Ofuscament de malware per al bypass dels antivirus comercials

Pau Olivé Montejo

Escola Politècnica Superior d'Enginyeria de Vilanova i la Geltrú

Resum

Els antivirus són la primera línia de prevenció i protecció de tots els sistemes informàtics moderns. Dins del món del programari maliciós (*malware*), des de l'aparició dels primers antivirus comercials, hi ha una guerra constant entre aquestes empreses i els desenvolupadors de *malware*. L'objectiu d'aquests últims és aconseguir que els seus *payloads* siguin totalment indetectables davant de qualsevol sistema de seguretat. Una de les maneres d'aconseguir-ho és amb l'ús d'una eina anomenada *Crypter*, *Software Packer* o *Packer*, que no deixen de ser programes que encripten un executable (normalment maliciós i actualment detectat per molts antivirus), i hi afegeixen funcions úniques encarregades de desencriptar aquest mateix executable inicial només quan l'usuari executi l'arxiu, i carregar aquest programari maliciós directament a memòria dinàmica sense tocar el disc dur utilitzant tècniques d'injecció en memòria.

1. Introducció

L'objectiu d'aquest treball és l'estudi de les tècniques d'ofuscació que s'utilitzen en els *crypters* moderns i la implementació d'un *crypter* com a *Proof of Concept* (PoC), posant a prova la majoria d'antivirus comercials. Això suposa conèixer les tècniques de detecció que utilitzen aquests antivirus i comprendre el seu funcionament. També s'aprofundeix en el funcionament intern de Windows, doncs és necessari tenir una idea ben clara sobre com funcionen i quina estructura mantenen els *PE* (*Portable Executable*) [1] d'aquest sistema operatiu i caldrà utilitzar funcions pròpies de l'API de Windows (*WinAPI*) [2]. Es farà ús d'una *webapp* a mode de presentació del *crypter,* on l'usuari podrà penjar els arxius i altres dades necessàries per poder rebre el seu executable final. Finalment, es realitzarà un conjunt de tests utilitzant *malware* real per avaluar l'eficàcia de la nostra implementació del *crypter*. Per aquesta avaluació es farà ús d'un *multiscanner* [3] (servei web que ofereix un escaneig pels 26 antivirus comercials amb més renom gairebé per tot tipus d'arxius obtenint els resultats de manera quasi instantània).

El propòsit d'aquest treball és reduir les ràtios de detecció de qualsevol *malware* al mínim possible, sent el millor resultat un arxiu *malware* FUD (*Fully Undetectable*), utilitzant el *crypter* juntament amb tècniques d'ofuscació addicionals.

2. Els *crypters*

Hi ha moltes maneres d'implementar un *crypter* i aquests poden estar programats amb diferents llenguatges de programació. Un *crypter* és un programari encarregat d'agafar qualsevol executable (ja sigui programari maliciós o no), llegir byte a byte el seu contingut, i encriptar-lo en una memòria intermèdia o buffer, per tal que aquest buffer sigui posteriorment desencriptat en temps real d'execució i carregat dinàmicament a memòria sense tocar el disc.

Un *crypter* està format principalment per dos projectes:

- *Builder*: Aquest projecte serà l'encarregat de llegir i encriptar l'executable que volem ofuscar. L'executable encriptat serà emmagatzemat en un buffer i, a partir d'aquí, hi ha dues maneres habituals d'afegir aquest buffer a l'altre projecte, el de l'*stub*. La primera manera és la d'emmagatzemar el buffer com a sortida en un arxiu capçalera que serà utilitzat pel segon projecte, el de l'*stub*. Per exemple, emmagatzemar-lo a l'arxiu `shellcode.h`. L'altra manera és la de simplement guardar aquest buffer com si fos un recurs dins del segon projecte (el mateix que es faria per agregar una icona, per exemple).
- *Stub*: Aquesta és la part més important del *crypter*, ja que serà el projecte encarregat de llegir el buffer amb l'executable encriptat, desencriptar-lo i carregar-lo dinàmicament a memòria. També s'aplicaran mètodes addicionals per reduir les possibles deteccions dels antivirus.

Per carregar un executable a memòria hi ha diversos mètodes d'injecció que es poden utilitzar. La injecció de codi a memòria consisteix bàsicament en carregar els bytes d'una imatge binària dins d'un procés actiu qualsevol, de tal manera que aquest procés es passi a comportar com l'executable que formaven aquests bytes però sense haver-lo executat directament des del disc dur. Més endavant s'explicarà detalladament el mètode d'injecció utilitzat, anomenat *Process Hollowing*.

Addicionalment, s'implementarà un tercer projecte extern al *crypter* per ajudar a millorar els resultats d'aquest. Aquest tercer projecte serà l'encarregat de clonar el certificat digital d'un executable reconegut, cap a qualsevol altre executable (en aquest cas, el clonatge serà cap a l'executable resultant del nostre *crypter*).

3. Antivirus

Els antivirus comercials són el software més utilitzat de cara a la protecció de dades i equips informàtics, sobretot pels usuaris comuns però també a nivell d'empreses.

En aquest apartat es tractaran els aspectes necessaris que s'han de saber sobre la manera en la que es detecten arxius maliciosos per així tenir un coneixement general dels passos que s'han de seguir per poder fer un *bypass* als seus sistemes de detecció [4].

- **Detecció basada en signatures**: Aquesta tècnica, principalment, té a veure amb l'anàlisi manual feta pels investigadors de *malware* (tot i que solen utilitzar eines que els automatitzen certes tasques) amb l'objectiu de trobar patrons en els arxius maliciosos. Quan un arxiu és detectat com a maliciós, es generen noves signatures relacionades i s'emmagatzemen a les seves bases de dades. Més endavant, quan un antivirus realitza un escaneig, utilitza tècniques de concordança de patrons (*pattern-matching*) i compara les possibles signatures detectades amb les de la seva base de dades. Hi ha molts tipus de signatures possibles i inclús algunes poden arribar a generar falsos positius. Aquesta tècnica no és gaire efectiva contra nous *malwares* desconeguts i, fins i tot, amb molts *malwares* ja coneguts tampoc acaba de ser molt efectiva perquè poden arribar a utilitzar polimorfisme en el seu codi mateix.

- **Heurístiques**: És una de les tècniques que s'utilitza per detectar virus nous o poc distribuïts dels quals encara no es posseeix cap signatura a la base de dades relacionada amb el seu codi maliciós. La idea general és la d'utilitzar definicions genèriques de *malware* (que no deixen de ser un tipus de signatures però de caràcter general) per poder detectar variants d'aquests. Al distribuir els diferents *malwares* existents en famílies, fa possible reconèixer àrees úniques que permeten crear signatures genèriques.

- **Detecció en temps real d'execució**: Aquest mètode de detecció està basat en l'anàlisi de comportament del *malware* mentre aquest està sent executat. Bàsicament, l'antivirus monitoritza tots els processos en execució. Això li permet investigar els processos que s'estan comportant d'una manera inusual com, per exemple, aquells que volen connectar-se a un servidor extern remot i descarregar arxius.

- ***Sandboxing***: Aquest mecanisme consisteix a executar els programes en un entorn separat / aïllat controlant tots els recursos assignats en cas de danys. Es considera un mecanisme de contenció contra les tècniques d'ofuscació del *malware*, ja que realment no es pot saber quin serà el comportament d'un arxiu ofuscat fins que aquest sigui executat.

4. Format PE

El format PE [1] és una estructura de dades que proporciona la informació necessària perquè el *loader* de Windows pugui gestionar el codi de l'executable de la millor manera possible. Això inclou poder resoldre referències a DLL, taules d'importació i exportació de les API, una gestió eficient de recursos i de dades d'emmagatzematge local de subprocessos. És important entendre certs components clau d'aquest format ja que s'utilitzaran per poder recórrer les estructures internes del *malware* amb les que tractarem més endavant.

En la Figura 1 es pot apreciar una representació gràfica del format PE.

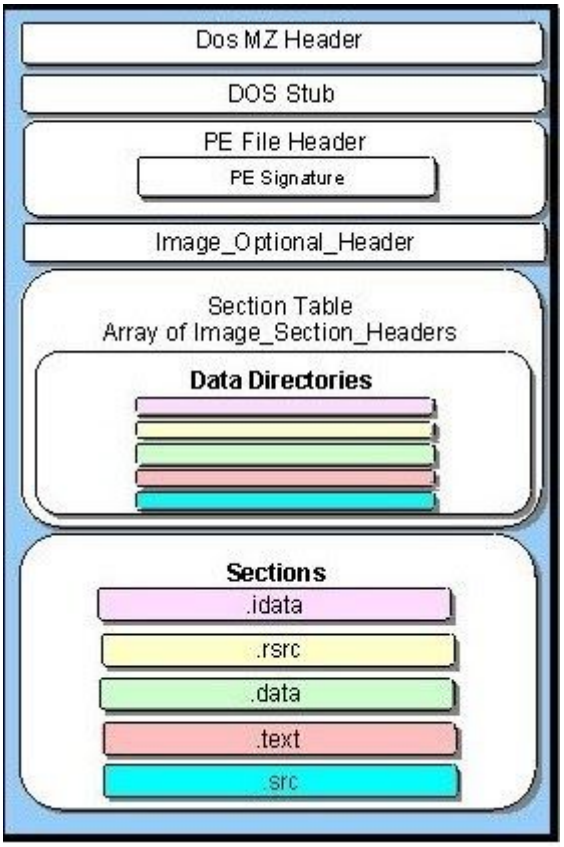

Fig. 1. *Estructura interna d'un fitxer PE*

Un arxiu PE es divideix en capçaleres i seccions. Les capçaleres contenen la informació més rellevant del executable en si, com les seves característiques, mentre que les seccions emmagatzemen el contingut de l'executable (codi, dades, recursos, etc.).

Fent ús de les capçaleres es pot accedir a tots els camps necessaris dins d'un executable. Això és essencial per poder realitzar la tècnica d'injecció de codi a memòria que

descriurem pròximament. Alguns dels camps més rellevants de les capçaleres del format PE són els següents:

1. Camp `e_lfanew` de la capçalera *DOS Header*, que ens permetrà accedir a la capçalera *PE File Header*.
2. Camp `NumberOfSections` de la capçalera *PE File Header*, que ens indica el nombre total de seccions de l'executable.
3. Camp `AdressOfEntryPoint` de la capçalera *Optional Header*, que ens indica la direcció del punt d'entrada relativa a la imatge base quan l'arxiu executable és carregat a memòria.
4. Camp `ImageBase` de la capçalera *Optional Header*, que ens indica l'adreça base preferida en l'espai de memòria del procés perquè la imatge de l'executable pugui ser assignada.

Addicionalment, la capçalera *Optional Header* conté directoris, formats per dos camps: el camp `VirtualAddress`, que apunta la taula d'aquest directori (el contingut) i el camp `Size`, que indica la mida d'aquesta taula. D'aquests directoris els més rellevants són:

- ***IMAGE_DIRECTORY_ENTRY_EXPORT:*** Directori en el qual el seu camp `VirtualAddress` apunta a una taula anomenada `Export Table`. En aquesta taula és on apareixen totes les funcions exportades (normalment per arxius DLL) per ser utilitzades per altres mòduls.

- ***IMAGE_DIRECTORY_ENTRY_IMPORT:*** Directori en el qual el seu camp `VirtualAddress` apunta a una taula anomenada `Import Address Table`, que és on apareixen totes les funcions importades de biblioteques, i adreces d'aquestes, juntament amb altres dependències externes, per l'executable.

- ***IMAGE_DIRECTORY_ENTRY_SECURITY:*** Directori en el qual el seu camp `VirtualAddress` apunta a una taula (sense nom oficial) que conté els certificats digitals de l'executable.

Les seccions d'un executable també tenen les seves pròpies capçaleres. Per a cada secció hi ha una única capçalera i el nombre total serà equivalent al camp `NumberOfSections` descrit prèviament. Hi ha diferents tipus de seccions predefinides com la secció `.text` (que conté el codi executable), les seccions de dades `.bss`, `.rdata` i `.data` (que contenen les dades, variables, etc.) i la secció `.rsrc` (que conté els recursos, com la icona, fonts, etc.).

5. Arquitectura del *Crypter*

Com s'ha descrit prèviament, el *crypter* està format per dos projectes: el *builder* i l'*stub*.

- El *builder* ha de rebre l'executable *malware* com a paràmetre d'entrada, encriptar els seus bytes amb un algoritme d'encriptació (RC4 [5] en aquest cas), codificar-los en Base64 [6] (per reduir l'entropia dels bytes resultants, ja que una alta entropia pot ser sospitós a ulls dels antivirus), i guardar aquests bytes encriptats en una variable continguda en un arxiu capçalera (`shellcode.h`, per exemple) resultant com a sortida. La representació gràfica del *builder* de la Figura 2 pot ajudar a entendre millor el seu funcionament.

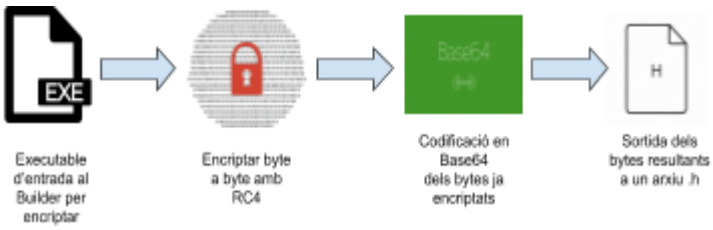

Fig. 2. *Representació gràfica del builder*

- L'*stub*, sent la part més important del *crypter*, utilitzarà aquest arxiu capçalera resultant del *builder*, descodificarà en Base64 i desencriptarà en RC4 utilitzant la mateixa clau la variable que conté els bytes del *malware*. Posteriorment, carregarà aquests bytes a memòria, sense tocar el disc dur, utilitzant la tècnica d'injecció de codi anomenada *Process Hollowing*. Afegir que, addicionalment, s'implementaran altres tècniques d'ofuscació dins de l'*stub* per millorar el seu rendiment.

6. *Process Hollowing*

Un cop ja tenim els bytes de l'arxiu binari original, hem de procedir a carregar-los dinàmicament a memòria sense tocar el disc dur. Això ho aconseguirem amb la tècnica anomenada *Process Hollowing* [7,8] que consta, de manera general, dels següents passos:

1. Accedir a les capçaleres del *malware* ja desencriptat, que ens permetran tenir accés a tot el contingut d'aquest.
2. Crear un nou procés en un estat suspès. En aquest cas, el procés que es crearà és una còpia del mateix procés.
3. Modificar-ne el contingut amb la imatge d'un altre executable (en aquest cas, pels bytes ja desencriptats).
4. Fer les modificacions necessàries perquè l'execució de la imatge en el nou procés sigui correcta (permisos i arranjaments d'adreces clau).
5. Reprendre l'execució del procés suspès.

La representació gràfica d'aquest mètode de la Figura 3 pot ajudar a entendre millor el seu funcionament.

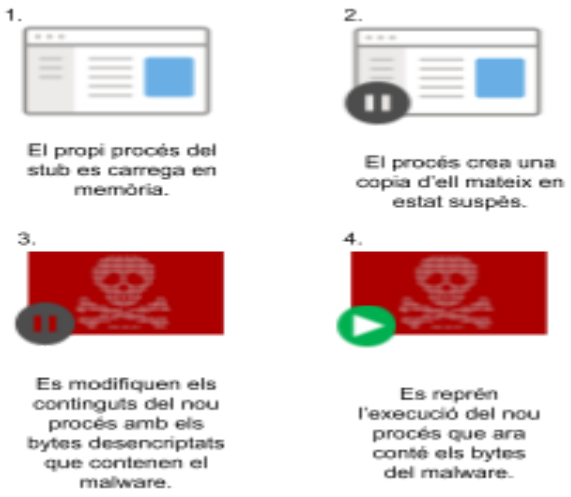

Fig. 3. *Representació gràfica del mètode d'injecció a memòria Process Hollowing*

7. Tècniques addicionals d'ofuscació per *l'stub*

Per millorar els resultats obtinguts del *crypter*, s'han implementat tècniques addicionals d'ofuscació. Les tècniques són les següents:

1. ***Junk API Calls***: Tècnica que es basa en afegir crides a l'API de Windows molt poc comunes en el *malware* (per tant, inofensives a ulls dels antivirus) de manera intercalada enmig de la implementació del *Process Hollowing*. L'objectiu es trencar signatures basades en patrons genèrics (detecció heurística).

2. **Ofuscació d'importacions**: Tècnica que es basa en l'ofuscació de les crides a l'API que apareixen dins de la `Import Address Table`, descrita en l'apartat 4. Consisteix en resoldre totes les importacions a aquestes crides en temps d'execució, així només apareixen quan el procés estigui carregat a memòria.

3. **Anti-emulació de codi**: Tècnica que es basa en la detecció d'un entorn d'emulació per part dels antivirus. Hi ha diverses maneres de detectar si el nostre executable està sent emulat. En aquest cas, es carregaran tant com DLL existents, amb la crida `LoadLibraryA` [9] de Windows, i si retorna `NULL`, voldrà dir que el codi està sent emulat; al mateix temps, també es carregaran DLL fictícies, i si la crida no retorna `NULL`, també voldrà dir que el codi està sent emulat. Això és així perquè els entorns emulats dels antivirus disposen de recursos limitats i no poden emular tot el sistema i els seus fitxers. Quan hi ha crides com aquestes que busquen dependències externes, o bé retornen sempre un resultat qualsevol, existeixin o no, o bé no retornen res perquè no ho tenen emulat (per qüestió de recursos). En cas que es detecti un entorn emulat, l'execució de l'*stub* finalitzarà, comportant-se per tant, de manera totalment inofensiva. Si no se'n detecta cap, l'*stub* seguirà la seva execució habitual.

8. Clonatge de certificat digital

Aquest és un projecte extern al *crypter* que podria considerar-se una tècnica d'ofuscació addicional. Bàsicament, rebrà dos executables d'entrada i clonarà el certificat (o signatura digital) d'un executable a l'altre. No tots els executables tenen certificat però, els que provenen d'empreses legítimes que volen oferir software com a producte, solen obtenir certificats per als seus arxius. Anant a propietats d'un executable, podrem veure si aquest conté o no una firma digital. En la Figura 4 podem veure un exemple del certificat digital de l'executable `Spotify.exe`.

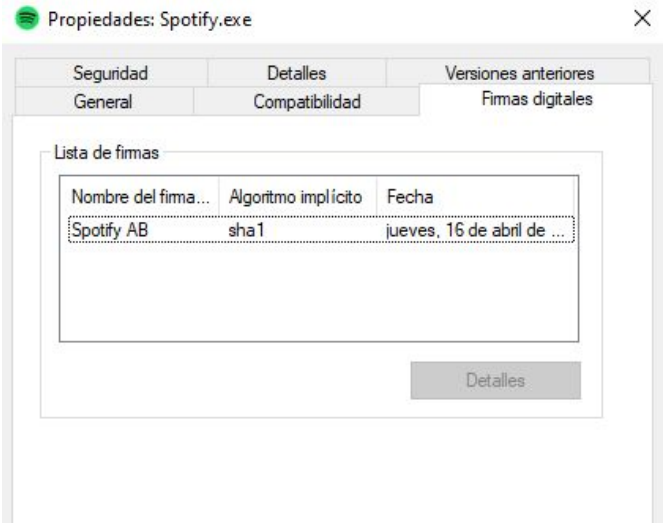

Fig. 4. *Firma digital de l'executable* `Spotify.exe`

Alguns antivirus tenen en compte si un executable conté un certificat digital o no i, si no el conté, pot aixecar les sospites d'aquest cap a l'executable. Amb tot, no vol dir que l'hagi de marcar com a maliciós només per això, ja que té en compte molts altres elements que ja s'han anat explicant al llarg de l'article.

Com es menciona a l'apartat 4, aquests certificats digitals s'emmagatzemen dins de l'estructura interna del format PE, específicament a la direcció on apunta el camp `VirtualAddress` del directori de dades *IMAGE_DIRECTORY_ENTRY_SECURITY*. Coneixent aquesta informació, haurem d'anar a buscar aquest directori a l'executable del que voldrem clonar el seu certificat, copiar tot el contingut de la seva taula fent ús dels camps `VirtualAddress` i `Size` del directori, i emmagatzemar aquesta còpia a l'executable final (el nostre *stub* que ja conté el *malware* encriptat en aquest cas), també recorrent l'estructura interna del PE per trobar el seu *IMAGE_DIRECTORY_ENTRY_SECURITY* corresponent i saber on emmagatzemar el certificat copiat.

9. *WebApp*

La *webapp* s'utilitzarà a mode de presentació del *crypter*. Consisteix en un *frontend* senzill que mostra a l'usuari un formulari on podrà penjar: l'arxiu que vol fer passar pel *crypter*, la icona per a l'executable final, la clau d'encriptació a utilitzar, l'executable del que vol clonar el certificat, i el nom de l'executable final. En la Figura 5 es pot apreciar la interfície inicial.

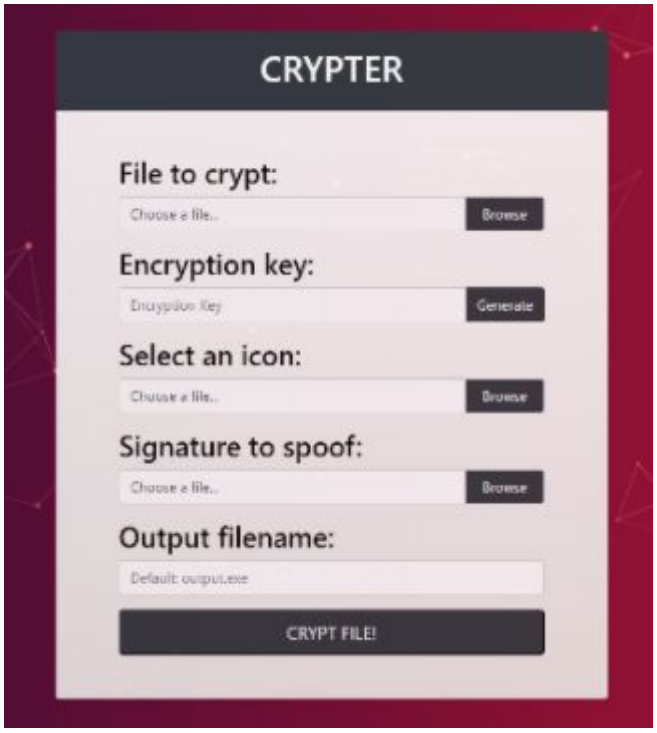

Fig. 5.*Interfície que mostra la pantalla principal quan l'usuari obre la webapp*

Quan l'usuari omple el formulari amb tota la informació necessària descrita anteriorment, podrà prémer el botó "CRYPT FILE!". Aquest botó canviarà de color a verd, es modificarà el contingut del text amb un *spinner* i un missatge per avisar a l'usuari que el seu arxiu està sent processat i que pot durar uns segons. En la Figura 6 es pot apreciar aquesta modificació del botó.

Fig. 6. *Botó no clicable que indica a l'usuari que el seu arxiu està passant pel crypter i està rebent altres modificacions (icona, certificat digital i el nom de sortida)*

En la Figura 7 es mostra la interfície final que veurà l'usuari un cop l'arxiu final ja està llest, indicant-li que ja pot descarregar el seu nou executable.

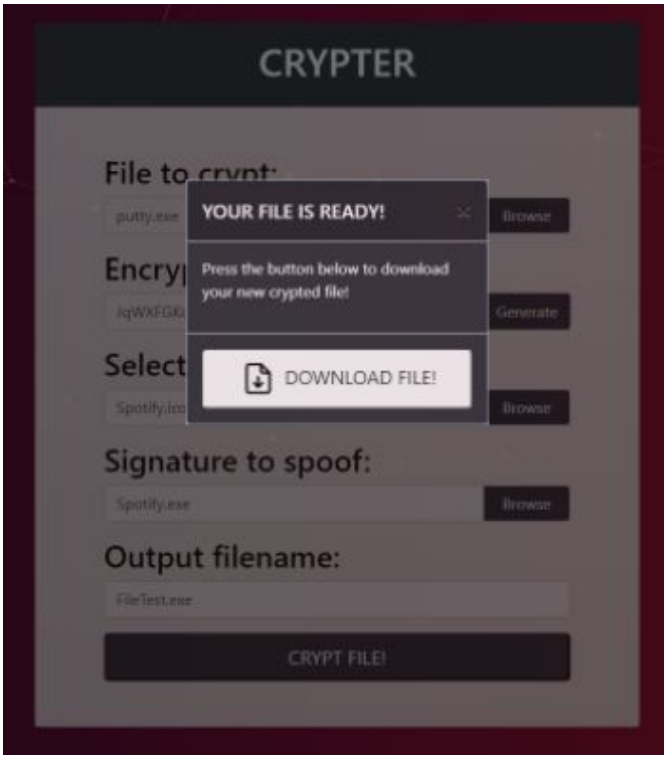

Fig. 7. *Interfície final on apareix el fons amb una opacitat reduïda i una nova finestra al centre indicant a l'usuari que ja pot descarregar el seu executable final*

El *backend* serà l'encarregat de rebre tots aquests fitxers i dades mitjançant el primer *endpoint* (mètode POST), processar-los fins a generar l'executable final, i retornar-lo a l'usuari amb el segon *endpoint* (mètode GET).

10. Model d'avaluació

Per analitzar l'eficàcia del nostre *crypter* s'ha utilitzat el servei d'anàlisi d'antivirus online **AntiScan.Me** [3]. El servei **AntiScan.Me** permet analitzar simultàniament i de manera molt senzilla arxius executables amb un total dels 26 antivirus comercials més coneguts en busca d'amenaces, obtenint els resultats d'aquestes anàlisis molt ràpidament. S'ha escollit el servei d'anàlisi d'**AntiScan.Me** sobre altres multiscanners perquè ofereix fins a sis escaneigs gratuïts per IP, no distribueix els resultats a les empreses dels antivirus (evitant així anàlisis exhaustives i manuals) i ofereix l'opció d'obtenir els resultats en format imatge.

L'arxiu que s'ha utilitzat per realitzar les proves és un *malware* anomenat Remcos [10]. Aquest tipus de *malware* és un RAT (*Remote Administration Tool*), un tipus de virus molt potent que atorga control absolut a l'atacant sobre l'ordinador de la víctima. Algunes d'aquestes funcions són: veure en directe la pantalla de la víctima, espiar la *webcam*, *keylogger* (un log per robar contrasenyes i demés informació de la víctima), *shell* remota, controlar i accedir als arxius de l'ordinador de l'usuari, etc.

La primera anàlisi serà amb l'arxiu *malware* intacte, és a dir, sense passar pel nostre *crypter* ni cap altra eina. Els resultats es poden apreciar en la Figura 8. Del total de 26 antivirus que ofereix aquesta pàgina, podem veure com el *malware* és detectat, quan **no** ha passat pel *crypter*, per un total de 23 antivirus. Alguns inclús reconeixen quin tipus de *malware* és i l'etiqueten com a **Remcos**. Això ens indica que la majoria d'aquests antivirus tenen constància d'aquest arxiu i han generat signatures per a la seva detecció. Podem veure com inclús el mateix Windows Defender, antivirus que ve instal·lat per defecte amb el sistema operatiu de Windows, detecta aquest arxiu com a maliciós; això vol dir que, tot i que l'usuari no tingui instal·lat cap antivirus comercial al seu ordinador, aquest *malware* serà automàticament detectat i eliminat del seu sistema.

Fig. 8. *Resultats obtinguts de l'anàlisi realitzada pel servei d'anàlisi d'antivirus online AntiScan.Me amb el malware Remcos*

L'anàlisi final serà amb el mateix arxiu *malware* passant pel nostre *crypter* implementat, i totes les tècniques d'ofuscació addicionals de l'*stub*, juntament amb el clonatge del certificat digital de l'aplicació `Spotify.exe`. Els resultats es poden apreciar en la Figura 9. Utilitzant totes les tècniques junt amb el *crypter* hem pogut passar de 23/26 deteccions a 0/26, transformant aquest *malware* en un arxiu completament indetectable (FUD).

Fig. 9. *Resultats obtinguts de l'anàlisi realitzada pel servei AntiScan.Me amb el malware Remcos després de fer-lo passar pel crypter juntament amb totes les tècniques d'ofuscació addicionals i el clonatge del certificat digital de l'aplicació* `Spotify.exe`

11. Conclusions i treball futur

Aquest projecte m'ha ajudat a entendre a fons com funcionen internament els executables de Windows, gràcies a l'estudi necessari del format PE que m'ha calgut realitzar per poder implementar el *crypter*. He après vàries tècniques d'ofuscació destinades a l'evasió dels antivirus i una tècnica d'injecció a memòria, com és el *Process Hollowing*, tot i que que tenim constància que hi ha més tècniques d'ofuscació i mètodes d'injecció a memòria que es poden incorporar. Aquest coneixement adquirit sobre els *crypters*, de per si, ja ens és útil per adonar-nos que l'usuari mig mai està protegit del tot, encara que aquest usi antivirus.

Una de les conclusions personals, i suposició al mateix temps, que he tret realitzant moltes proves (no totes documentades aquí), és que els antivirus, quan no disposen d'una signatura de l'arxiu que estan analitzant, per determinar si aquest és maliciós o no utilitzen com una espècie de sistema de puntuació, on van sumant punts si van veient "coses" que no els agrada. Si l'arxiu aconsegueix arribar a una certa quantitat de punts, és marcat com a maliciós. Imagino que aquest és un dels motius perquè a vegades es poden arribar a crear falsos positius. Per exemple, és possible que alguns antivirus, si veuen que un arxiu no té icona o bé no té certificat digital o bé fa servir certa crida de l'API de

Windows, etc. sumi punts, tot i que potser no sigui un arxiu maliciós.

A nivell personal, tot i que esperava poder reduir la ràtio de detecció d'un executable *malware* actualment detectat pels antivirus, en un cert percentatge, no esperava poder arribar a modificar-lo per aconseguir un *malware* FUD, ja que, ho miris com ho miris, aquestes companyies disposen de molts recursos i diners, i al principi això era bastant intimidant. Per tant, puc afirmar que estic molt satisfet amb els resultats i el projecte ha superat les meves expectatives inicials.

De cara al treball futur, hi ha moltes coses que poden ser millorades i afegides al *crypter*. La primera a considerar seria el poliformisme. Si s'aconsegueix fer el codi del *crypter* polimòrfic, on a cada generació de l'*stub* per al mateix arxiu executable el contingut binari fos diferent però la funció final la mateixa, no ens preocuparia en absolut que es generessin signatures pel nostre *stub*. De fet, part del *crypter* es podria dir que ja és polimòrfica, perquè els bytes encriptats del *malware* que conté l'*stub* poden haver estat encriptats utilitzant una clau diferent per al mateix executable. Però, la part que realment ens interessa convertir en polimòrfica és la que no està encriptada, la que conté el mètode d'injecció a memòria i altres tècniques d'ofuscació addicionals.

També seria interessant millorar la GUI de la *webapp* afegint-hi més opcions com, per exemple, donar a escollir a l'usuari entre diferents mètodes d'injecció a utilitzar pel seu *stub*, afegir també compatibilitat per a executables d'arquitectura x64, afegir-hi un *binder* (programa que rep d'entrada dos executables i els uneix en un de sol), poder clonar arxius *manifest* (metadades que descriuen el nom del software, número de versió, nom de l'empresa, llicència, descripció del producte, etc), entre moltes altres opcions.

En general, de manera resumida, el fet d'estudiar el funcionament intern dels *crypters* i haver-ne implementat un, ens ha servit com a base per seguir aprenent i desenvolupant sobre temes relacionats, per tal d'entendre encara millor com funciona el món de la seguretat informàtica en Windows i, en particular, el funcionament del *malware* sobre aquest sistema operatiu.

Referències

[1] Microsoft, *PE Format, 08/11/2020,* https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

[2] Microsoft, *Windows API Index, 05/31/2018* https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list

[3] AntiScan.Me, *Online Virus Scanner Without Result Distribution*, https://antiscan.me/

[4] Joxean Koret, Elias Bachaalany, *The Antivirus Hacker's Handbook*, 19/08/2015, pp. 7-13

[5] Wikipedia, *RC4*, 16/04/2020, https://w.wiki/wAv

[6] Wikipedia, *Base64*, 13/01/2021, https://es.wikipedia.org/wiki/Base64

[7] Mitre, *Process Injection: Process Hollowing*, 20/06/2020, https://attack.mitre.org/techniques/T1055/012/

[8] Red Teaming Experiments, *Process Hollowing and Portable Executable Relocations,* 2020, https://www.ired.team/offensive-security/code-injection-process-injection/process-hollowing-and-pe-image-relocations

[9] Microsoft, *LoadLibraryA function*, 12/05/2018, https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrarya

[10] BLEEPINGCOMPUTER, *Cybercriminals Undeterred by ToS For Remcos RAT*, 23/08/2018, https://www.bleepingcomputer.com/news/security/cybercriminals-undeterred-by-tos-for-remcos-rat/