c404486a5c443e43c1b7691de7801cece44a733

SHA256: 3c25dcb33e018c21a3dc709c54495c0e504aeee78d7f103deaf19c1d802d57da

ssdeep: 768:pQIbhJi7OB1/HzktBgWb8oiICMvahoICS4AIHOyMKIoAj:pQIDRBW4o8+ICS4AltoA

imphash: fcccb379816ade76b537359d17969ca4

MD5 (.text): 1e615b812bd1b6c205e27c4c5067fd8a

SHA1 (.text): 26d6f5c9a779dba2104fedb90d00bc1ff0aa5195

SHA256 (.text): 8cfa0f9caec35a80078db887a7cf80a4e903abdb010b3045ef6f54724ba0c4d2

ssdeep (.text): 384:BQIbhd3i7OGK10mXEGHzktMgM+mJ/RWb8oirUj0HM:BQIbhJi7OB1/HzktBgWb8oiICM

Kernel Driver Type 2b (temporarily dropped to disk, deleted after loading, Example 2)

Filesize: 34.816 Bytes (dumped from memory with moddump)

MD5: b96dfbc749b99bc672c74708373bbc97 (dumped with moddump)

SHA1: 4e45d9b0bc282cc93113c7ba51b1b4ac173a208d (dumped with moddump)

SHA256: 5af2edd199b6c4ea731449b202ea96faef6c11d1ac0ca7b22aa9023e0186621f (dumped with moddump)

ssdeep: 768:Zhf9ozikYw7rhcCMsahoICS4AIvm7tSw5iZ:W1Yw7rH7ICS4ANtSw5M (dumped with moddump)

MD5 (.text): ace45cab5b340beed180fce546f16bd6

SHA1 (.text): d058fcef882a6bfa993cefb2371f1eb5d187e356

ssdeep (.text): 384:Hh/HusRuVIL7ozi1B82zfR27rhp0p0HM:Hhf9ozikYw7rhcCM

Config Block Structure (Size 0x140)

Offset 0x0	VA of injected Block 3 (WinNTI Payload DLL)
Offset 0x8	(Optional) VA of injected Second Payload
Offset 0x10	String „Advapi32.dll“
Offset 0x24	(Optional) EventName „MD5-PID“

```
Offset 0x88           kernel32_VirtualAlloc
Offset 0x90           kernel32_HeapAlloc
Offset 0x98           kernel32_GetProcessHeap
Offset 0xA0           ntdll_memcpy
Offset 0xA8           ntdll_memset
Offset 0xB0           kernel32_IsBadReadPtr
Offset 0xB8           kernel32_VirtualFree
Offset 0xC0           kernel32_LoadLibraryA
Offset 0xC8           kernel32_GetProcAddress
Offset 0xD0           kernel32_VirtualProtect
Offset 0xD8           kernel32_CreateEventA
Offset 0xE0           advapi32_InitializeSecurityDescriptor
Offset 0xE8           advapi32_SetSecurityDescriptorDacl
Offset 0xF0           kernel32_OutputDebugStringA
Offset 0xF8           kernel32_OpenEventA
Offset 0x100   kernel32_CloseHandle
Offset 0x108   kernel32_GetLastError
Offset 0x110   ntdll_itoa
Offsets up to 0x13F   temporary data
```

Network

Possible C2 DNS Domain Name

```
*.dick.mooo.com
```

Possible C2 HTTP header

```
GET [Offset 0x10C in „config”] HTTP/1.1\r\n
Cookie: SN= [bin2hex(data_to_send)]
Accept: text/html, */*
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) Chrome/53.0.2785.148
Host: [Offset 0x8 in „config”]
Connection: Keep-Alive
```

Zusätzliche IOCs bereitgestellt durch die DCSO

variante-A/system.dat-output/driver1.sys	sha1:8b966bc4c4adde90f51f68a78aa326b761981fb4
variante-A/system.dat-output/payload.dll	sha1:611b4c014d4a29b632c167a613b677c08d206d1e
variante-A/system.dat-output/payload	sha1:fd04c0168b844d17828ee03a1e5249e7986ce9ba
variante-A/system.dat	sha1:5e00d36388ce0fe4bbd0624d674f2f007f7e500a
variante-B/TmsmHttp64.dll-output/driver1.sys	sha1:8b966bc4c4adde90f51f68a78aa326b761981fb4
variante-B/TmsmHttp64.dll-output/driver2.sys	sha1:003b5d82a9e208e0bc2f339d46bb907cbf588bc1
variante-B/TmsmHttp64.dll-output/payload.dll	sha1:a224a276213eaecc91f0b36a66809b9cb2e7b244
variante-B/TmsmHttp64.dll-output/payload	sha1:a2dd0e1f27fcaa51f42a7f5d4f2d50d8f4500bd9
variante-B/TmsmHttp64.dll	sha1:2da100999d323c0628df4878409269ac8f131cee
variante-C/iiscfg64.dll-output/driver1.sys	sha1:8b966bc4c4adde90f51f68a78aa326b761981fb4
variante-C/iiscfg64.dll-output/driver2.sys	sha1:3bb1daf9c5b39a026af5fd5a6c321cd3d0be04d6
variante-C/iiscfg64.dll-output/payload.dll	sha1:76bd5e3261609041f29bb429bc1741303e61f328
variante-C/iiscfg64.dll-output/payload	sha1:b0cfca2501096b914b0aedd35403d4505729c90c
variante-C/iiscfg64.dll	sha1:61032695b15bfcd1fbeceb015b16cea21bfaa791
variante-C/instapi64.dll-output/driver1.sys	sha1:857197c37751dcbc10a89fa962d60e428952ce93
variante-C/instapi64.dll-output/driver2.sys	sha1:dbe2e361989dd3e7d7c9e3c6aed69f2237c9aa02
variante-C/instapi64.dll-output/payload.dll	sha1:b8d35d436888b2f6d4ff2a958d48ca1df17e799e
variante-C/instapi64.dll-output/payload	sha1:e01c7793450e8b140fa13f88901fe041ea34be38
variante-C/instapi64.dll	sha1:8821beab255d943185c114c58f1996b40d5e1368
variante-CR/payload-output/driver1.sys	sha1:74cace25311ac0abead7bd94e039ef080e550328
variante-CR/payload-output/driver2.sys	sha1:c539ca5aa16de324551c913b61d22652e66de93f
variante-CR/payload-output/payload.dll	sha1:595392a8c3eb723bdca1885db2598fea1fa2b516
variante-CR/payload	sha1:48f2da6aeaef0cc342ea4bf9ff20aa8bfcde9872
variante-CRS/payload-output/driver1.sys	sha1:74cace25311ac0abead7bd94e039ef080e550328
variante-CRS/payload-output/driver2.sys	sha1:174101153536112422c594f6c3038aa47f3fd14e
variante-CRS/payload-output/payload.dll	sha1:3c8edeadaeb644341402d99ca8a0629368cb0125
variante-CRS/payload	sha1:7cfe9d75b3f7bb31a6d0c86da7a43f4bb9bdc7bd
variante-D/tsmgetst.dll-output/driver1.sys	sha1:2b319b44451abb0596b9187e06f1fb7b4ace969d
variante-D/tsmgetst.dll-output/driver2.sys	sha1:30d1dd1dd4f0ace7a4f2c24e31fb6a0ee33e8a3a
variante-D/tsmgetst.dll-output/dsefix.exe	sha1:2bc358ddc72f59ba0373b8635ab08ad747c12180
variante-D/tsmgetst.dll-output/payload.dll	sha1:df7732ce1a393c59889ae61321e7da3d3f1a1980
variante-D/tsmgetst.dll-output/payload	sha1:aaa6eeaf422b5a8451121513c66c6bd7cb3b9da3
variante-D/tsmgetst.dll	sha1:ffce6895a5bcade8631676ac67c1f919505d4f19
variante-E/sigc-2.4.dll-output/decrypted_strings.txt	sha1:3b1f3ed2eeb746733b3c2bb483a481ce2d7f7cf1
variante-E/sigc-2.4.dll-output/driver1.sys	sha1:98c32b4093ed1d7cba6fdcd7667f7ba10ba7a94c
variante-E/sigc-2.4.dll-output/driver2.sys	sha1:ca00eafde42f1456de01140556d8c3002866cc74
variante-E/sigc-2.4.dll-output/payload.dll	sha1:54f7d7c145bbae0979ad0b42689a9008ab3d3883
variante-E/sigc-2.4.dll-output/payload	sha1:10ceb3bd963708895c394303651dde0da315490e
variante-E/sigc-2.4.dll-output/ShutDownEvent.dll	sha1:11d6619900369643ebe6c0bbf6a28178cfa620bd
variante-E/sigc-2.4.dll-output/ShutDownEvent	sha1:3efae65475cb1f6a34e11e012c53dac0412674d4
variante-E/sigc-2.4.dll-output/start_function.bin	sha1:ee2a177f2e2ae8679b28caa8aba222d3fd80cdbb
variante-E/sigc-2.4.dll-output/sysmon-implant.dll	sha1:045e728362773c358b07e416d3cd3e66af71549c
variante-E/sigc-2.4.dll-output/sysmon-implant	sha1:b3f04f4e41afe17117204e0b48162886b58932ce
variante-E/sigc-2.4.dll	sha1:c11675257b9927cabd6e5e259021070a95266566
variante-F/glmf-2.0.dll-output/decrypted_strings.txt	sha1:08a4fa8b98d2c7efcfcc7710586e498c34be6b3f
variante-F/glmf-2.0.dll-output/driver1.sys	sha1:894c71f4fb27aa0285797a2735b23c0aecd81d74
variante-F/glmf-2.0.dll-output/driver2.sys	sha1:1994fdc0a26198e84c9e15ae071e3f759f85cfd0
variante-F/glmf-2.0.dll-output/payload.dll	sha1:550ceb58c15537c991ddf772200a888c0823eb06
variante-F/glmf-2.0.dll-output/payload	sha1:48bc1d610f3f9219ad9f47f44368c2ef2eb4d64c
variante-F/glmf-2.0.dll-output/start_function.bin	sha1:263ca823e42eea1f062bf375a4204f01aa883ad1
variante-F/glmf-2.0.dll-output/sysmon-implant.dll	sha1:045e728362773c358b07e416d3cd3e66af71549c
variante-F/glmf-2.0.dll-output/sysmon-implant	sha1:b3f04f4e41afe17117204e0b48162886b58932ce
variante-F/glmf-2.0.dll	sha1:39d8e4abc92ba068e30597cad0d195af4fe8372b