Scriptie ingediend tot het behalen van de graad van
PROFESSIONELE BACHELOR IN DE ELEKTRONICA-ICT

Creation of Covert Command & Control Channel Frameworks

Tom Kustermans

academiejaar 2015-2016

AP Hogeschool Antwerpen
Wetenschap &Techniek
Elektronica-ICT

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

Table of Contents

Abstract

EY biedt verchillende services aan, waaronder: Assurance, Tax, Advisory & Transaction. Advisory Services beschikt over een onderdeel ITRA- *Information Technology Risk and Assurance*, dat als doel heeft klanten bij te staan bij IT-gerelateerde risico's. Een dienst dat het ITRA-team bieden is de *redteam-test*, dit is een test waarbij het ITRA-team een bedrijfsnetwerk dient te testen door het netwerk aan te vallen vanuit het perspectief van een *hacker*. Op die wijze krijgen zij een goed overzicht over de *IT-security* van het bedrijfsnetwerk, waaronder het voorkomen, analyseren en reageren van het *IT-security* team van dat bedrijf op een potentiële *cyber attack*.

Het doel van de stageopdracht was om een *covert communication channel framework*, ofwel geheim communicatie kanaal, aan te bieden waarmee het redteam-testing team van buitenaf toegang krijgt tot het besturen van een computer binnen een beveiligd netwerk, zonder dat deze communicatie ontdekt en/of geblokkeerd kan worden.

Er zijn reeds een aantal gekende methoden om het bovenstaande te verwezenlijken. De meest gekende methode is HTTP-tunneling. Deze zorgt ervoor dat verschillende protocollen, die op verscheidene vaste poorten werken, worden geëncapsuleerd in het HTTP-protocol. Wanneer een SSH-verbinding geëncapsuleerd wordt in HTTP, zal deze SSH-connectie via een HTTP-tunnel door poort 80 gaan i.p.v. poort 22.
Een firewall zal, indien deze correct geconfigureerd is, alle trafiek op poort 22 blokkeren, waardoor SSH niet mogelijk is, maar omdat deze nu via poort 80 geleid wordt, zal de SSH-verbinding wel door de firewall kunnen.

Mijn doel is om een andere gelijkaardige methode te zoeken om een "verboden" communicatie vast te leggen die de firewall omzeilt of misleidt, om zo uiteindelijk een "command & control" kanaal te verkrijgen. Dit kan verwezenlijkt worden door het manipuleren van reeds bestaande protocollen of het ontwikkelen van een eigen kanaal. Tijdens de stage zijn er twee verschillende kanalen verwezenlijkt, die in deze scriptie besproken zijn.

Keywords: Firewall, Covert Communication Channel, Command & Control, C&C-kanaal, Remote-Access, Powershell, Malware, Tor.

1. Inleiding

1.1. Covert Communication Channels

Een *covert communication channel*, ook wel *command and control channel (C&C-channel)* genaamd, is een geheim communicatie kanaal tussen twee processen/systemen die niet met elkaar mogen communiceren.

Om te begrijpen waarom en hoe deze C&C-channels gebruikt worden, moet eerst geweten zijn hoe de beveiliging van een bedrijfsnetwerk in zijn werk gaat.
Een bedrijfsnetwerk wordt door verschillende technologieën beschermd waaronder een Intrusion Detection System (IDS), dit is een passieve beveiliging die al het netwertrafiek zal monitoren op verdacht trafiek. Dit kan gaan over een web-request naar een ip-adres of domein dat op de *blacklist* staat, dit is een lijst met websites die bekend staan voor slechte doeleinden gebruikt te worden. Wanneer de IDS merkt dat het gecontacteerde adres op de *blacklist* staat, of er verdacht trafiek tussen een *end-point* en een systeem plaatsvindt, bijvoorbeeld een zeer grote HTTP-request (vb.: 10MB), zal de IDS dit melden aan de firewall. De firewall zal dan de connectie tussen de *end-point* en de PC verbreken en al het verkeer van/naar die *end-point* blokkeren. Wat de IDS behandelt als verdacht trafiek hangt natuurlijk af van de configuraties/regels van de IDS zelf, dit zal in elk netwerk anders zijn afhankelijk van wat de werknemers nodig hebben om hun job te kunnen uitvoeren.
Indien een IP-adres/domein gecontacteerd wordt dat op de *blacklist* staat, wordt de web-request geblokkeerd of *gedropt*.
Sommige bedrijfsnetwerken hebben een Intrusion Prevention System (IPS), dit is gelijkaardig aan de IDS met het verschil dat de IPS in staat is zelf trafiek te blokkeren, terwijl de IDS dit enkel meldt aan de firewall om het te blokkeren. Meestal beschikt een bedrijfsnetwerk over ofwel een IDS ofwel een IPS, in uitzonderlijke gevallen worden ze ook samen geïmplementeerd.
Meeste bedrijfsnetwerk beschikken over een mailserver die instaat voor het controleren en onderhouden van het mailtrafiek, door regels in te stellen zullen alle mails aan bepaalde eisen moeten voldoen zodat deze doorgelaten worden.
Het belangrijkste onderdeel is de firewall, die restricties zal voorzien op protocollen die naar buiten (*outbound*) en/of naar binnen (*inbound*) willen communiceren door de poorten, die deze protocollen gebruiken, te blokkeren.
Al deze onderdelen vormen samen de *"network perimeter defences"* (NPD) van het bedrijfsnetwerk en beveiligd alles binnen het bedrijfsnetwerk van de buitenwereld.

De firewall kan gezien worden als een soort grenswachter, die bepaald wie er binnen en/of buitenmag. Om C&C-kanalen op te stellen is het belangrijk te weten wat de firewall wel of niet doorlaat. Zo wordt vanuit het perspectief van de firewall, voor een bedrijfsnetwerk, gekeken welke protocollen nodig zijn voor de werknemer om zijn job te kunnen doen. Een gemiddelde werknemer heeft toegang noig tot het web en zijn mails, daarom worden de protocollen HTTP, HTTPS en DNS, voor de browser, en SMTP, voor de mails, toegelaten door de firewall.
Uiteraard zijn er nog vele andere protocollen die handig zijn voor gebruik binnen het netwerk maar niet voor communicatie met de buitenwereld. Zo hebben werknemers geen nood aan SSH-verbindingen met de buitenwereld of connecties met een FTP-server. Daarom worden deze protocollen samen met talloze anderen geblokkeerd door de firewall om zo ook te voorkomen dat een *attacker* van buitenaf, via deze protocollen, een connectie naar een systeem binnen het netwerk kan vastleggen.
Voor dit project wordt gefocust op het "misbruiken" van protocollen om van buiten het bedrijfsnetwerk toch een C&C-kanaal vast te leggen met een systeem dat zich binnen het beveiligde netwerk bevindt. Het is daarom zeer belangrijk te weten welke protocollen nodig zijn voor de gebruikers binnen dat beveiligde netwerk, om zo te achterhalen welke protocollen beschikbaar zijn voor het opstellen van een C&C-kanaal. Voor dit project werd gekozen het gebruik van het SMTP protocol (mails) en het TOR-netwerk protocol om C&C-kanalen op te stellen.

Eens een C&C kanaal actief is, kan een *hacker* van buiten de firewall communiceren met een computer/proces dat door de NPD beschermd is. Het doel van een C&C-kanaal is dat het onopgemerkt blijft door de gebruiker en de NPD, daarom moeten alle regels van de NPD in acht genomen worden. Buiten alle trafiek dat een C&C genereerd, dat geïnspecteerd wordt door de NPD, is er ook de factor van het systeem en de gebruiker van dat systeem. Zo moet ervoor gezorgd worden dat het de gebruiker niets merkt van het C&C-kanaal of dat er geen anti-virus, of andere beveiligingsmaatregelen, het C&C-kanaal ontdekken. Bij goed beveiligde systemen zullen processen op dat systeem ook nagegaan worden. Zo zal het uitvoeren van bepaalde commando's opgemerkt worden (scripts e.d.) door een anti-virus. Een anti-virus programma zal nagaan of dit "verdacht" gedrag gekend staat in zijn database, zo niet is het niets kwaadaardig en zal er geen actie ondernomen worden. Daarnaast is het ook belangrijk dat de gebruiker van het systeem niets opmerkt van het C&C-kanaal of de *malware*, gebruikt om dit kanaal op te stellen.

1.2. Malware

Malware, kort voor *Malicious Software*, is software dat allerlei kwaadaardige bedoeling vervult, bijvoorbeeld het verzamelen van gevoelige data, computer programma's onderbreken/misbruiken, enzovoort. Er zijn verschillende soorten malware, ieder met zijn eigen doel of specifiek nut. In dit project is er malware geschreven die een C&C-kanaal zal verwezenlijken. Voorbeelden van dit soort malware zijn onder andere IRC-channel malware, TOR-channels. Omdat andere soorten malware niet relevant zijn voor dit project zullen deze hier niet besproken worden.

Voor het schrijven van malware zijn er enkele aandachtspunten waar zeker op gelet moet worden. Zo is het belangrijk te weten wat de beperkingen zijn van je toegangsrechten. Als ontwikkelaar moet je er van uit gaan dat je malware zal werken met standaard gebruikersrechten, d.w.z. dat men geen *admin* rechten heeft om bepaalde bestanden/configuraties te manipuleren. Het doel van de malware is dan ook om *remote access* te verkrijgen tot een systeem om zo *privilege escalation* te verwezenlijk en data te verzamelen.
Een ander aandachtspunt is het CPU-gebruik van de malware. Malware, zoals elke andere software, eist een bepaald percentage CPU-gebruik op om te werken. Dit kan vergeleken worden met *resources* die verdeeld worden, elk programma heeft een bepaalde hoeveelheid van die *resources* nodig, dus moeten deze resources verdeeld worden over verschillende processen. Wanneer een programma veel resources vergt, zullen er weinig resources overblijven voor andere processen waardoor deze andere processen trager of zelf niet zullen werken. Het is dan ook cruciaal dat malware *lightweight* is, wat wil zeggen dat het weinig CPU *resources* vergt. Dit zodat de programma's, die de gebruiker wil gebruiken, normaal werken zodat de gebruiker geen argwaan krijgt dat er ongewenste processen lopen. Deze argwaan leidt namelijk tot inspectie dat dan leidt tot het vinden van de malware processen, wat vermeden wil worden.

Malware bestaat hoofdzakelijk uit drie onderdelen, namelijk verdeling (*delivery*), het C&C-kanaal zelf en *persistency*. Met verdeling wordt bedoeld dat de malware zichzelf moet kunnen verspreiden en/of zichzelf in een systeem kan nestelen. Enkele mogelijkheden hiervoor zijn onder andere drive-by downloads, *malicious* documents, social engineering etc. Het C&C-kanaal is het gebruik van de technologie die het C&C-kanaal voorziet, dit is de ware *slechte* intentie of "nut" van de malware die de schade aan het systeem verricht (*payload*).
Persistency wil zeggen dat de malware "volhardend" is. Wanneer malware op een systeem geïnstalleerd wordt, zal het programma ook uitgevoerd worden. Dit programma blijft actief zolang het systeem actief is, zodra de gebruiker het systeem afsluit, stopt de malware ook met werken. Als een malware programma *persistent* is, zal die malware steeds uitgevoerd worden telkens het systeem actief is. Als de malware niet *persistent* is, zal het programma enkel actief zijn vanaf dat het op het systeem geïnstalleerd en uitgevoerd wordt, en blijft dit eenmalig lopen tot dat de gebruiker het systeem afsluit.

Als opdracht voor dit project werd gevraagd enkel te werken aan het C&C-gedeelte van de malware uit te voeren en niet zo zeer het onderdeel *delivery* noch *persistency*.

2. Technisch

2.1. Technologieën

2.1.1. Powershell

> Wikipedia: "Powershell is een objectgeoriënteerde shell- en scripttaal voor Microsoft Windows."

Deze technologie beschikt over een command line shell en interpreteert een scripting taal dat gebaseerd is op .NET framework. Met behulp van deze .NET scripting taal is het mogelijk om Windows processen te automatiseren of administratieve taken/zaken van het systeem aan te passen. Omdat het rond een .NET framework gebouwd is, is het mogelijk andere talen te implementeren zoals bijvoorbeeld C#, C++, C, enzovoort.
Zo is het ook mogelijk om met verschillende objecten van Microsoft applicaties te werken. Een voorbeeld hiervan, gebruikt voor dit project, zijn Outlook objecten.

Outlook Objecten

Powershell Outlook objecten geven de gebruiker toegang tot alle eigenschappen en functies waarover de Outlook applicatie beschikt. Het is mogelijk de gehele file-structuur te verkennen en alle eigenschappen van een mail te lezen, van zender tot ontvanger, onderwerp, de inhoud en de bijlagen. Andere velden van een mail zoals de tijdstip van versturen/ontvangst e.d. zijn ook beschikbaar.

Zoals reeds vermeld zijn zowel de eigenschappen, als de functies van de Outlook applicatie ter beschikking. Zo is het mogelijk om Outlook rules aan te maken. Deze zorgen voor geautomatiseerde regels waar Outlook naar moet leven. Een voorbeeld van deze Outlook rules zou een spamfilter zijn, die mails van een bepaalde zender in een junk-folder zal plaatsen of mogelijk verwijderen. Andere functies die beschikbaar zijn, zijn onder andere, het versturen van e-mails, het aanmaken van agenda-events en het manipuleren (verwijderen/verplaatsen) van mails.

In dit project wordt er gebruik gemaakt van Outlook objecten om een C&C-kanaal via Outlook mails te verwezenlijken. Tijdens de ontwikkeling van deze malware werd er opgemerkt dat er geen toegang was tot alle mogelijke eigenschappen en/of functie die de outlook objecten zouden moeten bieden. De oplossing voor deze beperking is een externe *library* genaamd *Redemption*.

Redemption Library

Vanwege een *security patch* op Outlook versies 98 en hoger, is er een restrictie op de toegang tot eigenschappen en/of functies van Oulook objecten.
Om toegang te krijgen tot deze eigenschappen en/of functies van de Outlook objecten, ondanks deze restricties, bestaat er een externe *library* genaamd *Redemption*. Deze library zorgt voor een *bypass* of omzeiling van de *security patch* op Outlook. Het maakt gebruik van legitieme objecten en zal hiermee werken om de security patch te omzeilen. Zo krijgt de gebruiker toch toegang tot de eigenschappen en kan hij/zij de functies gebruiken dat een Outlook object te bieden heeft.

Kort samengevat geeft Redemption library toegang tot de volle capaciteit van Outlook objecten die anders beschermd worden door de security patch. Een *library* integreren in powershell is mogelijk d.m.v. het `Add-Type -assemblyName <library>` commando.

2.1.2. Microsoft Visual Studio

Microsoft Visual Studio (MSVS) is een Integrated Development Environment (IDE), van Microsoft, gebouwd rond het .NET framework. Deze IDE biedt de mogelijkheid om applicaties te schrijven voor talloze platformen die .NET ondersteunen. Er worden ook talloze .NET libraries ter beschikking gesteld om het ontwikkelen eenvoudiger en overzichtelijker te maken. MSVS biedt de mogelijkheid om in verschillende talen te programmeren zoals bijvoorbeeld C, C++, C# (C-sharp) en Visual Basic for Applications(VBA).

C-sharp

C-sharp (C#) is een objectgeoriënteerde programmeertaal gebaseerd op .NET framework. Het biedt vele mogelijkheden aan en heeft een verscheidenheid aan libraries. Voor dit project werd één van de malware programma's in de C# taal ontwikkeld. Dit gebeurde allemaal met gebruik van de eenvoudig te gebruiken interface van Microsoft Visual Studio IDE.

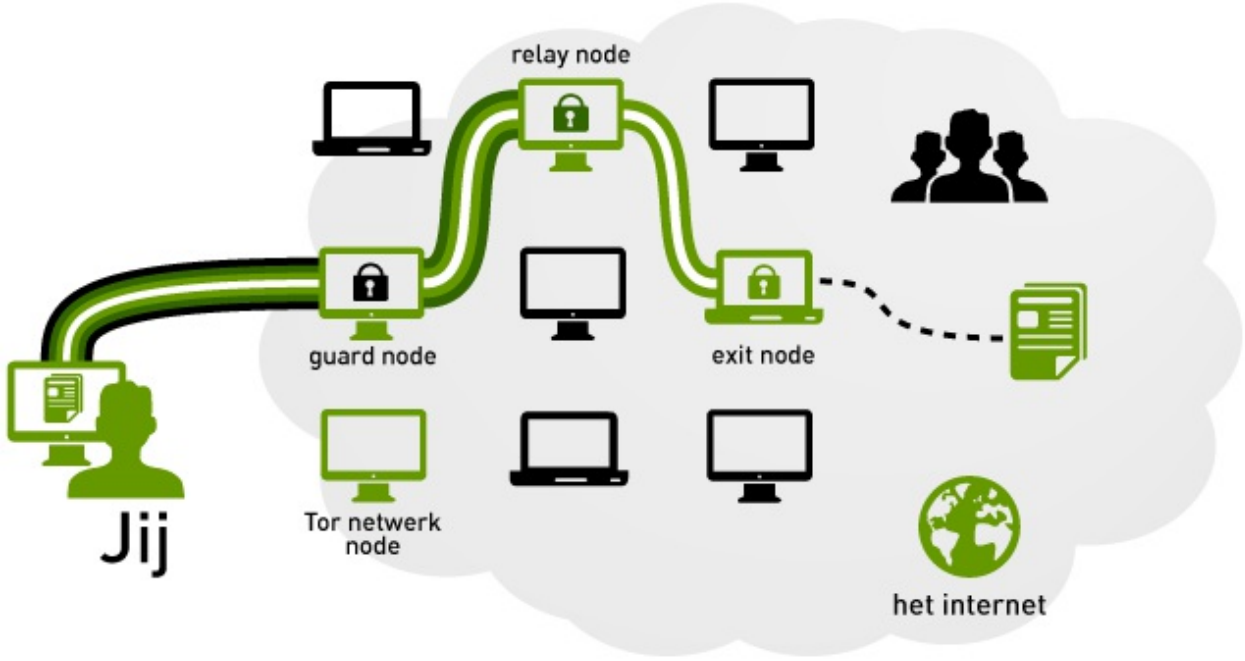
Visual Basic for Applications

Visual Basic for Applications (VBA) is een programmeertaal gebaseerd op het .NET framework. Deze programmeertaal wordt voornamelijk geïmplementeerd voor het schrijven van Microsoft Office Macro's. Dit zijn kleine programma's die de werking van een ander programma/applicatie ondersteunen. Zo kan men processen voor een applicatie automatiseren m.b.v. macro's.
VBAis een vereenvoudigde versie van C# en is snel aangeleerd.
In dit project werd VBAgebruikt voor het schrijven van *dropper macro's*. Dit zijn macro's die malware of andere programma's kunnen verspreiden op geautomatiseerde wijze.

2.1.3. The Onion Router

The Onion Router, beter bekend als TOR, is een netwerktechnologie die zorgt voor complete anonimiteit op het Internet door middel van encryptie. Deze encryptie bestaat uit lagen, gelijkaardig aan een ui, wat naar Engels vertaald *onion* is, vandaar de naam Onion Router. Deze technologie zorgt ervoor dat de activiteit van een gebruiker op het Internet niet terug getraceerd kan worden naar hem/haar.

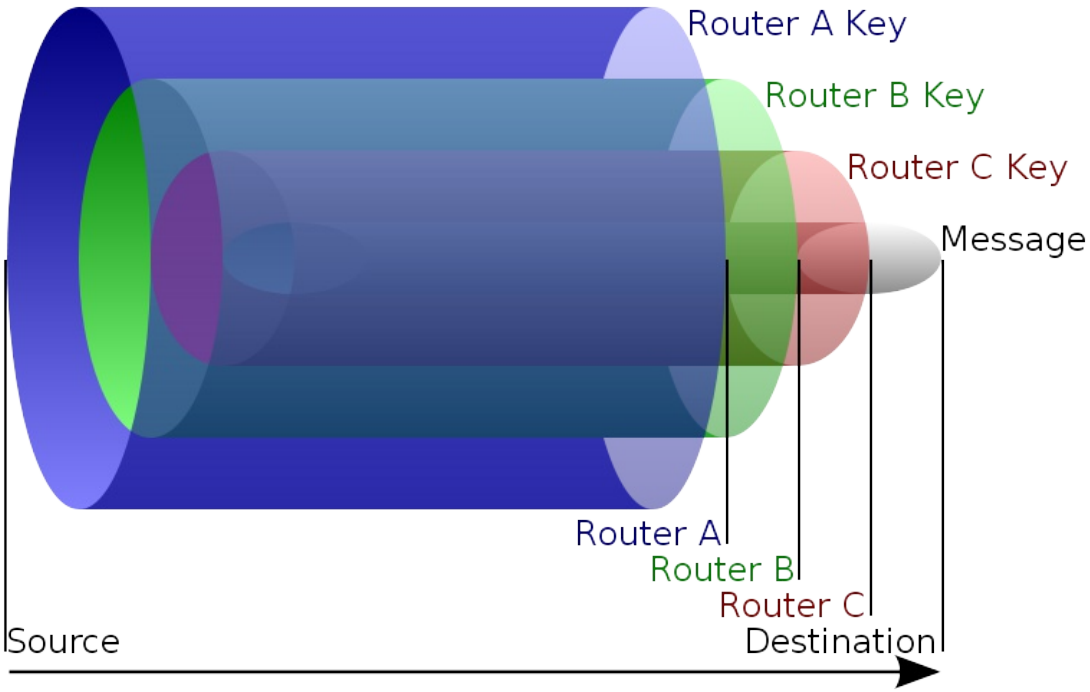
Het TOR-netwerk bestaat uit meerdere *nodes*, die in een bepaalde communicatie elk hun eigen rol vervullen. Zo zijn er *entry nodes* of *guard nodes*, die beschouwd worden als een ingang naar het TOR-netwerk. Hiervan zal er steeds slechts één aanwezig zijn in elke communicatieroute. *Relay nodes* zorgen voor *relay* of doorschakeling door het netwerk. Hiervan zal er tevens minstens één aanwezig zijn per connectie. Hoe meer *relay nodes*, hoe moeilijker het traceren van de oorsprong van het pakket wordt. Tenslotte zijn er ook nog *Exit Nodes*, die beschouwd kunnen worden als de uitgang uit het TOR-netwerk. Dit is de benaming voor de laatste *node* die rechtstreeks verbonden is met het doel dat de gebruiker wenst te bereiken, bijvoorbeeld een webserver e.d. Ook hiervan zal er slecht één aanwezig zijn per connectie.
Voor elke connectie die gemaakt wordt, zal er steeds een verschillende route worden vastgelegd.



figuur 1: TOR Netwerk Nodes

Het principe van TOR draait rond gelaagde encryptie van connecties. Dit wil zeggen dat elke node zijn eigen encryptielaag heeft op het originele bericht.
Voordat er communicatie vastgelegd wordt tussen gebruiker Aen B, zal er eerst een route doorheen het TOR-netwerk vastgelegd worden. Vervolgens worden de sleutels van elke *node* in die route verzameld en aan gebruiker gegeven voor encryptie.
Stel dat er het minimum vereiste (drie) nodes aanwezig zijn, namelijk de *guard/entry node, één relay node, exit node*, dan zullen er slechts drie lagen encryptie aanwezig zijn. Eerst wordt het bericht geëncrypteerd met de sleutel van de laatste node (= *exit node*). Dit geëncrypteerde bericht zal daarna geëncrypteerd worden met de sleutel van de tweede node (= *relay node*). Ten slotte zal dit geheel nogmaals geëncrypteerd worden met de sleutel van de eerste node (= *entry node*). Merk op dat de encryptie gebeurt in tegengestelde richting van de communicatie.

Onderstaande afbeelding toont de gelaagde encryptie van een bericht dat door het TOR-netwerk gaat. Hier zien we nogmaals de lagen van encryptie, gelijkaardig aan de opbouw van een ui, waaraan deze technologie dus zijn naam te danken heeft. TOR maakt gebruik van het RSA-encryptie algoritme voor de encryptie van alle data.



Figuur 2: TOR Message Encryption (bron: https://nl.wikipedia.org/wiki/Onion_routing#/media/File:Onion_diagram.svg)

Wanneer de encryptie van het bericht voltooid is, zal het bericht verzonden worden.
Eerst passeert het bericht de *entry node*. Omdat het bericht als laatste geëncrypteerd is met de sleutel van deze node, kan deze node deze (eerste) laag decrypteren. Eens de *entry node* de eerste laag gedecrypteerd heeft, zal het kunnen achterhalen welke node er volgt in de communicatieroute.
Vervolgens zal het pakket doorgestuurd worden naar de volgende *node*, in dit geval de eerste *relay node*. Nu de eerste encryptielaag reeds gedecrypteerd is, is de tweede laag blootgesteld voor deze *relay node*. Omdat deze laag geëncrypteerd werd met de sleutel van deze *relay node*, kan deze laag enkel gedecrypteerd worden door die *relay node*. Ook hier zal de *relay node* na decryptie de volgende *node* kunnen bepalen. Wanneer de volgende *node* gekend is, zal het pakket doorgegeven worden naar die *node*. In dit voorbeeld zal dat de laatste *node* zijn, dit is steeds de *exit node*. De *exit node* zal de laatste laag encryptie kunnen decrypteren, omdat deze laag geëncrypteerd is met de sleutel van deze *exit node*. Wanneer de laatste laag gedecrypteerd is, blijft het originele (*clear*) bericht (pakket) over. De *exit node* zal dan kunnen achterhalen wie de ontvanger hoort te zijn en waar deze zich bevindt, en zal dit bericht tenslotte aan de ontvanger bezorgen.
Merk op dat tijdens verzending de encryptielagen als het ware van het originele pakket worden gepeld (analogie met een ui).

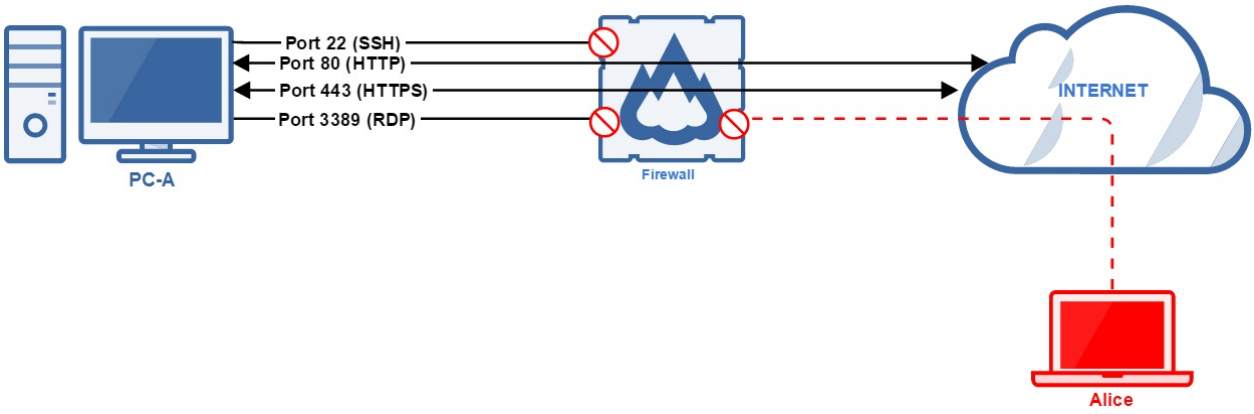
In dit project is gebruik gemaakt van TOR om het ITRAteam anoniem te houden tijdens hun redteam tests. Er is ook gebruik gemaakt van TOR hidden services om een anoniem C&C-kanaal op te stellen naar een *remote desktop*.

Hidden Services

TOR *Hidden Services* biedt de mogelijkheid voor gebruikers om een service te hosten binnen het TOR-netwerk, met behoudt van hun anonimiteit. D.w.z. dat een gebruiker een service zoals bijvoorbeeld een web server, SSH server, FTP server, enzovoort, kan openstellen voor andere gebruikers, binnen het TOR netwerk, zonder hun IP-adres vrij te geven.
Zo wordt er in één van de malware programma's van dit project een *hidden service* opgesteld dat alle trafiek, op het

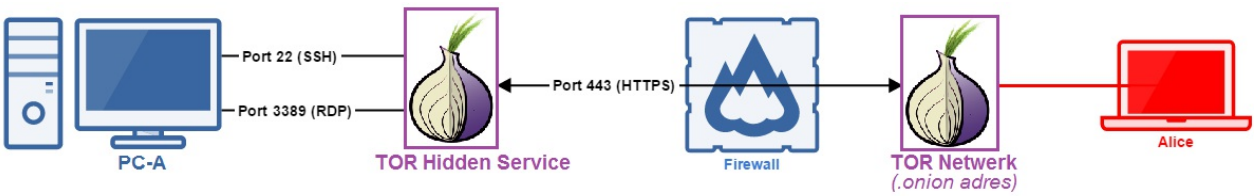
adres van de *hidden service* toekomt, zal doorsturen naar een geconfigureerde poort van de lokale machine van een gebruiker. De malware zal een poort van het systeem openstellen via het TOR netwerk, bijvoorbeeld poort 3389 (Remote Desktop Protocol). Dit protocol wordt normaal gezien door de firewall zowel *inbound* als *outbound* geblokkeerd, maar de malware zal ervoor zorgen dat deze poort toegankelijk is via het TOR netwerk door die poort open te stellen voor alle trafiek van de *hidden service* die opgesteld wordt. Wanneer de *hidden service* opgesteld wordt, wordt gedefiniëerd hoe het trafiek doorgestuurd moet worden, bijvoorbeeld: stel dat een *cliënt* de *hidden service* aanspreekt op poort 3389, kan ingesteld worden dat alle trafiek op die poort doorgestuurd wordt naar de lokale poort 3389 van de *host* van de *hidden service*. Op die manier kan de firewall blokkade omzeild worden en kan er een communicatie vastgelegd worden naar de lokale poort van het systeem, mits de infrastructuur TOR toelaat.

Onderstaande afbeelding toont een situatie waarin een *hacker* (Alice) een RDP (poort 3389) connectie probeert vast te leggen naar PC-Avia het internet. Maar de firewall blokkeerd alle RDP-trafiek waardoor deze connectie niet toegelaten wordt en dus niet mogelijk is. Merk op dat alle trafiek via poort 80 (HTTP) en 443 (HTTPS) wel doorgelaten wordt. Dit komt omdat, zoals eerder vermeld, de protocollen die op deze poorten werken, namelijk HTTP en HTTPS, nodig zijn om te kunnen *browsen* op het web, wat een werknemer nodig heeft om zijn job te kunnen uitvoeren.



TOR Hidden Service probleem situatie

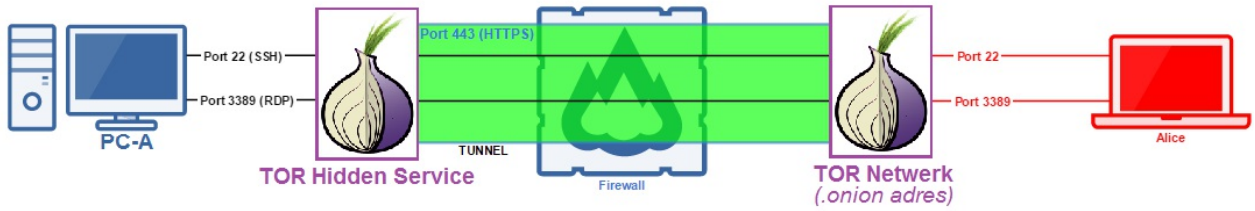
De TOR *client* zal de, met de nodige configuraties, de *hidden service* opstellen. Vervolgens wordt er een HTTPS (poort 443) connectie vastgelegd tussen de *hidden service* en het TOR netwerk. Daar zal de *hidden service* ter beschikking gesteld worden a.d.h.v. een *".onion"* adres dat afgeleid is van de *public key* van de *hidden service* (vb.: ayr7h93ke7nh6bv0.onion). Als Alice dit *".onion"* adres kent, kan zij hiermee connecteren via het TOR-netwerk. Die connectie wordt dan van het *".onion"* adress op het TOR netwerk doorgestuurd naar de *hidden service* via de HTTPS verbinding tussen de *hidden service* en het TOR netwerk dat door de NPD gaat. Tenslotte wordt de connectie tussen Alice en de *hidden service* doorgelinkt naar PC-A, zo heeft Alice een onrechtstreekse connectie naar PC-Avia de TOR *hidden service*.



TOR Hidden Service connectie

Afhankelijk van de configuraties van de hidden services kan Alice aparte poorten aanspreken op het *".onion"* adres, die connectie wordt dan, zoals hierboven, doorgestuurd naar de *hidden service*. De *hidden service* kan dan zien welke poort er werd aangesproken op het *".onion"* adres en bepaalt via de configuraties naar welke lokale poort van PC-Adie connectie doorgestuurd wordt. Stel dat de *hidden service* zodanig geconfigureerd is dat trafiek op poort 3389 van het *".onion"* adres doorgestuurd moet worden naar de lokale poort 3389 van PC-A. Dan kan Alice een connectie maken met de lokale poort 3389 van PC-Adoor te connecteren met poort 3389 van het *".onion"*

adres. Deze gehele connectie zal dan vanaf het *".onion"* adres, op het TOR netwerk, naar de *hidden service*, via de reeds vermeldde HTTPS-verbinding, doorgestuurd worden. De connectie wordt niet geblokkeer omdat de firewall de trafiek van het HTTPS protocol doorlaat. In dit geval werkt de HTTPS-connectie als een tunnel voor een verbinding voor andere protocollen waardoor deze wel toegestaan worden.
Onderstaand afbeelding toont aan hoe een RDP-connectie (poort 3389) door een HTTPS-connectie *getunneld* wordt m.b.v. de connectie tussen TOR-netwerk (*".onion"*) en de *hidden service*.



TOR Hidden Service HTTPS Tunnel forwarding

2.2. Outlook C&C

Als eerste C&C-kanaal werd gekozen voor de mailinfrastructuur m.b.v. de Microsoft Outlook applicatie. Om met Outlook te werken, werd er gebruik gemaakt van Powershell. Zoals reeds besproken biedt Powershell de mogelijkheid aan om met objecten van de Outlook applicatie te werken. Deze objecten bieden toegang tot eigenschappen van mails en de implementatie van functies, die Outlook beschikt, in een script.
Omdat Outlook versies 98, en hoger, een beperkte toegang bieden tot die eigenschappen en functies vanwege een *security patch*, werd de *Redemption Library* gebruikt. Deze *library* "omzeilt" deze security patch en geeft de gebruiker volledige toegang tot alle eigenschappen en functies die Outlook te bieden heeft.

De reden voor deze technologie, is omdat het mailtrafiek binnen een bedrijfsnetwerk zeer luidruchtig is. Een overvloed aan mailtrafiek van één kanaal zal dus niet opvallen voor iets/iemand die de mailtrafiek zou controlleren. De enige controle die hier bestaat is de *Mailserver*. Deze zal, voor elke mail dat passeert, controleren of deze aan zijn regels voldoet. Indien dit niet het geval is, zal de *Mailserver* deze mail terugsturen naar het zendadres met een melding dat die mail niet toegestaan is.
Het is dus belangrijk op voorhand te weten welke regels de mails van dit C&C-kanaal moeten naleven.

2.2.1. Werking

De principiële werking van dit C&C kanaal is zeer eenvoudig. Het script zal zich, op de achtergrond, aan de Outlook applicatie *client* nestelen. Wanneer deze applicatie actief is zal dat script steeds luisteren of er mails ontvangen worden die verzonden zijn door de *attacker*.
Zodra er een mail van de *attacker* ontvangen wordt, zal het script deze mail verbergen voor de gebruiker. Dit gebeurt m.b.v. een *Outlook Rule* dat het script zelf zal toevoegen. Deze *Outlook Rule* dient als een soort spamfilter die alle ontvangen mails van de *attacker* zal wegsteken in een "verborgen" folder. Op deze wijze zorgt het script er ook voor dat de gebruiker geen notificaties te zien krijgt met de melding dat hij/zij een mail heeft ontvangen.
Eens de ontvangen mail verborgen is, zal het script de mail lezen. De inhoud van de mail zal worden gekopieerd naar een *batch* bestand. *Batch* bestanden zijn scripts waarin DOS commando's staan en kunnen geïnterpreteerd en uitgevoerd worden door het Windows *Command Prompt*.
Zodra dit *batch* bestand opgesteld is, zal het script dit bestand uitvoeren en worden zo de commando's, die de *attacker* heeft doorgemaild, op het systeem uitgevoerd. Alle resultaten of *feedback* die verkregen wordt t.g.v. deze commando's zullen opgeslagen worden in een tekst bestand.
Wanneer de commando's voltooid zijn en dit tekst bestand aanwezig is, wordt er door het script een nieuwe mail opgesteld. Aan deze mail wordt het tekst bestand met alle resultaten van de commando's als bijlage toegevoegd. Vervolgens wordt deze mail verstuurd naar de *attacker* om de resultaten mee te delen.
Tenslotte zal het script zowel de ontvangen als de verstuurde mails verwijderen. Ook het *batch* bestand en het tekst bestand worden verwijderd. Op die manier blijven er geen sporen achter en zal er geen argwaan ontstaan bij de gebruiker van het systeem.
Telkens de *attacker* een mail stuurt, zal dit proces zich herhalen.

Dit C&C kanaal biedt de mogelijkheid voor de *attacker* om commando's te versturen naar zijn doelwit en die commando's op zijn systeem uit te voeren. Telkens de *attacker* commando's wenst uit te voeren zal hij een mail terugkrijgen met resultaten van de commando's zodat hij weet wat er zich voordoet.
Zo kan hij scans gaan uitvoeren om extra informatie te vergaren over zijn doelwit.

Kort opgesomde werking:

1. Controlleren of de Outlook applicatie actief is op het systeem
    - Indien actief --> Setup
    - Indien inactief --> Watchdog = Blijf controleren.
2. CreateRule = Controleer of Outlook Rule al bestaat

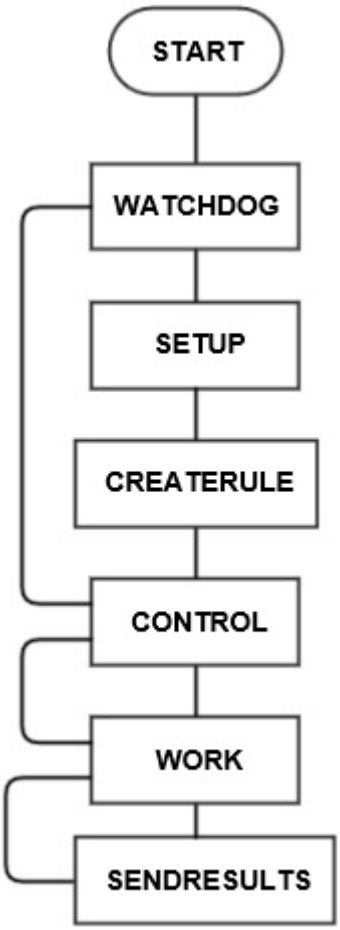
    - Zo niet --> Creëer Outlook Rule (spamfilter)
    - Zo wel --> ga verder
3. Controleer of er een mail van de *attacker* werd ontvangen.
    - Zo niet --> blijf controleren
    - Zo wel --> Work
4. Lees mail en zet commando's om naar *.bat* bestand
5. Voer *.bat* bestand uit en schrijf resultaten naar een tekst (*.txt.*) bestand.
6. Verstuur e-mail naar *attacker* met het tekst bestand als bijlage.
7. Verwijder tekst bestand, *.bat* bestand en de ontvangen mail van de *attacker*.

2.2.2. Code

Deze C&C bestaat uit een script dat volledig geschreven is in Powershell. Het is mogelijk om scripts met powershell te schrijven in Microsoft Visual Studio of de Powershell Integrated Scripting Environment (ISE).
In dit deel wordt de code toegelicht a.d.h.v. pseudo-code. Alsook het gebruik van Outlook objecten, de Redemption Library en andere Powershell functionaliteiten worden toegelicht.
Zo wordt elke functie overlopen a.d.h.v. en de gebruikte code m.b.v. pseudo-code besproken.
De functies in dit script zijn:

- Watchdog
- Setup
- Control
- CreateRule
- Work
- SendResults

Van deze functies kan de *Watchdog* functie worden beschouwd als de passieve staat, en de *Control* functie als de actieve staat. Onderstaand afbeelding toont een vereenvoudigd overzicht van de flowchart van deze code.



Add-Type

Het `Add-Type` commando maakt het mogelijk om *assemblies* van een .NET framework klasse te initialiseren. Nadien is het mogelijk van deze klassen objecten aan te maken.
Omdat dit script gebruik maakt van Outlook objecten, moet de .NET *assembly* van Outlook toevoegd worden. Dit

gebeurt met het commando `Add-Type -Assembly "Microsoft.Office.Interop.Outlook"` . Merk op dat er verduidelijkt moet worden dat het gaat om een *assembly* d.m.v. het `-assembly` argument. Vervolgens geeft men de naam van de *assembly* mee als argument. In het geval van Outlook is deze naam `Microsoft.Office.Interop.Outlook` .

Watchdog

Zoals reeds vermeld kan deze functie beschouwd worden als de passieve staat van het script. De functie gaat continu controleren of er een versie van de Outlook applicatie actief is op het systeem, zoals een soort waakhond. Zodra er een versie van de applicatie actief is, zal de functie vijf seconden wachten (*delay*) vooraleer het script uit de functie breekt en de *main* code uitvoert. Dit is zo gedaan omdat het cruciaal is dat de applicatie volledig is opgestart voordat er objecten van deze applicatie worden aangemaakt.

Door een `While()` lus te implementeren, zal er een zeker proces zich steeds herhalen zolang er aan een voorwaarde voldaan wordt. Deze voorwaarde is dat er geen instantie van de Outlook applicatie actief is. Deze voorwaarde ziet er in code als volgt uit `(Get-Process outlook) -eq $null` . In dit proces zal `Get-Process outlook` bepalen of er een proces actief is met de naam "Outlook". De vergelijking `-eq $null` zal waar zijn indien er géén actief proces genaamd Outlook bestaat, en dus gelijk is aan (`-eq`) *null* (`$null`).
Het proces dat herhaald wordt is een *delay* van één seconde. Dit wil zeggen dat zolang er géén instantie van de Outlook applicatie actief is op het systeem, zal er een *delay* van één seconde plaatsvinden. Dit gebeurt met het `Start Sleep -s <aantal seconden>` commando.

Wanneer deze lus wel een actieve instantie van de Outlook applicatie detecteerd, zal de functie uit de `while()` lus breken en zal er, zoals vermeld een *delay* van vijf seconden gestart worden. Dit is nodig zodat de applicatie de tijd heeft om volledig op te starten, wat nodig is om de objecten te kunnen initialiseren.
Na deze vijf seconden zal de *setup* functie worden opgeroepen.

Setup

Dit script werkt met verschillende objecten en variabelen. De setup functie zorgt voor de initialisatie van deze objecten en variabelen. Zo maakt deze functie een object aan van Outlook en een Redemption object. Dit doe je door een variabele te initiëren en deze gelijk te stellen aan een object als volgt `$a = New-Object <Instantie>` . Merk op dat je moet verduidelijken van welke instantie je een object wenst te maken. Voor Outlook is dit `Outlook.Application` en voor Redemption is dit `Redemption.RDOSession` .

Nu de objecten zijn geïnitialiseerd moet de juiste *namespace* nog bepaald worden. Dit wordt gerealiseerd met het commando `$ns = <application>.Session` , waar `<application>` het object van Outlook voorstelt. De *namespace* geeft toegang tot alle functionaliteiten en eigenschappen.
Vervolgens wordt ervoor gezorgd dat er in de huidige sessie van de gebruiker gewerkt wordt. Dit is mogelijk door de `Logon()` methode te gebruiken van het Outlook object. Dat ziet er als volgt uit voor het outlook object `<application>.Session.Logon()` , en voor het Redemption object is dit als volgt `<application>.Logon()` .

Als laatste wordt er een object aangemaakt dat verwijst naar een folder van de Outlook applicatie. Dit is de *mail* folder waarin gewerkt zal worden wanneer er een mail van de *attacker* ontvangen wordt. Er zijn een hele reeks standaard folders waar uit gekozen kan worden. In dit script is er gekozen voor de *Junk* folder, omdat deze standaard aanwezig is. Een ander voordeel van deze map is dat de gebruiker niet verwittigd wordt wanneer er iets in deze folder gebeurd. Meestal hebben de gebruikers niet de gewoonte om in hun *Junk* folder te kijken. Een folder object aanmaken kan met de methode `GetDefaultFolder` binnen de *namespace* of *session* van het huidig Outlook object. Als parameter vraagt deze methode een folder object dat gekozen moet worden. Dit kan a.d.h.v. een object van `[Microsoft.Office.Interop.Outlook.OlDefaultFolders]::<folder>` . In de plaats van `<folder>` kan dan de naam van folder, die gekozen wordt, ingevuld worden. Deze werkwijze zorgt voor een legitiem object van de *MAPI namespace* van Outlook. Dit object is nodig voor het aanmaken van de Outlook Rule (later besproken) omdat dit een *MAPI* object vereist.
Omdat de eerder besproken *security patch* van Outlook een beperkte toegang biedt tot folders en deze toegang

nodig is voor andere toepassingen in dit script, wordt er ook een folderobject aangemaakt van het Redemption object. Op die manier wordt er terug volledige toegang tot methoden en eigenschappen van het folderobject verkregen.
Het volledige commando ziet er dan als volgt uit `$f = <application>.GetDefaultFolder(23)`. Het `<application>` veld stelt hier het Redemption object voor. Merk ook op dat de `GetDefaultFolder()` methode een parameter eist. Deze parameter is een getal dat bepaald welke folder gekozen wordt. Onderstaande tabel geeft voor elke waarde, van zowel Outlook *MAPI* objecten als Redemption objecten, weer welke folder die waarde voorstelt.

Folder	Outlook MAPI	Redemption
Appointment item	1	n/a
Verwijderde items folder	3	olFolderDelete
Outbox	4	olFolderOutbox
Verzonden mails folder	5	olFolderSentMail
Inbox folder	6	olFolderInbox
Kalender folder	9	olFolderCalendar
Contacten folder	10	olFolderContacts
Jounal folder	11	olFolderJournal
Notities folder	12	olFolderNotes
Taken folder	13	olFolderTasks
Concepten folder	16	olFolderDrafts
Publieke folders - Alle publieke folders *	18	olPublicFoldersAllPublicFolders
Conflicten folder *	19	olFolderConflicts
Folder met synchronisatie foutmeldingen *	20	olFolderSyncIssues
Folder met locale gebreken (*failures*) (subfolder van SyncIssues) *	21	olFolderLocalFailures
Folder met Server gebreken (*failures*) (subfolder van SyncIssues) *	22	olFolderServerFailures
Ongewenste Mails (Junk) Folder	**23**	**olFolderJunk**
Folder met RSS-feed	n/a	olFolderRssFeed
To-Do folder	n/a	olFolderToDo
Top-Level folder in Managed Folder groep *	n/a	olFolderManagedEmail
Aanbevolen contacten	n/a	olFolderSuggestedContacts

CreateRule

Nu alle nodige objecten en variabelen gedeclareerd zijn in de *Setup* functie, zal de *CreateRule* functie opgeroepen worden. Deze functie zorgt voor het aanmaken van de Outlook rule die zal dienen als een soort *spamfilter* om alle mails van de *attacker* op te vangen.
Bij aanvang van deze functie zal eerst gecontroleerd worden of er reeds Outlook rules bestaan.

Om te controleren of er reeds een regel bestaat, worden eerst alle regels opgehaald en aan een array toegekend. Dit ziet er als volgt uit `$rules = <application>.Session.DefaultStore.GetRules()`, waar `<application>` in dit geval het Outlook object voorstelt. Hierna zal `$rules` de verzameling van alle Outlook *rules* voorstellen.
Vervolgens zullen er een reeks controles uitgevoerd worden. Als eerste controle zal er worden nagegaan of de array `$rules` al dan niet leeg is. Zo niet, zal er een tweede controle uitgevoerd worden om na te gaan of er reeds Outlook rules bestaan met dezelfde naam als mijn Outlook rule. Als er reeds een bestaat met dezelfde naam, wil dit zeggen dat er al een gemaakt is en zal het script uit de functie gaan en verder gaan met zijn normaal verloop.

Indien er nog geen Outlook rule bestaat met dezelfde naam als mijn Outlook Rule, zorgt het script ervoor dat de Outlook rule zal worden aangemaakt. Om dit te verwezenlijken moet er eerst een object van een Outlook *rule* opgesteld worden, dit gebeurt met de `Create()` functie van de Outlook rule objecten als volgt:
`$rule = $rules.create(<naam>,[Microsoft.Office.Interop.Outlook.OlRuleType]::<RuleType>)`.
Merk op dat `$rule` een object instantie creëert van de eerder gedefinieerde collectie `$rules`. De `create()` functie aanvaard twee parameters. De eerste is de naam van de regel die je wenst te maken. De tweede is het type van de Outlook rule. Dit bepaald op welke type mails de Outlook rule toegepast moet worden. Hier zijn slecht twee mogelijkheden voor namelijk, `olRuleReceive`, om te definiëren dat het gaat om ontvangen mails, of `olRuleSend`, om te definiëren dat het gaat om verzonden mails. Vervolgens wordt de nieuwe regel opgesteld met enkel vereiste eigenschappen. Zo moet gedefinieerd zijn op welk veld de Outlook rule moet letten.
Omdat het hier gaat om mails van de *attacker* is het voordelig dat er op het e-mail adres van de zender wordt gecontroleerd. Dit wordt gedefinieerd a.d.h.v. het commando `$condition = $rule.Conditions.SenderAdress`. Dit commando definieert op welke eigenschap van de mail er gelet moet worden. Het commando `$condition.Address = @("<attacker address>")` zal de waarde instellen waaraan de voorgaande eigenschap gelijk moet zijn. M.a.w. zal het commando bepalen naar welk e-mail adres moet gezocht worden, wat in dit geval het e-mail adres van de *attacker* zal zijn. Elke Outlook rule heeft een actie die uitgevoerd wordt in geval dat er aan de voorwaarde van de regel voldaan wordt. Deze kan ingesteld worden d.m.v. volgend commando `$action = $rule.Actions.<action>`. Als actie is er veel keuze, maar omdat in dit project de mails, die aan de voorwaarden voldoen, naar de *junk folder* verplaatst moeten worden, wordt de actie `MoveToFolder` gebruikt.

Tenslotte moeten de eigenschappen nog geactiveerd worden door de eigenschap `Enabled` waar te maken. Voor de voorwaarde is dit `$condition.Enabled = $true`, en voor de actie is dit `$action.Enabled = $true`. Vervolgens stellen we de gehele regel op m.b.v. volgend commando:
`[Microsoft.Office.Interop.Outlook._MoveOrCopyRuleAction].InvokeMember("Folder",System.Reflection.BindingFlags]::SetProperty, $null, <actie>,<folder>)`.
Dit commando zal de regel opstellen. Door de `InvokeMember()` functie kan de actie die ondernomen moet worden, indien aan de voorwaarde voldaan wordt, gedefinieërd. Dit gebeurd door de vooraf gedefinieerde actie in het `<actie>` veld in te vullen. Om aan te geven naar welke folder de mail verplaatst moet worden, indien die voldoet aan de voorwaarden van de Outlook rule, wordt het folderobject in het `<folder>` veld ingevuld. De folder die het script gaat gebruiken is de *Junk Folder* waarvan, in de *Setup* functie, reeds een object van geïnitialiseerd werd.
Als laatste moet de Outlook rule opgeslagen worden in de collectie. Dit gebeurd met het commando `$rules.Save()`.

Wanneer de Outlook rule opgesteld is zal het script verder gaan met zijn normale werking. Het script zal dan in zijn *actieve staat* komen te staan d.m.v. de *Control* functie op te roepen.

Nu deze Outlook rule bestaat, zal deze ervoor zorgen dat elke mail, die van de *attacker* komt, verplaatst wordt naar de *junk folder*. Herinner dat, omdat deze mail rechtstreeks naar de *junk folder* gaat i.p.v. de inbox, de gebruiker geen notificatie zal krijgen dat hij/zij een mail ontvangen heeft.

Control

Nu het script weet dat er een instantie van de outlook applicatie actief is op het systeem, en de hele *setup* (objecten, variabelen, outlook rules) afgerond is, is het script klaar voor zijn actieve staat.
De *Control* functie stelt de actieve staat voor. In deze functie zal het script steeds controleren of er mails, van de

attacker, ontvangen werden. Deze controle zal zich elke seconde herhalen, zolang de Outlook Applicatie actief is op het systeem.

In code gebeurd dit met een `while()` lus die blijft gaan zolang er een instantie van de Outlook applicatie actief is. De voorwaarde ziet er dan zo uit `While((Get-Process outlook) -ne $null)`. Merk op dat t.o.v. de *Watchdog* functie, deze while lus blijft gaan zolang er wél een instantie actief is.
Vervolgens zal er in de `While` lus steeds een `if`-controle gedaan worden naar mails van de *attacker*. Omdat de Outlook rule, die het script heeft aangemaakt, ervoor zorgt dat alle mails van de *attacker* meteen naar de *junk folder* verplaatst worden, zal de *Control* functie nagaan of er mails aanwezig zijn in die *junk folder*.

Wanneer er mails, van de *attacker*, ontvangen worden, zal de Outlook rule, die het script aangemaakt heeft, deze mails rechtstreeks in de *junk folder* plaatsen. Van zodra die mails in de *junk folder* zitten, zal de *control* functie deze detecteren.
Wanneer er mails in de *junk folder* gedetecteerd worden, zal de *Work* functie opgeroepen worden om de mails te verwerken. Nadat de mails verwerkt zijn zal het script terug in de `While` lus van de *Control* functie komen zolang er een instantie van de Outlook applicatie actief is op het systeem.

Zodra de gebruiker de instantie van de Outlook applicatie sluit, zal het script van zijn actieve stand terug naar zijn passieve staat gaan. M.a.w. verlaat het script de *Control* functie en wordt de *Watchdog* functie terug opgeroepen.

Het geheel van Watchdog, Setup en Control zal zich blijven herhalen zolang het systeem (PC) van de gebruiker actief is. Omdat dit in dit project enkel gefocust wordt op het C&C-kanaal en niet zo zeer de *persistency* noch de *delivery* van de malware, is dit script niet *persistent* gemaakt. Hierdoor zal het slecht eenmalig blijven werken zolang het systeem actief is, van zodra de gebruiker het systeem afsluit of heropstart zal het script niet meer actief zijn noch heractiveren.

Work

De *Work* functie zal de inkomende mails van de *attacker* verwerken. Deze functie kan gezien worden als de kern van dit C&C-kanaal. Het zorgt ervoor dat de commando's die de *attacker* wenst uit te voeren, daadwerkelijk uitgevoerd worden op het systeem van het doelwit.
Dit gebeurd door eerst na te gaan of het werkelijk gaat om een mail van de *attacker*. In dien dit het geval is, wordt de inhoud (*body*) van de mail gelezen en overgeschreven naar een *batch* bestand. Dit is een bestand waar DOS-commando's in staan die geïnterpreteerd en uitgevoerd kunnen worden door het Windows *command prompt*.
De resultaten van de commando's worden opgeslagen in een tekst bestand. Vervolgens wordt de *SendResults* functie opgeroepen om de resultaten naar de *attacker* te versturen.
Tenslotte zal deze functie alle sporen van het C&C-kanaal verwijderen namelijk, het batch bestand, het tekst bestand met de resultaten en de mail van de *attacker*.
Nu de mail verwerkt is gaat het script terug naar de *Control* functie tot er weer een nieuwe mail van de *attacker* ontvangen wordt.

Om na te gaan of de mail werkelijk van de *attacker* is, gaat het script het *"SenderAddress"* veld van alle mails, aanwezig in de *junk folder*, uitlezen en vergelijken met het opgegeven adres. Eerst worden alle mails in de *junk folder* in verzameld door `$mails = <folder>.Items` te gebruiken. Het `<folder>` veld stelt de folder voor waaruit we de mails gaan verzamelen, wat in dit geval de *junk folder* is.
Vervolgens wordt voor elk mail-object in deze verzameling gecontroleerd of het afkomstig is van de *attacker*. Dit gebeurd m.b.v. een `foreach` lus. De controle gebeurd a.d.h.v. een `if` met als voorwaarde `$mail.SenderEmailAddress -eq '<attacker's address>'` heeft, wat na gaat of het e-mail adres van de verzender van de mail, gelijk is aan het opgegeven e-mail adres. Zo niet, is die mail niet bedoeld voor dit C&C-kanaal en gaan we die mail negeren. Zo wel, gaan we die mail verwerken.
De eerst stap is het uitlezen van inhoud en die wegschrijven naar een *batch* bestand. De bedoeling is dat de *attacker* het batch bestand schrijft in de *body* van de mail. Op die manier kan het script die *body* wegschrijven (of

"pipen") naar een batch bestand. Dit gebeurd d.m.v. het commando `$mail.body | Out-File <bestandsnaam>.bat -Encoding ASCII`. Zo wordt de waarde van het *body* veld, in ASCII tekens, geschreven naar een batch bestand met de naam `<bestandsnaam.bat`.

Zodra de mail gelezen is, zal het script de mail onmiddelijk verwijderen zodat deze zeker niet zichtbaar zal zijn voor de gebruiker van het systeem. Dit is mogelijk door gebruik te maken van `Delete()` methode van het mailobject als volgt `$mail.Delete()`. Dit gebeurt vóór dat het batch bestand uitgevoerd wordt, omdat de commando's in het batch bestand soms van lange duur kunnen zijn, waardoor de mail anders overbodig lang aanwezig zou blijven. Hoe langer de mail aanwezig blijft, hoe groter de kans dat die gevonden wordt en het kanaal achterhaald wordt.

Nu het batch bestand compleet is, kan deze uitgevoerd worden d.m.v. het commando `.\<bestandsnaam>.bat`. Door het commando `| Out-File <naam>.txt` achter voorgaand commando te plaatsen, zal enige feedback van de commando's van het batch bestand naar een tekst bestand met naam `<naam>.txt` geschreven (*"gepiped"*) worden. Nadat het batch bestand uitgevoerd is en het tekst bestand met de resultaten daarvan opgesteld is, wordt de *SendResults* functie opgeroepen om die verkregen resultaten naar de *attacker* te versturen.

Eens de resultaten verstuurd zijn, zal de *Work* functie het batch bestand en het tekst bestand met de resultaten verwijderen van het systeem, zodat deze niet meer zichtbaar zijn voor de gebruiker van het systeem en het dus verborgen (*Covert*) blijft. De bestanden kunnen verwijderd worden m.b.v. het `rm <bestand>` commando, waar `<bestand>` zowel het tekst bestand als het batch bestand kan zijn. Tenslotte gaat het script terug over naar de *Control* functie (actieve staat).

SendResults

De *SendResults* functie voorziet de mogelijkheid voor tweezijdige communicatie. Dit wil zeggen dat de *attacker* een mail kan sturen en dat het script de *attacker* zal beantwoorden. Waar de *Work* functie zorgt dat de commando's van de *attacker* uitgevoerd worden en de resultaten van die commando's opslaat in een tekst bestand, zal de *SendResults* functie ervoor zorgen dat die resultaten naar de *attacker* verstuurd worden.
Dit wordt verwezenlijkt a.d.h.v. een nieuw mailobject waaraan het tekstbestand met de resultaten als bijlage toegevoegd wordt. Dat mailobject wordt dan verstuurd naar het opgegeven e-mail adress van de *attacker*.

Als eerste stap zal de *SendResults* functie een nieuw mailobject aanmaken door de `CreateItem()` methode te gebruiken van het Outlook object dat in de *Setup* functie werd aangemaakt. Deze methode vereist een *integer* als parameter. Die *integer* bepaald welk type item gecreëerd wordt. Er is keuze uit volgende lijst:

Value	Item type
0	Mail Item
1	Afspraak Item
2	Contact Item
3	Taak Item
4	Journaal Item
5	Notitie Item
6	Post Item
7	Distributie Lijst Item

In dit geval gaat het script een *Mail item* maken, dus wordt waard *'0'* meegegeven als parameter. Dit commando ziet er als volgt uit `$mail = <Applicatie>.CreateItem(0)`, waar `<applicatie>` het outlook object voorstelt. Vanaf nu stelt `$mail` het mailobject voor waarmee gewerkt wordt in deze functie.

Om de mail te kunnen versturen zullen er enkele eigenschappen ingesteld moeten worden. Zo gaat het script eerst de prioriteit of *"importance"* van de mail instellen van hoge prioriteit. De *Importance* eigenschap kan als volgt worden ingesteld `$mail.Importance = [Microsoft.Office.Interop.Outlook.OlImportance]::<importancelevel>` , waar `<importancelevel>` het niveau insteld. Dit niveau kan drie mogelijk waarden hebben namelijk.

- `olImportanceLow` of `0` voor een lage prioriteit
- `olImportanceNormal` of `1` voor een normale prioriteit of
- `olImportanceHigh` of `2` voor een hoge prioriteit.

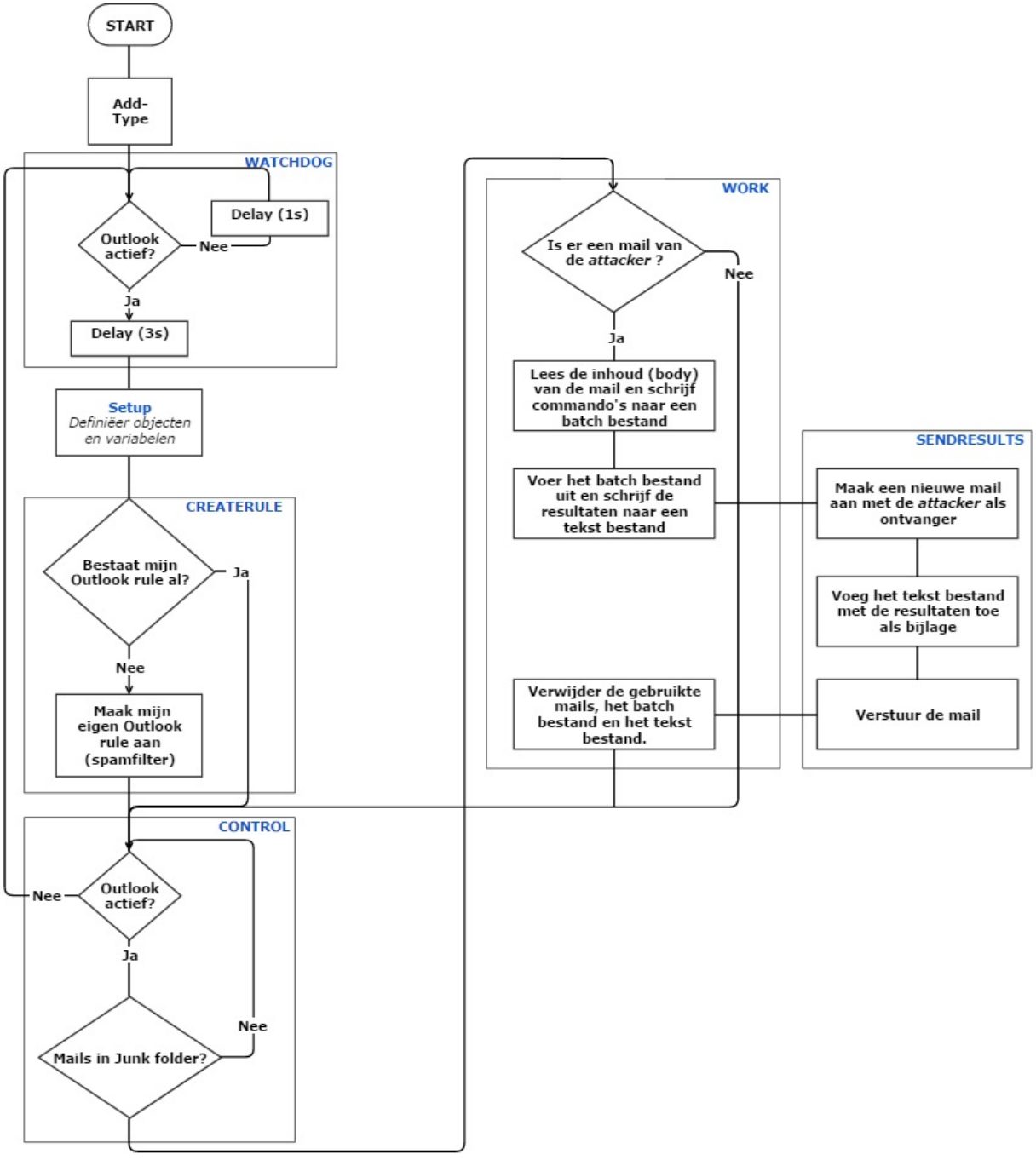
Zoals eerder vermeld beschikken Outlook versies 98, en hoger, over een security patch. Deze security patch beperkt de toegang tot de eigenschappen van de Outlook objecten. In deze functie is er dankzij deze security patch geen toegang tot de eigenschappen die ingesteld moeten worden, zoals bijvoorbeeld de inhoud van de mail, de ontvanger, de bijlage, enzovoort. Daarom gaat het script gebruik maken van mailobjecten van de redemption library. Bij het versturen van mails verwacht de mailserver dat er legitieme mails verstuurd worden. Omdat Redemption een *third party software* is, worden objecten hiervan niet bezien als legitiem. Daarom werd er eerst een legitiem mailobject vanuit het Outlook object aangemaakt. Dit legitiem mailobject kan worden overgenomen in een mail object van de *Redemption Library*. Op die wijze is het mailobject legitiem en wordt er volledige toegang tot de eigenschappen verkregen. Om dit te verwezenlijken bestaat er binnen de *Redemption Library* de `GetRDOObjectFromOutlookObject()` functie. Deze functie zal een *Redemption Library* object aanmaken vanuit een legitiem Outlook object. Als parameter voor deze functie moet het object meegegeven worden dat overgenomen moet worden, in dit geval het mailobject `$mail` .

Nu er toegang is tot alle eigenschappen van een legitiem mailobject, kunnen de vereiste eigenschappen ingesteld worden. Eerst wordt de ontvanger van de mail bepaald door de `To` eigenschap gelijk te stellen aan het e-mail adres van de ontvanger met het commando `<Redemption mail>.To = <mail adres>` . In dit geval zal het ontvanger adres dat van de *attacker* zijn.
Vervolgens wordt het onderwerp van de mail aangepast. Dit wordt verwezenlijkt door de `Subject` eigenschap van de mail een *string* waarde te geven. Ook de inhoud van de mail wordt ingevuld door de `Body` eigenschap, van de mail, gelijk te stellen aan een *string* waarde.
Als aller belangrijkste veld voor dit C&C-kanaal, kan er een bijlage aan de mail worden toegevoegd d.m.v. de `Attachment` eigenschap in te stellen. Deze eigenschap heeft een `Add()` functie die als parameter de *filepath* naar het bestand, dat men wenst toe te voegen, eist. Dat geheel ziet er als volgt uit `<Redemption mail>.Attachment.Add("<filepath>")` .

Tenslotte wordt er de `DeleteAfterSubmit` eigenschap op *true* ingesteld. Wanneer die eigenschap *true* (waar) is, zal het mailobject automatisch verdwijnen uit de mailbox van zodra de mail verzonden is. De mail zal ook niet in de *"verwijderde e-mails"* folder komen, en zal gewoon verdwijnen.
Nu alle eigenschappen ingesteld zijn, zal de mail verstuurd worden m.b.v. zijn `Send()` methode en loopt de functie tot zijn einde. Hierna gaat het script verder met zijn normale verloop.

2.2.3. Flowchart

In dit deel wordt de sequentie van het script weergegeven aan de hand van een *flowchart*. In deze *flowchart* zijn de processen die eenzelfde functie voorstellen omkaderd. In de rechter boven hoek van deze kaders staan telkens de namen van de functies, uit het script, die voorgesteld worden, in het blauw.
Merk op dat het programma geen *"STOP"* onderdeel bevat. Dit is omdat het programma continu blijft werken zolang het systeem actief is.



Figuur 4: Flowchart van het Outlook C&C script

2.2.4. Problemen & Oplossingen

Outlook Session

Om gebruik te maken van Outlook objecten moet er binnen een bepaalde *session* (sessie) worden gewerkt. Wanneer er buiten een sessie wordt gewerkt, is er geen toegang tot folder- en/of mail-objecten.
Om in een sessie van Outlook te werken, wordt het commando `session.logon()` van het Outlook object gebruikt. Dit zorgt ervoor dat er, ofwel in een actieve sessie *ingelogd* wordt, ofwel een nieuwe sessie aangemaakt wordt.
Wanneer er een nieuwe sessie aangemaakt wordt, opent er een venster waarin nagevraagd wordt welke sessie de gebruiker wenst aan te maken. Dit is nadelig voor dit C&C-kanaal, omdat het zeer opvallend is.
Omdat het de bedoeling is dat er niets interactief gebeurd, en alles verborgen blijft. Wordt er in dit C&C gebruik gemaakt van de reeds actieve sessie van de gebruiker van het systeem. Op deze wijze is er toegang tot alle folders en mails van de gebruiker en openen er zich geen vensters, om dat de sessie al bestaat.
Nu er in een sessie gewerkt kan worden, is er toegang tot alle objecten van de gebruiker van die sessie.

Outlook Security Patch

Tijdens de ontwikkeling van dit C&C-kanaal werden beperkingen ondervonden vanwege de reeds besproken *security patch* voor Outlook versies 98 en hoger. Deze *security patch* en de oplossing hiervoor zijn reeds besproken in het deel Powershell.

Verzend vertraging

Tijdens het opstellen van de C&C werd een probleem ondervonden bij het verzenden van mails via Powershell Outlook objecten. Wanneer het script mails verstuurde naar de *attacker*, duurde het minstens één minuut vooraleer de mail verstuurd was. Tijdens dit process was deze mail zichtbaar in de *outbox* folder.
Omdat het de bedoeling is dat de gebruiker van het systeem, en de Outlook applicatie, niets mag zien van wat het script doet, is het niet optimaal dat de mail zichtbaar is in de *outbox* voor deze lange tijd. Hoe langer de mail zichtbaar is in de *outbox*, hoe groter de kans dat de gebruiker deze mail ziet en onderschept, hoe groter de kans dat het C&C-kanaal ontdekt en geblokkeerd wordt. Daarom is het cruciaal dat de zendtijd zo kort mogelijk is.
Om dit te verwezenlijken wordt de prioriteit van de mail op "hoog" ingesteld. Omdat de prioriteit nu hoog is, zal de mail voorrang krijgen en zo snel mogelijk worden verwerkt (verzonden). Nu zal de gebruiker geen mails meer zien in zijn/haar outbox en blijft het C&C-kanaal veilig verborgen voor ogen.

2.3. TOR C&C

Het tweede C&C-kanaal is gebaseerd op het TOR-netwerk. Zoals eerder besproken biedt TOR totale anonimiteit op het Internet voor gebruikers en/of services. Dit zorgt ervoor dat het C&C-kanaal anoniem is, wat wil zeggen dat niet achterhaalt kan worden met wie gecommuniceerd wordt via het kanaal. Door deze anonimiteit blijft de *attacker* onbekend in geval dat het C&C kanaal achterhaald wordt.

De bedoeling van dit kanaal is dat een *attacker*, van het EY ITRA-team, tijdens een redteam test, *remote-access* verkrijgt tot een systeem dat zich binnen een bedrijfsnetwerk, dat beschermd wordt door NPD, bevindt. Dit gebeurd dan op anonieme wijze zodat het C&C-kanaal en de *attacker* minder kans maakt achterhaald te worden.

Voor de verwezenlijking van dit C&C-kanaal werd gebruik gemaakt van de volgende technologien:

- Microsoft Visual Studio (MSVS)
  - C#
  - Visual Basic for Applications (VBA)
- Powershell (script)
- MinGW
- The Onion Router (TOR)

2.3.1. Werking

De principiële werking van dit kanaal is om op een systeem een TOR *hidden service* opstellen waarmee de *attacker* kan verbinden om zo met het systeem van het doelwit te communiceren op volledig anonieme wijze.

De uitvoering van dit C&C kanaal gebeurd a.d.h.v. malware geschreven in C#. Deze malware zal a.d.h.v een word bestand met *dropper macro* verspreid worden. *Dropper Macro's* zijn kleine programma's die software kunnen injecteren (*"laten vallen" = drop*) in een systeem. Voor dit C&C kanaal werdt gebruik gemaakt van een Word macro, terwijl dit ook mogelijk is voor Excel bestanden en andere Microsoft Office applicaties.
Voor dit C&C-kanaal is een Powershell script geschreven dat automatisch een dropper macro zal genereren. Aan de hand van dat script wordt de malware geëncodeerd en weggeschreven in variabelen van een Word bestand. Vervolgens kan de *attacker* deze gegenereerde *macro* aan een Word bestand toevoegen en dit Word bestand versturen naar zijn/haar doelwit.
Zodra het doelwit dit Word bestand opent, zal de macro automatisch het malware programma wegschrijven naar een verborgen folder in het systeem van het doelwit. Vervolgens zal de macro dat malware programma voor de eerste keer uitvoeren. Weet dat de malware werkt met dezelfde gebruikersrechten als die van de gebruiker van het systeem.

De malware zal eerst controleren of er reeds een *Windows Registry Key* voor zichzelf bestaat. *Windows Registry* is een interne database dat *low-level* instellingen van het systeem opslaat. Zo bestaat er een tabel dat bepaald welke programma's opstarten zodra het systeem opstart of een gebruiker inlogt. Daar zal de malware nagaan of het geregistreerd staat, zo niet zal de malware een niewe *Registry Key* aanmaken voor zichzelf in deze tabel. Dit zorgt ervoor dat de malware steeds zal opstarten van zodra het systeem actief is (=*persistency*).

Nu de malware *persistant* is zal worden nagegaan of de TOR *client* aanwezig is op het systeem, zo niet zal de malware deze downloaden van een *host* server en verbergen in het systeem. Omdat de TOR *client* groot is en veel *.dll* files meebrengt zal dit rommelig zijn en moeilijk om te verbergen, wat niet zo compact is.
Omwille van compactheid en eenvoudigheid werd er voor dit project een *static binary* van de TOR *client* gecompileerd. Een *static binary* is een *binary* file waarin alle benodigde *dll* files e.d. in éénzelfde *binary* inbegrepen worden. Zo is de volledige TOR *client* met al zijn benodigdheden tot één compact `.exe` bestand gevormd. Dit maakt het eenvoudiger om de file te downloaden en te verbergen.

Nadat de TOR *client* gedownload is en verborgen werd door de malware, zal de malware een configuratie file aanmaken voor de TOR *hidden service*. Deze file bepaald welke TOR *bridges* gebruikt moeten worden, naar welke poorten van de *hidden service* geluisterd moet worden en naar welke lokale poort het verkeer op die poorten van de *hidden service* doorgestuurd moet worden.
Vervolgens zal de malware de *hidden service* actief maken door de TOR-*client* (.exe) uit te voeren met de configuraties van de configuratie file.

Zodra de *hidden service* actief is, wordt de configuratie file verwijderd en zal de *hostname* file uitgelezen worden. Deze file bevat het *.onion* adres van de *hidden service*. Wanneer het adres gelezen is wordt het *".onion"* gedeelte van het adres verwijdert en wordt het domein van de DNS server van de *attacker* er aan toegevoegd. In geval van het EY ITRA-team zal dit bijvoorbeeld *"ey.be"* zijn.
Vb.: *hidden service* adres: "ayr7h93ke7nh6bv0**.onion**" wordt dan "ayr7h93ke7nh6bv0**.ey.be**".
Vervolgens zal er een *DNS-query* gedaan worden naar het adres met het domein van de *attacker* ("ayr7h93ke7nh6bv0**.ey.be**"). Wanneer die *DNS-query* gebeurt, zal die automatisch naar de DNS-server van de *attacker* gestuurd worden. Daar kan de *attacker* zien dat deze request van zijn doelwit afkomstig is en kan hij/zij het adres van de *hidden service* achterhalen. Dit kan door simpelweg zijn/haar domein van dat adress te verwijderen en er terug het *".onion"* adres aan toe te voegen.
Vb.: *query* geeft aan: "ayr7h93ke7nh6bv0**.ey.be**", dit wordt dan terug "ayr7h93ke7nh6bv0**.onion**".

Nu de *attacker* het *.onion* adres van de *hidden service* kent, moet hij/zij slechts op correcte wijze met dit *.onion* adres connecteren om toegang tot het systeem te verkrijgen. Het type communicatie/connectie is afhankelijk van het protocol (lees: poort) dat luistert naar de *hidden service*. Dit protocol wordt bepaald in de configuratie file voor de hidden service.

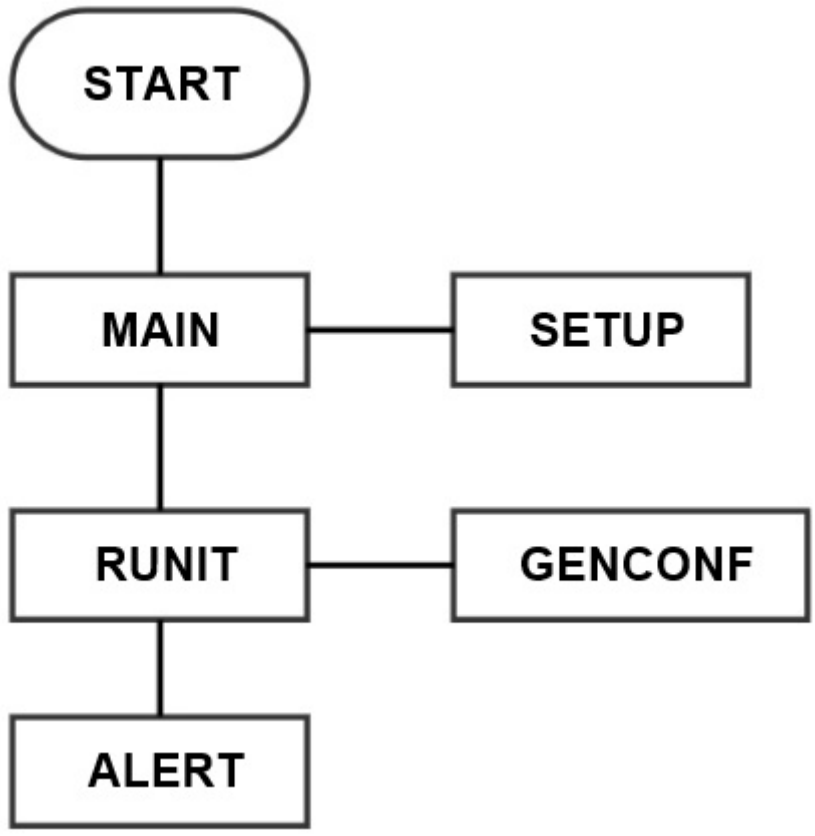
2.3.2. Code

Voor het gehele pakket van dit C&C-kanaal werden verschillende programma's geschreven. Zo is er het hoofd programma, de malware. Dit is het kern programma essentieël voor dit C&C-kanaal en werd in de programmeertaal C# geschreven m.b.v. de Microsoft Visual Studio (MSVS) IDE.
Het tweede deel is de *dropper macro* die ervoor zal zorgen dat de malware zich, vanuit een Word bestand, nestelt in het systeem van het doelwit. Deze *macro* werd geschreven in de programmeertaal Visual Basic for Applications (VBA) en wordt gegenereerd a.d.h.v. een Powershell script.
Dit powershell script is dan ook het derde en laatste programma voor dit C&C-kanaal. Het zorgt ervoor dat het kern-programma (dé malware) wordt geëncodeerd met het *Base64* algoritme. Vervolgens zal het die Base64 geëncodeerde *string* opsplitsen per 900 karakters. Deze gesplitste *strings* worden dan in de macro geschreven als omgevingsvariabelen van een Word bestand.
In dit onderdeel wordt elk programma apart besproken en zal de code uitgelegd worden a.d.h.v. pseudo-code.

2.3.2.1. De C# malware

Zoals reeds vermeld is de malware het hoofdbestanddeel van dit C&C-kanaal. Het is dit programma dat het kanaal openstelt. Dit programma werd in de C# programmeertaal geschreven m.b.v. de Microsoft Visual Studio (MVS) IDE. Hier wordt de functionaliteit besproken en de werking van de code wordt aangehaald. Dit programma beschikt over enkele verschillende functies met elk hun eigen doel. De functies in dit programma zijn:

- Main
- Setup
- Run
- GenConf
- Alert
- ExecuteProcess



Main

De Main functie is de begin functie van het programma. Elk C# programma heeft een Main functie. Deze functie is een entry point, wat wil zeggen dat wanneer het programma uitgevoerd wordt, zal alles wat zich in deze functie voordoet gebeuren.

In deze functie gebeuren twee controles nodig voor de start van het programma. Zo zal eerst worden gecontroleerd of het malware programma *persistant* is, dit wil zeggen dat het steeds zal werken als het systeem aanstaat. Wanneer de gebruiker van het systeem, het systeem uitzet stopt uiteraard het malware programma. Als het malware programma *persistant* is, zal het opstarten zodra het systeem terug actief is.
Voor *persistancy* van deze malware is gekozen voor een *Windows Registry Key* in de Autorun map. Die key zal ervoor zorgen dat wanneer het systeem opstart, de malware wordt opgestart.

Eerst zal dus nagekeken worden of de *Registry Key* voor het malware programma al bestaat. Dit is in C# mogelijk met de `Microsoft.Win32` library die toegang biedt tot de `Registry` objecten. Zo kan met de `GetValue()` functie, van die objecten, gezocht worden naar een bepaalde *Registry Key Value*. Als deze functie `null` teruggeeft, wil dit zeggen dat de gezochte *Registry Key Value* nog niet bestaat.
In dit geval zal het malware programma zelf een *Registry Key Value* schrijven voor zichzelf. Dit is dan mogelijk door de `SetValue()` functie van de `Registry` objecten.

Nu de *persistancy* een feit is, kan het programma beginnen aan het opstellen van het C&C-kanaal. Eerst moet nog gecontroleerd worden of alle nodige bestanden aanwezig zijn. Er wordt dus gecontroleerd of de TOR *client* aanwezig is m.b.v. de `if(File.Exists(<bestand>))` voorwaarde. Het `<bestand>` veld zal in dit geval de *static binary* van de TOR *client* (*"tor.exe"*) zijn.
Indien deze bestand niet aanwezig is, zal eerst de *Setup* functie opgeroepen worden om alle benodigdheden te voorzien. Als het TOR *client* bestand reeds aanwezig is, zal de *Run* functie opgeroepen worden om het C&C-Kanaal op te stellen.

Setup

De setup functie zal alle benodigde bestanden gaan downloaden en verbergen in het systeem van de gebruiker. Eerst wordt er een tijdelijk `WebClient` object aangemaakt waarvan de `DownloadFile()` functie het downloaden van het bestand zal afhandelen. De `DownloadFile()` functie verwacht twee parameters namelijk, de url vanwaar het bestand gedownload moet worden en een locatie in het systeem waar dat gedownloade bestand opgeslagen zal worden (vb: "C:\Users\gebruiker\Bureaublad").
Wanneer het bestand succesvol gedownload en opgeslagen is, zal de *ExecuteProcess* functie opgeroepen worden met als parameter het commando om het bestand verborgen (*hidden*) te maken. Elk bestand heeft een eigenschap "verborgen" of *"hidden"* die met het commando `attrib +h <bestand>` als waar wordt ingesteld. Dit zorgt ervoor dat het bestand niet zichtbaar is voor de gebruiker van het systeem, tenzij de gebruiker heeft ingesteld om verborgen mappen te weergeven.
Wanneer de map verborgen is, wordt de *Run* functie opgeroepen om de *hidden service* op te starten.

Run

De *Run* functie zorgt voor het opstarten van de *TOR Hidden Service* wat het hart van dit C&C-kanaal is. Om de *hidden service* te kunnen opstarten, is er nood aan een configuratie bestand met alle instellingen voor de *hidden service*. Om die rede zal in deze functie eerst gecontroleerd worden of die configuratie bestand aanwezig is, zo niet zal de *GenConf* functie opgeroepen worden, die die nodige configuratie bestand zal genereren.
Wanneer het configuratie bestand aanwezig is, kan de *hidden service* opgestart worden. Dit gebeurd a.d.h.v. de *ExecuteProcess* functie die het commando `powershell.exe -WindowStyle Hidden -Command "<tor> -f <Config bestand>"` zal verwerken dat die service opstart. In dit commando zal `Powershell.exe` ervoor zorgen dat er een powershell commando zal uitgevoerd worden. `WindowStyle Hidden` bepaald dat, wanneer het powershell commando uitgevoerd wordt, het powershell venster verborgen blijft. De *command* parameter wordt dan bepaald met de code `-Command "`

`<tor> -f <config bestand>"` . Het commando dat uitgevoerd wordt, is dan alles wat tussen accolades staat, in dit geval `<tor> -f <config bestand>` met het *tor.exe* bestand als `<tor>` en het config bestand in de plaats van `<config>` . De `-f` zal de mogelijkheid bieden om argumenten mee te geven, namelijk die uit het *config* bestand.

Omdat de *hidden service* enkele tijd nodig heeft om op te starten, is er een *delay* van vijf seconden aanwezig die het programma even stilzet tot de *hidden service* volledig opgesteld is. De duur van het opstarten is afhankelijk van hoe snel de route binnen het TOR netwerk vastgelegd wordt, m.a.w hoe snel alle nodes gevonden en gecontacteerd worden.
Tenslotte, wanneer de *hidden service* opgestart is, wordt de *Alert* functie opgeroepen om het adres van de *hidden service* met de *attacker* mee te delen.

GenConf

De *GenConf* functie zorgt voor het aanmaken/genereren van het configuratie bestand, nodig voor het opstellen van de *hidden service*. Dit bestand beschikt over alle configuraties voor de hidden service. Zo bepaald het of er TOR *bridges* gebruikt worden, zo ja, welke. Het bevat informatie over welke poort gebruikt wordt voor TOR trafiek, waar de folder met informatie over de *hidden service* komt te staan. Het configuratie bestand bepaalt ook welke poorten van de *hidden service* open staan. Indien er trafiek op een poort van de *hidden service* binnenkomt moet bepaald worden naar welke lokale poort, van het systeem, dit trafiek geleid moet worden. Bijvoorbeeld wanneer de *hidden service* trafiek krijgt op poort 2222 moet deze trafiek naar poort 22 van het lokale systeem van de gebruiker. Indien er dan een proces luistert op die poort van het systeem, kan met dat proces gecommuniceerd worden via de TOR *hidden service*.

Alle nodige configuraties worden dan in een array geschreven. Vervolgens wordt deze array naar een bestand geschreven d.m.v. de functie `File.WriteAllLines(<bestand>,<configuraties>)` , waar `<bestand>` het bestand voorstelt waar naar geschreven wordt, en `<configuratie>` is de array met configuraties die naar het bestand geschreven wordt. Tenslotte wordt de *ExecuteProcess* functie opgeroepen om het commando af te handelen dat de config bestand verborgen zal maken.

Alert

De alert functie zal de *attacker* waarschuwen dat de *hidden service* actief is en zal ook het adres van de *service* meedelen. Zo weet de *attacker* dat de *hidden service* bereikbaar is, en op welk adres deze bereikbaar is. Deze waarschuwing gebeurd a.d.h.v. een DNS-query. DNS-queries kunnen in C# verwezenlijkt worden door gebruik van de `System.Net.Dns` objecten. Om deze objecten te gebruiken moet deze library beschikbaar zijn op het systeem, wat niet altijd het geval is waardoor er dan ook geen DNS-query kan gedaan worden. Om die reden werd het commando `nslookup` verwerkt met de *ExecuteProcess* functie. Het `nslookup` commando is standaard beschikbaar voor Windows gebruikers, wat zekerheid biedt dat de DNS-query uitgevoerd kan worden.

Het adres dat *gequeried* wordt, is een knipsel van het adres van de *hidden service* en het domein van de DNS-server van de *attacker*. Eerst zal het bestand, waar het adres van de *hidden service* opgeslagen is, uitgelezen worden en opgeslagen in een *string* variabele. Dat adres is een combinatie van karakters, afgeleid uit de *public key* van de *hidden service*, gevolgd door het *".onion"* domein (vb.: "ayr7h93ke7nh6bv0**.onion**") .
Vervolgens wordt deze *string* gescheiden van het *".onion"* domein m.b.v. de `Split()` functie. Het resultaat is dan slechts de combinatie van karakters (vb.: "ayr7h93ke7nh6bv0").
Hierna wordt het domein van de DNS-server van de *attacker* (vb: *ey.be*) aan die restant toegevoegd (vb.: "ayr7h93ke7nh6bv0**.ey.be**"). Dit verzekerd dat de DNS-query op de DNS-server van de *attacker* terechtkomt, zodat hij/zij de query zal ontvangen en het adres er uit kan afleiden. Tenslotte zal het `nslookup` commando, gevolgd door het adres met het domein van de DNS-server van de *attacker*, verwerkt door de *ExecuteProcess* functie. Nu alles opgestart is en de *attacker* verwittigd is, kan het config bestand verwijderd worden zodat er zo weinig mogelijk sporen op het systeem aanwezig zijn die argwaan kunnen opwekken bij de gebruiker van het systeem. Zo blijft het C&C-kanaal geheim en is er minder kans dat het achterhaald wordt.

ExecuteProcess

Deze functie is een zelfgeschreven functie die het mogelijk maak om, m.b.v. `Process` objecten, DOS commandos uit te voeren. De functie verwacht twee parameters. De eerste parameter is een *string* variabele met het DOS-commando dat uitgevoerd moet worden. De andere parameter is een *boolean* die bepaald of het commando een continu lopend proces is of een eenmalig commando'. Als er dan een eenmalig commando zoals bijvoorbeeld `attrib +h` uitgevoerd wordt, zal, indien er iets foutloopt, na bepaalde tijd het proces gestopt worden. Wanneer een service, zoals TOR, wenst op te starten met het DOS-commando, moet dit continu actief blijven en niet afgesloten worden. Om dit onderscheid te maken is die *boolean* aanwezig.

Eerst zal het commando opgevangen worden en gecontroleerd worden of het argumenten bevat, zo ja wordt het commando omgezet naar binaire data met alle parameters.
Vervolgens wordt een `Process` object aangemaakt waarvan verschillende eigenschappen/configuraties voor de `Start()` methode worden ingesteld. Zo wordt de `FileName` eigenschap ingesteld op het commando die bepaald welk document/applicatie opgestart zal worden, in dit geval is dit het commando dat als parameter meegegeven werd.
Om alles zo verborgen mogelijk te houden en interactie met de gebruiker van het systeem te vermijden zijn er enkele eigenschappen ingesteld die hiervoor cruciaal zijn. Zo is de `CreateNoWindow` eigenschap als `true` ingesteld zodat er géén venster geopend wordt bij het uitvoeren van het proces. De `ErrorDialog` eigenschap wordt op `false` ingesteld zodat er geen interactie is met de gebruiker indien er iets misloopt. Cruciaal is dat de `WindowStyle` eigenschap als `Hidden` wordt ingesteld zodat tijdens het proces vensters verborgen blijven.
Tenslotte wordt ook nog de `UseShellExecute` eigenschap op `false` ingesteld om te vermeiden dat de shell van het besturingssysteem gebruikt wordt om het proces op te starten.

Nu alle starteigenschappen ingesteld zijn, kan het process gestart worden d.m.v. de `Start()` methode van het `Process` object. Vervolgens wordt gecontroleerd of het een eenmalig commando is of een continu proces. Indien het een eenmalig commando is zal er 10 seconden gewacht worden op het aflopen van het proces d.m.v. de `WaitForExit(<MilliSec>)` functie. Indien het na die 10 seconden nog niet afgehandeld is, wordt het proces afgesloten met de `Kill()` methode.
Indien het een continu proces is, en geen eenmalig commando, zal het proces niet afgesloten worden. Zo zal, in geval van deze malware, de TOR *hidden service* actief blijven.

2.3.2.2. Macro Generator

De macro generator is een script geschreven in Powershell m.b.v. de Powershell Integrated Scripting Environment (ISE) van Windows. Dit script zorgt ervoor dat de *dropper macro* voor een bestand gegenereerd wordt. Om die reden verwacht het script één parameter namelijk het bestand waarvoor een *dropper macro* gegenereerd moet worden. Het script zal eerst controleren of er werkelijk een bestand als parameter werd meegegeven, en ook of dit een geldige bestand is. Zo niet geeft het script een foutmelding. Indien het wel een geldige bestand is wordt de macro gegenereerd. Dit script bevat 3 functies nodig voor de generatie van de macro namelijk:

- EncodeBase64
- Split
- MacroGen

EncodeBase64

De *EncodeBase64* functie heeft al doel het encoderen van het gekozen bestand met Base64. Base64 is een algoritme gebruikt om binaire data te converteren naar ASCII-tekens. In dit geval zal de binaire data van het gekozen bestand geconverteerd worden naar ASCII-tekens m.b.v. het Base64 encoderingsalgoritme. Deze geëncodeerde data wordt opgeslagen in een *string* variabele.

Eerst zal de binaire data uitgelezen worden d.m.v. het commando `Get-Content -Path <bestand> -Encoding Byte`. Het `Get-Content` verwacht twee parameters waarvan de eerste de `-Path` parameter is die de *filepath* vraagt van het bestand die uitgelezen moet worden. Deze *filepath* is de parameter die gegeven moet worden wanneer het script opgeroepen wordt.

De tweede parameter is de `-Encoding` parameter. Deze parameter bepaald welke encodering gebruik is voor de inhoud van het bestand. Omdat de binaire data gelezen wordt, zal deze parameter ingesteld worden op `Binary`. Eens de binaire data van het bestand gelezen is wordt deze in een *string* variabele opgeslagen.

Vervolgens wordt de *string* variabele met de binaire data geëncodeerd met het Base64 algoritme. Dit gebeurd m.b.v. de functie `[Convert]::ToBase64String(<data>)` waar `ToBase64String` bepaald met welk type encodering geconverteerd wordt. Deze functie verwacht als parameter de data die geconverteerd moet worden, wat in dit geval de binaire data van het gekozen bestand is. Die binaire data werd reeds opgeslagen in een *string* variabele. Daarom wordt deze *string* variabele meegegeven als parameter voor de `[Convert]` functie. Deze geëncodeerde data wordt ook opgeslagen in een *string* variabele.

Tenslotte zal de *Split* functie opgeroepen worden die een *string* variabele als parameter verwacht. Omdat de reeds geconverteerde data gesplitst moet worden, wordt de *string* variabele, waarin die geconverteerde data opgeslagen werd, meegegeven als parameter voor de *Split* functie. De *return* waarde die teruggegeven wordt door de *Split* functie wordt dan ook ingesteld als *return* waarde van deze *EncodeBase64* functie.

Split

Omdat binaire data van een bestand zeer veel tekens bevat, en een Base64 encodering dit aantal verdubbeld, moet die Base64 geëncodeerde data, afkomstig van de *EncodeBase64* functie, gesplitst worden zodat die data in verschillende variabelen van een Word bestand geplaatst kunnen worden.
Eerst wordt de *string* met Base64 data, meegegeven aan de functie, omgezet naar een *array* van karakters d.m.v. het commando `$Array = <Base64data>.ToCharArray()`.
Vervolgens wordt er door de voorgaande *array geloopt* a.d.h.v. een `foreach` loop die zal herhalen voor elk karakter in de *array*. Telken de lus herhaald wordt, wordt eerst nagekeken of er al 900 tekens gepasseerd zijn. Dit gebeurd a.d.h.v. een `if` functie in combinatie met een teller variabele. Indien er nog geen 900 tekens gepasseerd zijn, zal het huidige teken opgeslagen worden in een *string* variabele, die dient als een soort *buffer*, en wordt de teller met één verhoogt.
Indien de teller gelijk is aan 899, dit geldt elk 900ste kakrakter, wordt het huide karakter ook in de *buffer string* opgeslagen en de gehele waarde van die *buffer string* opgeslagen worden naar een index van een array. Hierna worden de *buffer* en de teller leeg gemaakt/*gereset*. Zo wordt de gehele *array* opgesplitst per 900 karakters. Die gesplitste waarden worden, per 900 karakters, in een nieuwe *array* opgeslagen.

Nadat de `foreach` lus ten einde is gelopen, zullen de laatste waarden in de *array* worden opgeslagen en zal deze *array* als *return* waarde voor deze functie gegeven worden.

MacroGen

De *MacroGen* functie zal de VBAmacro genereren naar een tekst bestand. Dit gebeurd door VBAcommando's als een *string* te schrijven naar een tekst bestand. De reden dat de VBAmacro gegenereerd wordt i.p.v. deze manueel te schijven in een IDE die hiervoor dient, is omdat de Base64 string in dit script gegenereerd wordt en deze variabele naar de VBAmacro geschreven moet worden. Binaire data encoderen met het Base64 algoritme is zeer omslachtig in VBAcode, daarom wordt dit op eenvoudige wijze opgelost met een script.
De code van de Macro wordt dus in feite gewoon als een *string* variabele weggeschreven naar een tekstbestand. Vervolgens kan de inhoud van dit tekstbestand gekopieerd worden naar een Word macro IDE zodat de code betekenis heeft. De code en werking van de macro zelf wordt in een apart deel behandeld.

2.3.2.3. VBA Dropper Macro

De VBA*dropper* macro zorgt ervoor dat wanneer een gebruiker het Word bestand opent, de malware automatisch geïnstalleerd wordt op zijn/haar systeem. Eerst moet de *attacker* de macro in de Word bestand laden. Dit gebeurd door de inhoud van het tekst bestand, gegenereerd door de Macro generator, te kopiëren naar de macro programmeer omgeving van het Word bestand. Deze omgeving is toegankelijk via de *shortkey* toetsencombinatie `Alt + F11` in een Word bestand.

Vóór dat het bestand verzonden wordt naar een doelwit, moet het bestand dat *gedropt* moet worden, ingeladen worden. Dit bestand werd eerder geëncodeerd met het Base64 algoritme en vervolgens werd het resultaat opgesplitst per 900 karakters. Om dit bestand in te laden, worden die gesplitste *string* variabelen, van elk 900 karakters, in een omgevingsvariabele van de Word applicatie ingeladen. Een voorbeeld van die variabelen zijn de variabelen van een *charset*. Zo bestaat de `charset_be_latin` die ingesteld kunnen worden met een van die gesplitste *string* variabelen.
Het inladen van deze variabelen gebeurt a.d.h.v. een functie `DefineVariables` die ervoor zorgt dat elk deel van de gesplitste Base64 *string* in het Word bestand terecht komt. Deze functie moet verwijderd worden vóóraleer het bestand naar het doelwit verstuurd wordt. Indien die functie niet uit de macro verwijderd wordt, is het gemakkelijker voor een malware analist om de malware te achterhalen en/of te analyseren.

In deze macro werd gebruik gemaakt van twee standaard functies van de Microsoft Office Macro's. Deze functie zijn de `Document_Open` functie die opgeroepen wordt telkens het document geopend wordt. In meeste versies van Microsoft Office applicaties, zoals Word, worden macro's geblokkeerd vanwegen beveiligingsrisico's. Door deze beveiligingsmaatregelen zal de `Document_Open` functie vaak geblokkeerd worden.
Telkens er een macro geblokkeerd wordt, zal de gebruiker een waarschuwing krijgen dat er macro's aanwezig zijn, en wordt de mogelijkheid aangeboden om de macro's toch toe te staan en de blokkering op te heffen. Door gebruik te maken van de `Auto_Open` functie zal, telkens de gebruiker de macro's toestaat (deblokkeerd), deze functie uitgevoerd worden. Zo zal de macro in elk scenario uitgevoerd kunnen worden.
Wanneer de `Document_Open` of de `Auto_Open` functies opgeroepen worden zullen die functies beiden op hun beurt de `Drop` functie oproepen.

Drop

De drop functie zal de eerder ingeladen *string* variabelen terug uit de omgevingsvariabelen halen en samenbundelen in een *array*. Dit gebeurd a.d.h.v. een `For Each` lus die door de de variabelen van het Word bestand zal lussen. Voor elke variabele dat de lus tegenkomt, zal de waarde van die variabele in de *array* opgeslagen worden en vervolgens wordt die omgevingsvariabele verwijderd. Dit gebeurd met elke variabele die eerder ingeladen werd.
Vervolgens wordt de *array* van de aparte *strings* samengeplakt tot eenzelfde *string*. Dit gebeurd m.b.v. een `For Each` die elk deel in de *array* aan eenzelfde *string* plakt door het commando `geheel = geheel + arraydeel` .

Nu de variabelen opgeslagen zijn in één *string*, is het tijd om het bestand, dat eerder ingeladen werd om te *droppen*, terug te reconstrueren en weg te steken in het systeem van het doelwit. Eerst wordt er een *Shell* object aangemaakt met het commando `set S = CreateObject("Wscript.Shell")` , wat het mogelijk maakt DOS-commando's uit te voeren en omgevingsvariabelen van het systeem aan te spreken. Er wordt ook een object van het *FileSystem* aangemaakt d.m.v. het commando `set FS = CreateObject("Scripting.FileSystemObject")` , wat zorgt voor toegang tot het lezen/schrijven/maken van folders en/of bestanden.

Vervolgens wordt m.b.v. het eerder geïnitialiseerde *Shell* object een folder aangemaakt op een gekozen locatie in het systeem. In geval van dit project werd de folder aangemaakt in de *AppData* folder. Eerst wordt het pad gedefiniëerd waar de folder komt te staan m.b.v. commando `Folder = S.ExpandsEnvironmentStrings("%APPDATA%") &"\<folder>")` waar `<folder>` de naam van de folder voorstelt. Daarna kan de folder effectief aangemaakt worden m.b.v. het Powershell commando `md Folder` dat d.m.v. het `Shell` commando kan worden uitgevoerd . Eens de folder aangemaakt is, wordt deze als "verborgen" ingesteld d.m.v. het Powershell commando `attrib +h <folder>` . In VBAis dit mogelijk a.d.h.v. commando `Shell "powershell.exe -Command "attrib +h <folder>"` . Omdat het even duurt om de folder aan te maken, wordt de `Wait` functie opgeroepen. Deze dient als een soort *delay* dat een parameter verwacht voor het aantal seconden dat er gewacht moet worden.

Na de delay zal de folder aangemaakt zijn en kan er in deze folder worden gewerkt. In deze folder wordt dan een tijdelijk bestand geplaatst waar de inhoud van de eerder opgestelde *string* naartoe zal worden geschreven. Vervolgens wordt er een `Open Text File` object aangemaakt met als doelbestand het net aangemaakte tijdelijke bestand `<Tijdelijk bestand>` . Daarna wordt de *string* naar dat tijdelijk bestand geschreven met het commando `OTF.WriteLine geheel` .

Tenslotte wordt er ook nog een pad gedefiniëerd waar het uiteindelijk resultaat en dus werkelijke bestand terecht moet komen. Vervolgens wordt het *Shell* commando `Shell "powershell.exe -WindowStyle Hidden -Command " [System.Convert]::FromBase64String([System.IO.File]::ReadAllText(<Tijdelijke bestand>)) | Set-Content -Path "<Resultaat bestand>" -Encoding Byte; attrib +H " <Resultaat bestand> "; rm " <Tijdelijke bestand> "; " <Resultaat bestand> "} "", 0` uitgevoerd.
Het eerste deel zorgt ervoor dat de inhoud van het tijdelijke bestand uitgelezen wordt. Dit bestand bevat de Base64 geëncodeerde data van het bestand dat eerder geëncodeerd werd. Dit gebeurt m.b.v. `[System.IO.File]::ReadAllText(<Tijdelijk bestand>)`. De waarde die uitgelezen wordt, zal dan terug geconverteerd worden a.d.h.v. het `[System.Convert]::FromBase64String()` commando. Deze conversie wordt weggeschreven naar een permanent bestand, wat hetzelfde bestand zal zijn als het oorspronkelijke bestand, m.b.v. het `| Set-Content -Path " & Result & " -Encoding Byte` commando, waar `-Path` bepaald naar welk bestand de inhoud wordt geschreven, en `-Encoding` bepaald naar welk type encodering geconverteerd moet worden. In dit geval zal het bestand naar `Byte` geëncodeerd worden, zo wordt de binaire data terug hersteld en ontstaat het originele bestand. Vervolgens wordt het permanente bestand ook verborgen a.d.h.v. het `attrib +h <bestand>` commando. Hierna wordt het tijdelijke bestand verwijderd met het commando `rm <Tijdelijk bestand>`.

Tenslotte wordt het permanente bestand, dat de malware voorstelt, uitgevoerd op het systeem a.d.h.v. het command `<Resultaat bestand>`. Om ervoor te zorgen dat dit proces op de achtergrond gebeurt en de gebruiker niets te zien krijgt wordt achter het commando de parameter *0* meegegeven als volgt `Shell " ",0`.

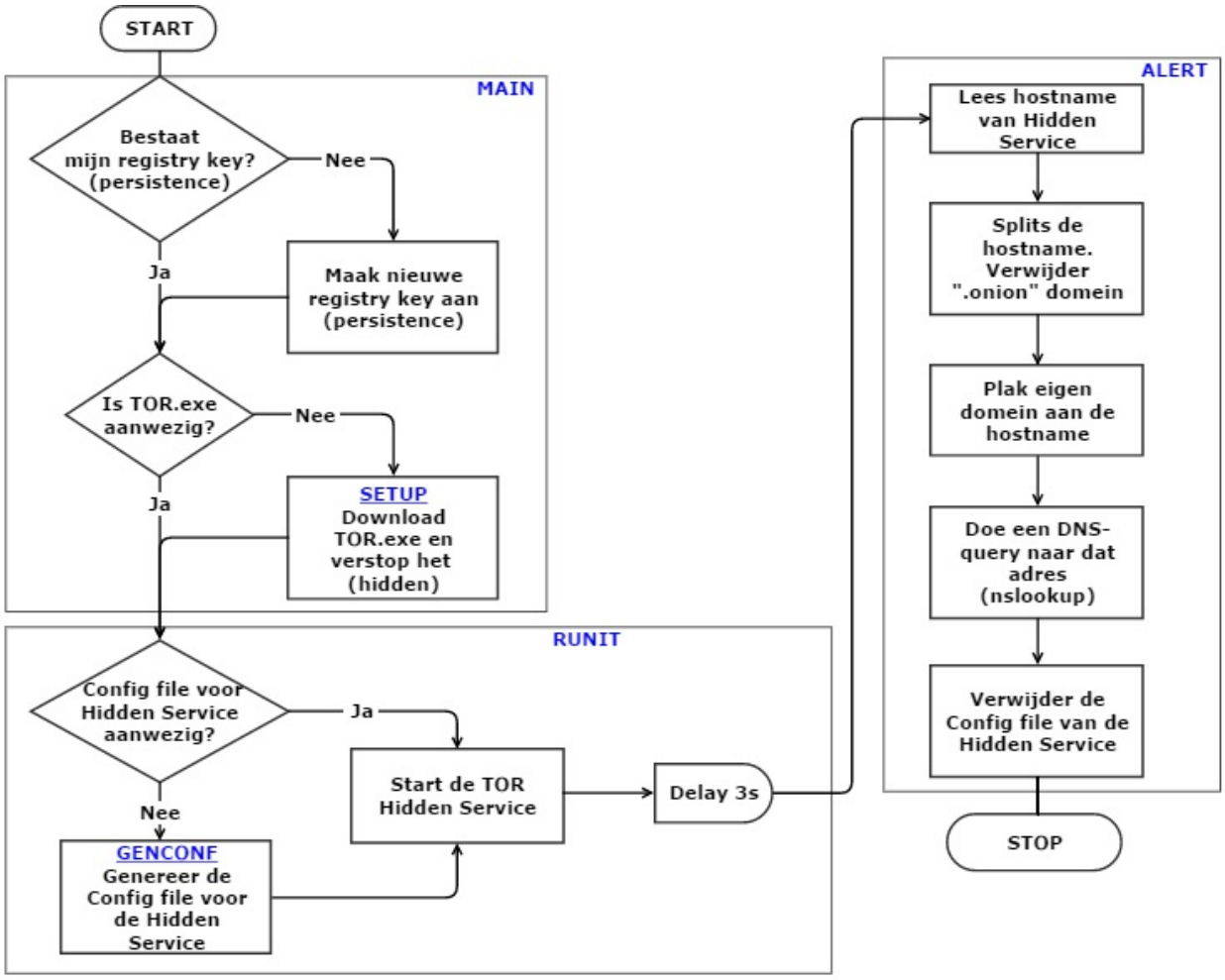
WaitFor

De *WaitFor* functie dient in dit script als een soort *delay*. Omdat VBAniet over voorgemaakte *delay* functies beschikt, werd voor dit project een eigen *delay* functie opgesteld. Deze functie vraagt een parameter dat het aantal seconden voorstelt en zal een `while` lus doorlopen zolang de gevraagde wachttijd nog niet verstreken is. Zodra de wachttijd verstreken is, stopt de while lus en gaat het programma verder met zijn normale verloop.

2.3.3. Flowchart

In dit deel wordt de sequentie van het malware programma, geschreven in C#, getoond. Dit programma is het kern programma dat het C&C-kanaal opstelt en mogelijk maakt. Merk op dat andere programma's zoals het *MacroGenerator* script, of de VBA *Dropper Macro* niet besproken worden. Dit is zo omdat deze scripts geen meerwaarde hebben aan de werking van het C&C-kanaal. Deze programma's hebben enkel bijdragen aan het versturen/verdelen van de malware.

In deze *flowchart* zijn de processen die eenzelfde functie voorstellen omkaderd. In de rechter boven hoek van deze kaders staan telkens de namen van de functies die voorgesteld worden in het blauw.



2.3.4. Problemen & Oplossingen

2.3.4.1. Compact TOR Client

In dit project werd gebruik gemaakt van de TOR client voor het opstellen van een anoniem C&C-kanaal. Zo was het idee om een TOR *client* op het systeem van een doelwit te krijgen zodat er vanuit dat systeem een TOR *hidden service* gehost kan worden.
Het probleem was dat de TOR-client bestaat uit een groot pakket aan *dll* bestanden. Dit is niet zo eenvoudig om in zijn geheel te verplaatsen naar een ander systeem. Downloaden en installeren op het doelwit systeem was ook geen optie omdat dit zekere rechten vereist op het systeem alsook interactie met de gebruiker, wat liefst vermeden wordt.
De oplossing was het compileren van een *Static Binary* van de TOR *client*. Een *Static Binary* is één bestand van een applicatie zoals, in dit geval, de TOR *client*, met daarin alle benodigde *dll* bestanden. Op deze manier is de gehele TOR *client* samengeperst tot één enkel bestand, zo is deze compact gemaakt. Dit compileren gebeurde m.b.v. de MinGW applicatie.
MinGW staat voor Minimalist GNU for Windows, dit wil zeggen dat het een ontwikkelings applicatie is van GNU voor Windows kernels. In weze is het een terminal emulator gebaseerd op bash dat de GNU utils aanbiedt. Het maakt gebruik van *gcc* en *binutils* en kan daarom gebruikt worden voor crosscompilatie. Dit is het compileren van software voor Windows kernel, onder gebruik van een ander systeem (Linux, BSD, Mac, ...).
Voor dit project werd gebruik gemaakt van de stap-voor-stap instructies hier [1] beschikbaar.

Dit leidde tot een ander probleem namelijk dat de *Static Binary* te groot was in opslagruimte. Zo was deze *Static Binary* 10MB groot, wat zeer groot is voor een malware bestand. De bedoeling is dat deze file zo klein mogelijk is zodat deze zo snel mogelijk gedownload is en niet al teveel plek op het systeem inneemt. Grotere files die gedownload worden zullen meer ook meer opvallen binnen het netwerktrafiek, dit is ook nog nadelig.
Om de file kleiner te maken zijn er verschillende stappen genomen. Zo werdt de *Static Binary* eerst *gestript*. *Strippen* houdt in het wegnemen van de *debug* configuraties wat wil zeggen dat alles wat te maken heeft met *debuggen* in de binary zal worden verwijderd. Aangezien deze *static binary* niet bedoeld is om *gedebugged* te worden, maar om gewoonweg te werken, kunnen we deze *debug* configuraties missen. Na *strippen* was de grootte van de *Static Binary* gedaald van 10MB naar 4MB. Dit is al een opmerkelijk verschil, maar nog steeds een groot bestand.
Om de *Static Binary* nóg kleiner te maken werd gebruik gemaakt van een *Binary Packer*, dit zijn software programma's die bedoeld zijn om een *binary* bestand de compresseren tot een kleinere compacte grootte. Binnenin deze compressie wordt code toegevoegd die ervoor zorgt dat wanneer de gecompresseerde versie uitgevoerd wordt, het bestand zichzelf zal decompresseren *at runtime* en uitgevoerd wordt op normale wijze. Op deze wijze is de *Static Binary* compact en blijft de werking onveranderd.
Als *packer* werd voor dit project gebruik gemaakt van de *Mew* packer. Andere packers zijn bijvoorbeeld UPX, ASP, Armadillo, etc. Door het gebruik van een *packer* verkleinde de grootte van de *Static Binary* van 4MB naar 1,36MB.

Nu de TOR *client* succesvol gecompresseerd werd tot één *Static Binary* die vervolgens *gestript* en *gepacked* werd, kan deze gehost worden op een server. Zo zal de uiteindelijk malware (geschreven in C#) deze *Static Binary* kunnen downloaden vanuit die server naar het systeem van het doelwit.

2.3.4.2. Persistency

Wanneer een malware *persistant* is, wil dit zeggen dat het "volhardend" is op het systeem. M.a.w. zal de malware steeds actief zijn als het systeem actief is. Er zijn verschillende mogelijkheden om een malware *persistant* te maken enkele voorbeelden zijn *DLL Hijacking*, *Scheduled Tasks*, *Windows Registry Keys*,...
In dit project werd gekozen voor de *Windows Registry Key Persistence* techniek. Voor deze techniek wordt gebruik gemaakt van de *Windows Registry*, dit is een database waar alle *low-level* instellingen van het systeem worden opgeslagen. Zo kunnen bepaalde *Keys* in de *Registry* bepalen welke programma's uitgevoerd worden bij het

opstarten van het systeem waar in deze techniek dan ook gebruikt wordt. Zo zal er een *Value* worden toegevoegd aan de *Registry Key* die bepaald welke programma's opgestart worden. Die *Value* refereert dan naar het malware programma, wat ervoor zal zorgen dat het malware programma steeds zal starten telkens het systeem opstart.

2.3.4.3. Delen van de Hidden Service hostname

Een probleem dat zich voorstelde tijdens het opstellen van dit C&C-kanaal was het meedelen van de hostname of adres van de TOR *hidden service*. Indien de *attacker* toegang wenst tot de *hidden service* die de malware openstelt vanuit het systeem van het doelwit, moet de *attacker* het adres, of de hostname, van die *hidden service* kennen. Dit adres bevindt zich in een folder op het systeem van waar de *hidden service gehost* wordt. Omdat de *attacker* nog geen toegang heeft tot dat systeem, kan hij/zij dit adres niet zien te achterhalen. Daarom moest hier een oplossing voor bedacht worden.
De oplossing voor dit probleem is een DNS-query. Om deze oplossing te begrijpen wordt eerst de basiswerking van DNS toegelicht.

Het Domain Name Service (DNS) protocol zorgt voor het omvormen van IP-adressen naar een hostname en vice versa. Zo kan er bijvoorbeeld een aanvraag gedaan worden naar een bepaalde *hostname* om zo te weten te komen wat het IP-adres voor die *hostname* is. Bij DNS-query wordt er aan uw lokale DNS-server gevraagd of die weet van waar de hostname afkomstig is. Wanneer die DNS-server dit onderzoek zal hij eerst kijken naar het domein van de hostname. Het domein van een hostname kan bijvoorbeeld zijn *.be .com .uk*.
Wanneer het domein gekend is zal de lokale DNS-server vaststellen waar de algemene DNS-server is voor dat domein. Stel bijvoorbeeld dat het domein *".be"* is, dit is het domein voor België. Dan zal de lokale DNS-server de DNS-query doorsturen naar de centrale DNS-server voor België, omdat die server alle adressen met dat domein kent.

Door gebruik te maken van dit principe, is het mogelijk om de *hostname* of adres van de *hidden service* te delen met de *attacker*. Zo zal de malware de file, waarin de *hostname* staat, uitlezen en de *hostname* opslaan. Elke *hostname* van een TOR *hidden service* heeft het domein *".onion"*. Omdat de *attacker* gecontacteerd moet worden, zal de *attacker* een eigen DNS-server opstellen met zijn eigen domein. Vervolgens zal de malware het *".onion"* domein van de *string* met de *hostname* verwijderen en wordt het domein van de *attacker* er aan toegevoegd.
In dit project stelt het ITRAteam de *attacker* voor, dus wordt in dit geval het domein van hun DNS-server (*".ey.be"*) eraan toegevoegd. Dan wordt bijvoorbeeld de *hostname* "ayr7h93ke7nh6bv0**.onion**" omgevormd naar "ayr7h93ke7nh6bv0**.ey.be**".
Vervolgens wordt er een DNS-query gedaan naar de *hostname* met het domein van de *attacker* (*"ey.be"*). Omdat dit het domein van het ITRAteam is zal de DNS-query doorgestuurd worden naar de DNS-server van het ITRAteam. Het team kan dan hun DNS-trafiek controleren en deze query terugvinden. Vervolgens kan het team hieruit afleiden dat die query de *hostname* van de *hidden service* is.

Tenslotte wordt het domein van de *attacker*, *"ey.be"* in dit geval, terug vervangen door *".onion"*, waardoor de oorspronkelijke *hostname* hervormd wordt. Nu de *attacker* de *hostname* van de TOR *hidden service* kent, kan de *attacker* met die *hidden service* verbinden door, via TOR, met die *hostname* te verbinden.

3. Conclusie

De doelstellingen voor dit project zijn behaald. Zo zijn er twee C&C-kanalen verwezenlijkt, waar er minstens één uitgewerkte gevraagd werd. Bovendien is ook de mogelijkheid tot verspreiden van de malware (*dropper*) en *persistency* van de malware uitgewerkt. Dit was geen doelstelling voor het project, maar als extra uitbreiding en uitdaging is dit toch verwezenlijkt. Uiteraard zijn er nog talloze andere mogelijkheden voor C&C-kanalen, maar ik koos voor de gebruikte technologieën omdat deze voor mij het meest interessante waren. Het TOR-netwerk is een interessante technologie, zeker voor *Covert Communication Channels* omdat deze technologie, zoals besproken, voor totale anonimiteit op het Internet zorgt.

Dit project bespaard het EY ITRA-team veel tijd bij het uitwerken van hun *Remote Acces Framework*. Voor dit *framework* zoekt het team namelijk voortdurend nieuwe technologieën om hun arsenaal aan C&C-kanalen mogelijk uit te breiden. Afhankelijk van de technologie kan het uitwerken van een C&C-malware lang duren. Hierbij bespaart dit project het ITRA-team dus de tijd die ik spendeerde aan het uitwerken van deze C&C-kanalen, terwijl hun arsenaal uitgebreid wordt. Omdat deze C&C-kanalen steeds andere regels moeten volgen, afhankelijk van het netwerk waarin ze worden gebruikt, zijn deze flexibel gemaakt, wat wil zeggen dat ze eenvoudig aan te passen zijn a.d.h.v. variabelen.

De stage was zeer leerrijk. Ik was nooit eerder in contact gekomen met C&C-kanalen, noch het schrijven van malware. Ik heb zeer veel bijgeleerd omtrent dit onderwerp en het hele *cyber security* gebeuren.

In de toekomst is het interessant de malware te herschrijven in de programmeertaal C++ omdat deze meer *low-level* is dan C#. De C++ programmeertaal is ook opmerkelijk moeilijker te analyseren voor malware analisten, omdat het moeilijker is om deze te *reverse engineeren* dan de *high-level* programmeertalen zoals C#.

3.1. Statistieken

In dit deel worden de statistieken en testresultaten van de C&C-kanalen besproken.
Beide C&C-kanalen werden volledig getest binnen een werkend bedrijfsnetwerk. Vanwege confidentialiteit mag de naam van het bedrijf noch details over de netwerkinfrastructuur en alle configuraties niet vrijgegeven worden.

3.1.1. Outlook C&C

De Outlook C&C werkte zonder enig probleem en de vereisten zijn minimaal. Zo moet er een Outlook applicatie aanwezig en actief zijn, en verbonden zijn met de Exchange server. Vereist is dat er ingelogd is in een geldig Outlook account. Het script zal steeds met dezelfde rechten als de gebruiker van het systeem werken. Het is belangrijk voor de *attacker* om te weten wat exact de vereisten zijn voor e-mails om doorgelaten te worden door de Exchange server (mailserver). Indien mails niet zouden voldoen aan de eisen die de Exchange server opstelt, zullen de mails nooit aankomen bij het doelwit. Deze configuraties zijn voor elk bedrijf anders.

3.1.2. TOR C&C

De TOR C&C malware werkte, net zoals de Outlook C&C, zonder enig probleem op het voorvernoemde bedrijfsnetwerk.
Enkele vereisten voor dit C&C-kanaal is dat er toegang is tot het ophalen van de TOR static binary. Uiteraard moet de netwerk infrastructuur het gebruik van TOR toestaan, zo niet zal het gehele C&C-kanaal niet opgestart kunnen worden. Dit zal voor elk bedrijfsnetwerk anders zijn.
De malware zelf zal steeds werken met dezelfde rechten als de gebruiker van het systeem. Er zijn geen specifieke vereisten voor administrator rechten.

Voor het *delivery* aspect van de malware is het belangrijk dat de Word *macro* uitgevoerd kan worden. Het is mogelijk dat macro's over het algemeen geblokkeerd worden binnen een bedrijfsnetwerk, maar dit zal weinig tot bijna niet voorkomen. Dit komt omdate veel documenten die intern opgesteld worden gebaseerd zijn op de werking van *macro's*. Daarom zal de meest voorkomende beveiligingsmaatregel zo zijn dat *macro's* geblokkeerd worden, maar dat de gebruiker ervoor kan keizen deze toch toe te staan indien nodig.

4. Bronnenlijst

- Wikipedia: https://www.wikipedia.org
- .NET ontwikkeling (powershell, Outlook objecten, C#, ...): https://msdn.microsoft.com/en-us/default.aspx
- Outlook Security Patch: http://www.slipstick.com/outlook/outlook-email-security-update/
- Redemption Library:
  - Objects and Folders: http://www.msxfaq.de/code/redemption.htm
  - Redemption Library: http://www.dimastr.com/redemption/home.htm
- The Onion Router (TOR): https://www.torproject.org/
  - Afbeeldingen:
    - TOR Netwerk Nodes: https://nl.wikipedia.org/wiki/Tor_(netwerk)#/media/File:Wat_is_Tor_(The_onion_routing)%3F.png
    - TOR Encryption Layers: https://en.wikipedia.org/wiki/File:Onion_diagram.svg
- Static Binaries compileren: http://www.mictronics.de/2014/04/how-to-build-tor-for-win32
- MinGW : http://www.mingw.org/
- Stackoverflow: http://stackoverflow.com/
- Flowcharts: https://www.gliffy.com/
- Interessante lectuur omtrent TOR C&C: https://foxglovesecurity.com/2015/11/02/hack-like-the-bad-guys-using-tor-for-firewall-evasion-and-anonymous-remote-access/
- Blog: Didier Stevens: https://blog.didierstevens.com/
- Informatie omtrent Persistency en opbouw van malware: https://www.fireeye.com/index.html

5. Woordenlijst

- C&C = Command & Control
- IDS = Intrusion Detection System
- IPS = Intrusion Prevention System
- NPD = Network Perimeter Defences
- TOR = The Onion Router
- HS = Hidden Service
- SSH = Secure SHell
- HTTP = HyperText Transfer Protocol
- HTTPS = HyperText Transfer Protocol Secure
- RDP = Remote Desktop Protocol
- SMTP = Simple Mail Transfer Protocol
- CPU = Central Processing Unit
- IDE = Integrated Development Environment
- ISE = Integrated Scripting Environment
- MSVS = MicroSoft Visual Studio
- VBA= Visual Basic for Applications