Resumen

Todo empezó en 1959 con la creación de un juego basado en la teoría de Von Neuman de autómatas capaces de replicarse y desde entonces, los programas de moralidad dudosa no han hecho más que aumentar en cantidad y complejidad hasta crear lo que hoy conocemos como malware.

Los primeros en crearse fueron simples virus capaces de reproducirse, aumentar su población en una máquina y realizar alguna acción consumiendo recursos de esta, tal y como haría un virus biológico.

A continuación se crearon los worms (gusanos). Pequeños programas capaces de expandirse por la red entrando en máquinas y ejecutando en ellas algún tipo de código. Probablemente con el objetivo de infectar las máquinas con un virus.

Hasta el año 2004 estos eran los malwares que predominaban y sus creadores tan solo los hacían por diversión y obtención de reconocimiento. Sin embargo, a partir de ese año se dieron cuenta que estos programas maliciosos podían servir para algo más que obtener reconocimiento, podían servir para obtener dinero.

A partir de entonces se desarrollaron una gran cantidad de nuevas categorías de malware, y los programas maliciosos empezaron a ser mucho más complejos que los anteriores pudiendo englobarse en varias categorías de malware a la vez. Se desarrollaron los Backdoors, Troyanos, RATs, Spyware, Ransomware, Exploit Kits, Botnets, RootKits... cada tipo de malware cumplía con una característica que permitía que la unión de varios tipos de malware en uno solo hicieran de este un ejecutable malicioso complejo, difícil de detectar y capaz de realizar una gran cantidad de acciones sobre la máquina.

De este modo, las características de un virus dan a un malware la capacidad de infectar otros ejecutables con código malicioso; la característica principal de un gusano es que es capaz de expandirse por la red; un backdoor es cualquier código malicioso capaz de autoejecutarse sin la necesidad de intervención por parte del usuario; un troyano es un malware capaz de esconderse en un ejecutable benigno. Un RAT permite el control total de la máquina víctima mediante una conexión de esta con el atacante, las capacidades de un Spyware permite a un malware espiar al usuario de la máquina víctima, un ransomware es capaz de secuestrar un ordenador y pedir dinero a cambio. Los exploit kits intentan vulnerar la seguridad los programas clientes que se usan para navegar por la web y los rootkits sirven para ocultar las acciones del malware al usuario.

Desde hace años ya no es complicado encontrarse con malwares que reúnen todas esas capacidades y más, de forma que muchas veces el atacante tiene un control sobre la máquina víctima superior al que tiene el usuario.

El objetivo de la prueba de concepto desarrollada para este proyecto es darse cuenta de que un malware que reúne varias de estas capacidades no es tan difícil de crear.

Palabras clave

malware, virus, infectar, gusano, exploit, backdoor, caballo de troya, troyano, rat, spyware, bombas logicas, ransomware, exploitkit, botnet, bot, zombie, rootkit, antivirus, polimorfismo, metamorfismo, api windows, c++, keylogger

Summary

It all began in 1959 with the creation of a game based on Von Neumans's theory of automatas capable of replicating themselves and, since then, the dubious morality programs have increased in quantity and complexity until becoming what we now know as malware.

The first malicious programs created were simple viruses. They could reproduce themselves, increase their population in a machine and perform some actions consuming resources of the device.

Then, the worms were created. They were small programs able to expand through the network going into machines and running on them some kind of code, probably with the aim of infecting machines with a virus.

Until the year 2004 these were the main malwares created and their creators only built them for fun and to obtain recognition. However, this year they started to realize that these malicious programs could serve for more than gaining recognition, the could be used to obtain money.

Then, a large number of new categories of malware were developed and malicious programs began to be much more complex than the previous ones, being able to be included in several categories of malware at the same time. Backdoors, Trojans, RATs, Spyware, Ransomware, Exploit Kits, Botnets, RootKits, etc were developed. A malware who combines several types of malware could be able to perform a lot of actions in the machine and would be hard to detect.

The characteristics of a virus gives a malware the ability to infect other executables with malicious code; the main characteristic of a worm is that it is able to spread itself over the network; a backdoor is any malicious code capable of self-executing without needing the intervention of the user; a trojan is a malware that is hidden in a bening executable. A RAT allows full control of the victim machine through a connection of it with the attacker; the capabilities of a spyware allows a malware to spy on the user of the victim machine; a ransomware is able to hijack a computer and ask for money in return. The exploit kits try to violate the security client programs that are used to navigate through the web and the rootkits can hide the actions of the malware to the user.

Nowadays it is not difficult to find malwares that meet all those capabilities and more, so in many occasions the attacker has a control over the victims machine higher than the user.

The goal of the proof-of-concept developed for this project is to realize that malware that brings together several of these capabilities is not very complex to create.

Keywords

malware, virus, infect, worm, exploit, backdoor, troyan horse, troyan, rat, spyware, logic bombs, ransomware, exploitkit, botnet, bot, zombie, rootkit, antivirus, polymorphism, metamorphism, api windows, c++, keylogger

Índice

1. Introducción y objetivos

1.1 Introducción

La primera persona que imaginó el funcionamiento de un virus informático fue **Von Neumann** en 1949 cuando expuso La Teoría y Organización de Autómatas Complejos. En ella se mostraba la posibilidad de crear programas capaces de replicarse y tomar el control de otro programas. A pesar de que este concepto tenga muchas aplicaciones, es fácil observar que una de ella es lo que actualmente se conoce como virus informático (programas capaces de reproducirse).[1]

Sin embargo, no fue hasta 1959 cuando en los laboratorios de Bell Computer Robert Thomas Morris, Douglas McIlroy y Victor Vysottsky crearon el juego que sería el precursor de los virus informáticos: **CoreWar**. Este estaba basado en la anterior teoría de Von Neuman y el objetivo del juego era que los programas combatieran entre sí intentando ocupar toda la memoria de la máquina.[2]

En 1972, uno de los creadores del juego CoreWar, **Robert Thomas Morrison** creó el primer programa considerado virus informático. Este fue llamado **Creeper** e infectaba máquinas IBM 360 conectadas a ARPANET. Una vez infectada la máquinas emitía un mensaje en pantalla que decía “Soy un enredadera, atrápame si puedes”. Poco después de este virus también lanzó **Reaper**. El cual infectaba a las mismas máquinas buscando a Creeper y eliminándolo. Se podría considerar que Reaper es el precursor de los antivirus.

Poco a poco los PCs se fueron haciendo más populares gracias al aumento del número de personas que entendía la informática, lo útil que estaba empezando a ser visto usar un ordenador y a que cada vez era más sencillo utilizarlo.

Este uso masivo de PCs empezó a hacer posible ya en la década de los 80 que los virus pudiesen infectar a una cantidad de máquinas inimaginable hasta entonces.

Por ejemplo en 1981 **Richard Skrenta** escribe **Elk Cloner**, el primer virus de amplia reproducción. El cual mostraba un poema cuando detectaba que el equipo había sido arrancado 50 veces.

A pesar de que primer virus informático había sido creado ya hacía 12 años, no fue hasta 1984 cuando **Frederick B. Cohen** acuña por primera vez este término y lo define como “**Programa que puede infectar a otros programas incluyendo una copia posiblemente evolucionada de sí mismo**”.

En 1987 apareció uno de los virus más famosos de la historia, el virus **Jerusalem** o **Viernes 13.** Fue llamado Jerusalem pues fue reportado por primera vez desde la Universidad Hebrea de Jerusalem. Este virus infectaba los archivos .COM y .EXE que eran ejecutados mientras el virus residía en memoria y, en caso de ser viernes 13, eliminaba cada programa que se ejecutaba. Fue uno de los primero virus con mayor repercusión mediatica.[3]

En 1988 uno de los primeros gusanos(worms) informáticos ve la luz de la mano de **Robert Tappan Morris**. Su nombre era **The Morris Worm** y el objetivo de este gusano no era causar ningún daño, sino medir el tamaño de Internet. Para expandirse utilizaba varias vulnerabilidades conocidas de los ordenadores de la época y la prueba de contraseñas débiles en caso de que no la máquina no fuese vulnerable a ninguna de las vulnerabilidades. Sin embargo, este programa tuvo algunos fallos en su

programación que le permitía infectar varias veces una misma máquina llegando a consumir todos los recursos de esta y dejándola inservible.[4]

En 1999 aparece otro de los gusanos más famosos y cuya principal característica persiste hasta el día de hoy y cualquiera con una cuenta de correo electrónico la ha sufrido alguna vez. Este es el gusano **Happy,** creado por el francés **Spanska.** Este gusano se lograba expandirse por la red mediante su transmisión a través de mensajes de correo electrónico.

En el año 2000 se liberó un gusano que causó una infección que tuvo muchísima repercusión mediática debido a los daños ocasionados por la infección tan masiva que produjo. Fuel el gusano **I Love You**, el cual se basaba en técnicas de ingeniería social para infectar a los usuarios a través del correo electrónico.[5]

El 2004 fue el año álgido para este tipo de epidemias. En ese año aparecieron gusanos como el **Mydoom**, el **Netsky**, el **Sasser**, o el **Bagle**. El objetivo de este tipo de gusanos era alarmar a la sociedad realizando algún tipo de acción maliciosa en las máquinas que infectaban. Sus autores creaban este tipo se software buscando reconocimiento entre los grupos que se dedicaban a este ámbito.

A pesar de que el **2004** fue en año en el que más malware de este tipo se liberó, también fue el último. Esto fue debido a que los creadores de virus, gusanos… se dieron cuenta de que sus conocimientos se podían usar para algo más atractivo que ganar reconocimiento entre un reducido grupo de personas, **ganar dinero.**

Fue a partir de 2005 cuando el malware cuyo único objetivo era realizar alguna acción maligna sobre la máquina infectada fue desapareciendo y empezó a aparecer el malware más común en nuestros días. Aquel dedicado a **estafar** al usuario que infecta o a obtener datos de él con los que poder sacar beneficio económico.

Estos nuevos programas maliciosos seguían utilizando los mismos métodos de infección que los primeros virus y gusanos, usaban exploits, spam… Sin embargo, el código que corrían en la máquina víctima era completamente distinto.

Un buen ejemplo de este nuevo tipo de malware fueron los **troyanos bancarios**, cuyo objetivo era robar las credenciales que introducía el usuario para acceder a su cuenta del banco con el objetivo de que el atacante pudiese entrar en esta y robarle el dinero.

Otro de los tipos de malware que también proliferó fue el **spyware** que tenía objetivos muy similares a los troyanos bancarios, robar credenciales de acceso.

A parte de para los ordenadores que corrían el sistema operativo de Windows, también apareció malware para otras plataformas, como los **móviles**. Estos al principio estaban hechos sin tener demasiado en cuenta la seguridad del usuario, lo que permitió al software malicioso hecho para infectar estos dispositivos expandirse a través de bluetooth o de mensajes **MMS** y poder robar dinero al dueño del móvil infectado.

A partir de esta época los softwares maliciosos que crean y utilizan los cibercriminales se han ido perfeccionando permitiendo crear campañas de estafas e infección de víctimas de forma más sutiles y complejas, a la vez que el uso de estos malwares es cada vez más sencillo.

1.2 Objetivos

Este proyecto tiene como objetivo realizar un estudio de los tipos de malware que se han ido desarrollando a lo largo de la historia de la programación para luego desarrollar una prueba de concepto de malware que realice las acciones más típicas de estos.

En el siguiente apartado se detallarán las características fundamentales de cada tipo de malware. También se explicará cómo realizan dichas acciones y, en caso de existir, se hablará de varios subtipos que un malware puede tener.

Estos subtipos siempre cumplirán con la característica general del malware del que proceden pero además tendrán una característica especial que les hace diferenciarse dentro de su categoría software malicioso.

También se comentarán diversas técnicas que utilizan muchos programas maliciosos para evitar ser detectados por el antivirus y el usuario, así como las bases que un usuario debe conocer para defenderse de los malwares.

Además, como se podrá deducir del siguiente apartado, a un programa malicioso no se le puede clasificar tan solo dentro de una categoría de malware sino que todos los programas maliciosos que se crean actualmente cumplen con las características fundamentales de varios de los tipos explicados.

Esto se entenderá mucho mejor con la prueba de concepto creada. Esta puede realizar acciones que le permiten ser considerada perteneciente a más de cinco de los tipos explicados.

En los siguientes apartados será explicada detalladamente las acciones que puede realizar dicha prueba de concepto y, para terminar, existirá un apartado de conclusiones del proyecto realizado.

1.3 Estructura de la memoria

La estructura de la memoria presentada es lo más sencilla y lógica posible. Ha comenzado con resumen de toda ella y a continuación se ha presentado una introducción a la historia del malware. Después del capítulo introductorio se encuentra el apartado de la clasificación de los distintos tipos malware más comunes, para, una vez entendidos los distintos tipos y sus características, encontrarse con la explicación detallada de la prueba de concepto creada dividida en dos capítulos: Diseño y Desarrollo.

Finalmente la memoria termina con un pequeño capítulo de las conclusiones obtenidas tras la realización del proyecto.

2. Análisis y Categorización del Malware

Debido a que el malware puede clasificarse de distintas formas según distintas características, se han seguido las clasificaciones que se realizan en el libro *Malware: Fighting Malicious Code* escrito por *Lenny Zeltser* que recoge las clasificaciones más aceptadas[6]. Además, se han añadido otros tipos de malware que aún no eran muy comunes en la fecha de escritura del libro.

2.1. Virus

La característica principal que define a un virus es el ser capaz de infectar otros archivos con algún tipo de malware de forma que cuando el usuario quiera ejecutar el archivo infectado este ejecute el malware introducido y puede que también el código original del programa.[7, 8]

Según cómo cumpla con la característica principal podemos encontrar varios tipos de virus:

2.1.1. Companion viruses

Este tipo de virus suplanta un archivo ejecutable o lo infecta de formas muy sencillas.

Una forma de hacer esto es llamar al virus por el mismo nombre que un programa muy usado por la víctima y situarlo en el mismo directorio que dicho ejecutable con el atributo hidden y la extensión *com*.

Un ejemplo muy bueno es el de guardar un malware llamándolo *notepad.com* (usando la extensión com) y ponerle al archivo la flag de hidden. De forma que la víctima que ejecutase el programa desde la terminal sin poner su extensión ejecutaría por defecto el de la extensión *.com*.

Otra forma de conseguir dicho objetivo de una forma aún más sencilla consiste en llama el virus como el programa original, renombrar el original y poner el atributo hidden al original, moverlo a otro sitio, o incluso se eliminarlo. De esta forma el usuario siempre ejecutaría el malware.

Un último método consiste en infectar un ejecutable de forma muy fácil usando alternate data streams (ADS). La forma de guardar ejecutables de NTFS permite guardar ejecutables en diferentes streams. Por lo que cuando se llama al ejecutable se va accediendo a los diferentes streams y ejecutando el código que ahí se encuentra. Un ejecutable normal de Windows suele tener un único stream donde se encuentra el código ejecutable. Sin embargo, un virus puede poner su código en el primer stream y mover el código del ejecutable a un stream secundario, que será ejecutado después del código malicioso.[9]

Infectar un programa de esta forma es muy sencillo, simplemente basta con ejecutar en la consola de comandos de windows:

type c:\windows\notepad.exe > c:\windows\System32\calc.exe:hidden.exe

Para iniciar el ejecutable contenido en el ADS bastaría con hacer: **start c:\windows\System32\calc.exe:hidden.exe**

2.1.2. Infección mediante sobreescritura

Se abre con permisos de escritura el programa que se desea infectar y se sobreescribe el código del virus. De esta forma cuando la víctima ejecute el programa se ejecutará el malware introducido e inmediatamente después el ejecutable le avisará de que ha habido un problema, sin embargo, el código malicioso ya habrá corrido en la máquina.

2.1.3. Infección mediante añadido

El virus se coloca al final del ejecutable de forma que al principio del código del programa real se pone un JMP para que se salte al código malicioso introducido al comenzar la ejecución del ejecutable, y posteriormente el malware pasa el control al programa original mediante otra instrucción JMP.

Otra forma de hacer esto mismo es, en vez de colocar un JMP al principio del código del ejecutable, modificar el EntryPoint de la cabecera de este para que al iniciarse lo primero que ejecute sea el código malicioso añadido y que de ahí salte al EntryPoint original mediante un JMP colocado al final del código malicioso.

2.1.4. Infectar Boot Sectors

Al iniciar el PC la BIOS inicia el hardware, posteriormente localiza el primer sector del primer disco y ejecuta un pequeño programa llamado master boot record (MBR). El código del MBR sirve para enumerar las particiones disponibles y transferir el control al boot sector de la partición preparada para iniciar el sistema operativo. El Boot Sector está al principio de de la partición (llamado partition boot sector o PBS). El PBS inicia los archivos para levantar el sistema operativo seleccionado. En la figura 1 se puede ver un resumen del camino seguido al arrancar un PC:

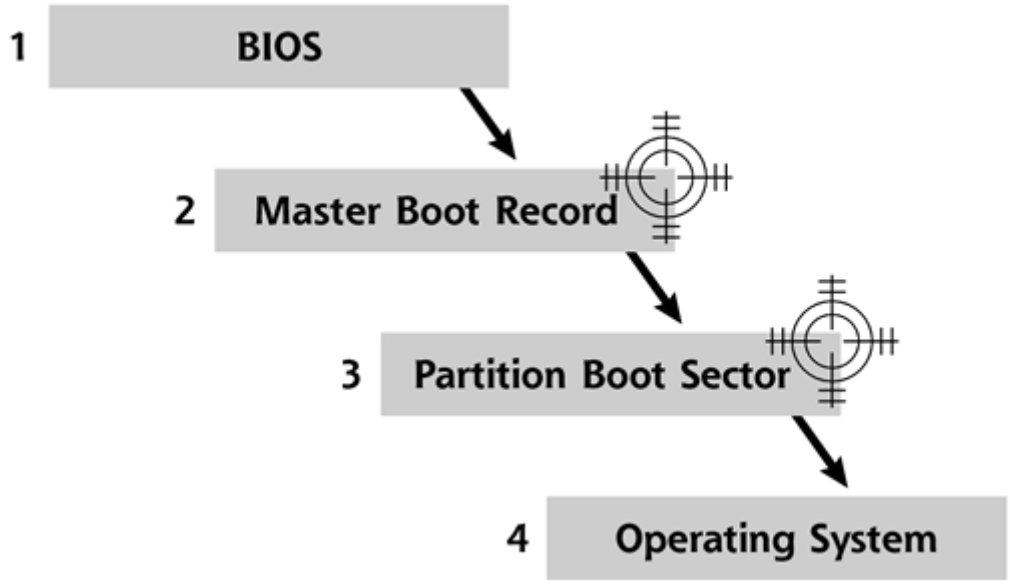


Figura 1 - Inicio de un PC

De esta forma los virus llamados boot sector infectan la MBR o algún PBS.

Estos virus son ahora menos efectivos pues desde hace tiempo cuando Windows se carga no permite ejecutar acciones al código situado en estos sitios. Sin embargo, no

dejan de poder ser letales ya que podrían eliminar todos los archivos al arrancar el PC antes de que se inicie el sistema operativo.

2.1.5. Infectar documentos

Este tipo de virus infectan archivos a los que se puede añadir código ejecutable en forma de macros. El ejemplo más típico es añadir las macros en documentos de microsoft word usando principalmente las funciones Document_Open() y Document_Close().

Debido a que este tipo de archivos tiene permiso para ejecutar cualquier tipo de código se les puede infectar para que realicen cual quiera acción. Sin embargo, lo más normal es que se les infecte de forma que cuando sean ejecutados se descarguen un malware desde internet y lo ejecuten.

2.1.6. Multi-Virus

Este tipo de virus es capaz de infectar a varios tipos de archivos utilizando varias de las técnicas ya vistas.

2.1.7. Otros

Como se ha dicho al inicio del capítulo, un virus es cualquier código malicioso que consigue infectar otro código para que ejecute unas órdenes dadas por el virus siempre que el programa infectado se ejecute.

Se pueden infectar scripts de VBS, perl, shell, java y otros lenguajes muy comunes de programación. De forma al infectar dichos scripts y si estos no son revisados frecuentemente pero sí se ejecutan de forma automática, correrán el código malicioso que se les haya inyectado siempre que se ejecuten.

No solo tienen por qué poder infectar scripts, también pueden infectar binarios ya compilados, jar, o de otros tipos.

2.2. Worms

La característica principal de un worm es que sea capaz de propagarse por medio del uso de cualquier red. Por lo tanto, todos los ataque informáticos que usan como vector de propagación una vulnerabilidad descubierta en algún tipo de equipo conectado a internet se puede considerar que son worms, además de otros tipos de malware, según lo que hagan.[10, 11]

Otro tipo de worms que antes eran muy comunes son aquellos que se enviaban a sí mismos en correos electrónicos desde la cuenta de una víctima que había sido infectada por un malware a sus contactos.

En la figura 2 se muestran los componen que forman un Worm:

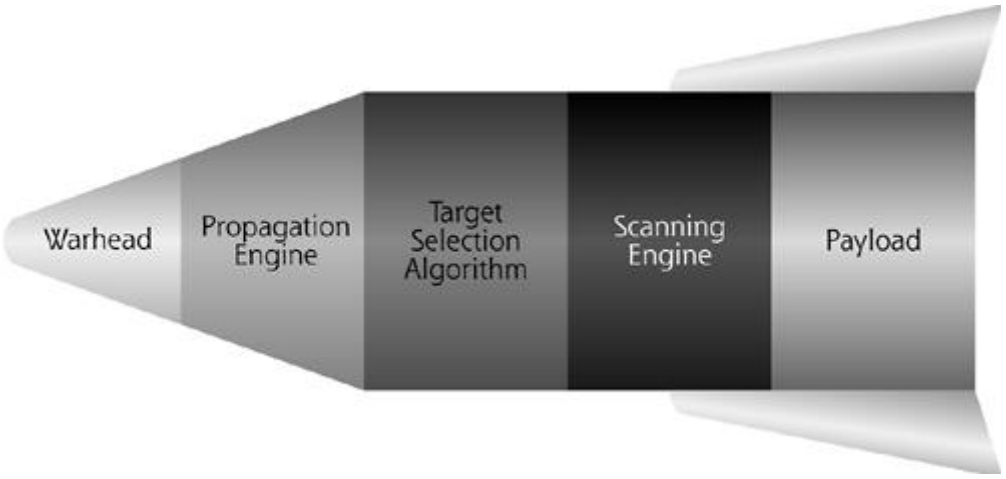


Figura 2 - Componentes de un Worm

- **Warhead:** Es el método que utiliza el worm para conseguir entrar en la máquina víctima. Normalmente se suele usar un exploit para lograr esto.

Además, debido a que la seguridad aún no es tenido en cuenta como algo importante en muchas empresas, un exploit conocido hace meses puede seguir entrando en muchos equipos pues nadie los habrá actualizado aún.

Algunos de los warheads más usados son:

  - Buffer Overflows Exploits
  - File-sharing Attacks: Métodos de compartición de archivos mal configurados. El worm sobreescribe un programa que luego el usuario ejecutará o que se ejecuta automáticamente.
  - E-mail: Necesita que el humano receptor lo ejecute, o algunos e-mail readers son vulnerables y con un exploit se puede hacer que ellos ejecuten el worm.
  - Other Common Misconfiguration: Por ejemplo ataques de diccionario para entrar en sistemas probando distintas combinaciones de nombres de usuario y contraseñas.

- **Propagation engine:** Es el método que usa el worm para propagarse. Generalmente suele usar alguno de los siguientes protocolos: FTP, HTTP, SMB, SMTP, Telnet, SSH.

- **Target selection algorithm:** Es el algoritmo que se usa para seleccionar máquinas a las que intentar infectar. Este algoritmo puede elegir máquinas dentro de una lista de máquinas que ya han sido previamente escaneadas y se sabe que son vulnerables, o puede elegir máquinas conectadas a internet para escanearlas y comprobar si son vulnerables.

Otras de las formas más comunes de seleccionar objetivos a parte de las ya comentadas son:

  - Direcciones e-mail que encuentra en la máquina víctima.
  - Lista de hosts que haya en /etc/hosts o LMHOSTS.
  - Los sistemas confiables que se pueden encontrar en /etc/hosts.equiv o .rhosts. Son hosts a los que se puede acceder sin credenciales.
  - Sistemas activos en la red local buscando con NetBIOS y SMB.

- **Scanning Engine:** Busca vulnerabilidades en equipos conectados a una red a la que el ordenador atacante pueda acceder. En concreto, busca vulnerabilidades que el worm programado pueda usar para acceder a estos equipos y ejecutar código.

- **Payload:** Es el código que va contenido en el worm y que ejecutará en cuanto este consiga entrar en una máquina vulnerable. Debido a que el malware ya se encuentra dentro del equipo víctima, se puede ejecutar cualquier tipo de código. Sin embargo, para que el worm no sea muy grande, se suelen usar payloads pequeños que realicen una acción que permita al atacante ejecutar más código de forma sencilla. Por ejemplo, puede abrir un backdoor al que pueda acceder el atacante, puede ser código que descargue un archivo malicioso más grande y lo ejecute, puede ser un agente que esté a la escucha para realizar ataques de DDoS...

2.2.1. Cómo evitan dificultades y limitaciones

Debido a que no es seguro que el ordenador víctima contenga las funciones que necesita el payload para funcionar,pueden llevar todas las librerías y software que necesitan.

También pueden estar programados para poder propagarse de varias formas.

Tienen que tener cuidado de no ser muy pesados pues sino podrían llegar a crashear el sistema, o usar demasiado ancho de banda.

Deben comprobar antes si ya han infectado una máquina antes de ejecutar el payload, sino, puede ser que una máquina esté siendo constantemente atacada y se paralice antes incluso de que algún worm haya podido ejecutar su payload.

También deben tener cuidado que otros worms no entren usando la misma vulnerabilidad y deshabiliten el worm. Una buena forma es eliminar la vulnerabilidad una vez el worm se haya hecho con el control.

Según las acciones que realice el worm se pueden encontrar distintas características en estos:

2.2.2. Worms de multiplataforma

Son aquellos worms que están programados y preparados a para poder atacar a distintas plataformas. Para ello deben tener preparados varios payloads a ejecutar en función de la plataforma afectada.

2.2.3. Worms de multiexploit

Son aquellos worms capaces de usar varios warheads a través de varios propagations engines. Es decir, están preparados para entrar en sistemas pudiendo explotar varias vulnerabilidades.

2.2.4. Zero-Day Exploit Worms

Son aquellos worms que utilizan un exploit día 0 para aprovecharse de una vulnerabilidad y entrar en el sistema.

Una exploit día 0 es un exploit que se aprovecha de una vulnerabilidad para la cuál aún no existe solución.

2.2.5. Worms de expansión veloz

Son aquellos que usan la técnica Warhol o Flash para su propagación.

Esta técnica consiste en que antes de comenzar la expansión del malware, el atacante escanea internet en busca de equipos vulnerables. Una vez termina, comienza a atacarlos de forma que cada vez que infecta a uno divide las direcciones de los demás servidores vulnerables entre ambos, de esta forma el worm se expande a una velocidad exponencial.

Se suele empezar por infectar a los sistemas con más ancho de banda y preferiblemente cercanos al Backbone de Internet, pues estos sistemas podrán infectar a los que se les asigne de forma más rápida.

Usando esta técnica de infección es posible infectar a todos los hosts vulnerables de internet en 15 minutos o menos. Lo cual, comparado con los varios días que requería previamente a un worm infectar a todos los host vulnerables, supone un gran aumento de velocidad de expansión.

2.2.6. Worms éticos

Eran worms que aprovechaban las vulnerabilidad para infectar sistemas y aplicar el parche que corregía dicha vulnerabilidad. Una vez realizada la acción la buena acción, solían preguntar al usuario si le permitía escanear otros sistemas en busca de dicha vulnerabilidad para corregirla y eliminarla de la red de la forma más rápida.

Sin embargo, estos worm dejaron de crearse pues en algunos casos el parche modificaba algo que el software del servidor necesitaba para ofrecer su servicio y la aplicación del parche acababa provocando un ataque de denegación de servicio.

2.3. Backdoors

Se puede considerar backdoor a cualquier malware capaz de conseguir permitir al atacante tener acceso a la máquina víctima de forma muy sencilla, saltándose todas las restricciones de seguridad que dicha máquina pudiera tener. [12]

Muchos malwares incorporan características de backdoor para no perder acceso al sistema de forma que tan solo sea necesario vulnerarlo una vez para tener siempre acceso a él.

Estos tipos de malware suelen ubicarse en partes del sistema operativo o en archivos que sean ejecutados de forma automática cuando se cumpla algún tipo de condición, la más típica, que la máquina esté encendida.

2.3.1. Windows

Siempre que Windows se inicia ejecuta distintos tipos tipos de archivos por varios motivos. Poco a poco los creadores de malware han ido siendo más cautelosos a la hora de decidir dónde colocar el malware para que se ejecute solo pero de forma que la víctima tenga más complicado saber que se ha ejecutado, y cómo.

La forma más sencilla de ejecutar un programa al encender Windows es ubicándolo en la carpeta StartUp, el path completo a esta carpeta se puede encontrar en: C:\Users\NOMBRE_USUARIO\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

Cualquier programa que se coloque en esta carpeta en concreto será iniciado en cuanto dicho usuario entre en el sistema.

Otra forma muy usada de ganar persistencia es escribir la dirección dónde se encuentra el malware en algún registro que cargue los programas inscritos al encender el equipo. Hay muchos registros que realizan dicha acción, los más conocidos y usados son:

- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce
- HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce

A pesar de que estos son los más típicos e intuitivos de usar a la hora de ganar persistencia, hay decenas de registros en los que poder escribir para ejecutar código cuando el ordenador se inicie, por ejemplo:

- HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components
- HKLM\SOFTWARE\Wow6432Node\Microsoft\Active Setup\Installed Components
- HKLM\Software\Classes*\ShellEx\ContextMenuHandlers
- HKLM\Software\Classes\Drive\ShellEx\ContextMenuHandlers
- HKLM\Software\Classes\Directory\ShellEx\ContextMenuHandlers
- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Font Drivers
- HKLM\Software\Microsoft\Windows NT\CurrentVersion\Drivers32
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\Credential Providers

Otra forma muy típica para ganar persistencia es la creación de un servicio que ejecute el malware al iniciar el ordenador. Esta forma es un poco más sigilosa que las anteriores ya que, a pesar de seguir escribiendo el path del malware en un registros de windows, al iniciarse el servicio (es decir, correr el malware), este no tendrá que correr en su propio proceso sino que podrá hacerlo en un proceso de srvhost.exe, permitiendo que el malware se ejecute de forma más sigilosa.

En caso de que el malware pueda ganar persistencia usando una Dll, a parte de varias de las formas anteriores, también puede ganar persistencia y cargarse en casi todos todos procesos del sistema (todos aquellos que carguen User32.dll) incluyéndose en la lista de Dlls llamada AppInit_Dlls situada en el registro de windows.

Para que el malware gane persistencia no siempre necesita ser ejecutado cuando se inicie el sistema o entre un usuario en él. También se pueden programar tareas que ejecuten el malware cuando se den condiciones más específicas.

Por ejemplo, se puede programar una tarea que ejecute el malware en una hora determinada de un día determinado, que lo ejecute cada X tiempo, o que lo ejecute cuando algún otro tipo de condiciones ocurra. Esto se consigue de manera muy sencilla usando el servicio de tareas programadas que ofrece Windows.

Todos los ejecutables que se iniciarán al iniciar Windows de la forma comentada hasta ahora y mediante alguna de las formas que se comentará a continuación pueden obtenerse mediante el programa Autoruns.exe de SystemInternals proporcionado gratuitamente por Windows.

A pesar de que la lista de ejecutables que ofrece este programa es muy completa, existen formas más sigilosas de iniciar automáticamente un malware sin que aparezca en dicha lista.

Por ejemplo, ocultando su inicialización en archivos que sirven para configurar el sistema tras su arranque como:

- **Win.ini**: Este archivo puede ser modificado para que corra el malware al ser leido. Para ello bastaría con escribir una línea como run=[PATH_MALWARE] o load=[PATH_MALWARE], o configurarlo de forma que cada vez que se quiera abrir un archivo con una extensión determinada (como .doc o .txt) se ejecutase el malware.
- **System.ini**: Este archivo realiza funciones similares al anterior y también se puede modificar para que inicie algún ejecutable al encender Windows.

Otra forma cada vez más común de ganar persistencia usando una Dll consiste en la suplantación de de alguna Dll que cargue un programa que se ejecute al iniciar Windows, como explorer.exe.

Esta técnica se basa en el orden de paths que sigue Windows a la hora de cargar una Dll en un proceso que la importa. El orden es el siguiente:

1) El directorio donde reside el ejecutable
2) El directorio desde donde se invoca al ejecutable
3) El System Directory (C:\Windows\System32)
4) El System DIrectory de 16 bits (C:\Windows\System)
5) Los directorios listados en la variable de ambiente PATH

De esta forma se observa muy fácilmente que si explorer.exe carga una Dll que no esté en el mismo directorio y un malware crea una Dll con el mismo nombre y la guarda en el directorio donde reside explorer.exe, este cargará dicha Dll en vez de la legítima.

Para evitar dicha vulnerabilidad se crearon las **KnowDlls**. Son DLLs que están en el registro HKLM\System\CurrentControlSet\Control\Session Manager\KnownDLLs y que siempre que un programa quiera cargar una de ellas, la cargará de la carpeta System32. De esta forma, las DLLs listadas en dicho registro ya no son vulnerables a la suplantación.

Además, todas las Dlls que cargan las KnownDlls también serán cargadas desde la carpeta System32. De forma que también se evita la suplantación de las Dlls que cargan las KnowDlls.

Sin embargo, no todas las Dlls que cargan los ejecutables que son iniciados al arranque del sistema están dentro de la lista de KnownDlls, y las Dlls que cargan las

Dlls cargadas por alguna KnownDll tampoco. Por lo que suplantando cualquier Dll de alguno de estos dos tipos se consigue que la Dll maliciosa sea cargada al arrancar el sistema.

2.3.2. Unix

En caso de los sistemas operativos basados en Unix también existen varias formas de que un malware consiga persistencia, aunque hay menos formas que en el caso de Windows debido a que no existe el registro.

La forma más típica consiste en modificar archivos que se van a ejecutar al arrancar el sistema. Por ejemplo, el primero proceso en ejecutarse es el demonio **init** que lee del archivo /etc/inittab para saber qué procesos arrancar. Por lo que muchos malwares modificarán este archivo para que el malware sea una de las primeras cosas que se ejecuten en el sistema.

Otros archivos que corren automáticamente se encuentran en /etc/rc.d y /etc/init.d. Estos también podrían ser susceptibles de ser manipulados con el objetivo de que ejecutasen el malware.

Lo mismo ocurre con los servicios programados para ser iniciados poco después de cargar el sistema operativo, podrían ser modificados para que ejecutasen el malware.

Además, muchos de estos servicios leen archivos de configuración que también podrían ser modificados con el mismo propósito.

Otra forma típica más específica con la que muchos malwares consiguen ganar persistencia consiste en modificar el demonio **inetd** para que cuando reciba tráfico por algún puerto UDP/TCP active el código malicioso.

Finalmente, una última forma también muy utilizada por los creadores de malware para ganar persistencia en equipos basados en unix consiste en crear tareas programadas con **cron**. De forma que se especifica una serie de parámetros (como día, hora, mes...) de forma que cuando se cumplan, se ejecutará el malware. Aunque lo más normal es programar una tarea que corra el malware al iniciar el ordenador.

2.4. Troyanos

Este tipo de malware se caracteriza por aparentar ser un programa benigno que ofrece un servicio legítimo pero que en realidad no es así. De hecho algunos troyanos ejecutan primero el código malicioso y luego el esperado por el usuario para que este sospeche lo menos posible, pero hay otros que solo se hacen pasar en apariencia por un programa confiable pero que al ser ejecutados tan solo corren el código malicioso.[13]

Una forma muy común de crear troyanos en Windows es hacer pensar al usuario que el archivo tiene una extensión distinta de la que realmente tiene.

Esto se consigue de forma muy sencilla cambiando el nombre de un ejecutable por ejemplo de “calc.exe” a “calc.txt.exe” y como Windows por defecto no muestra la extensión del archivo el nombre que mostrará será “calc.txt”. Un usuario podría sospechar de que ese archivo mostrase la extensión y los demás no, pero si además

se modifica el icono de dicho archivo al típico icono que representa a los archivos ".txt" probablemente ni lo pensara.

Otra forma muy común de crear troyanos que aparenten ser programas legítimos pero que luego ejecuten algún tipo de código malicioso consiste en hacer uso de las técnicas de infección de ejecutables vistas en el apartado de los virus. De esta forma se puede llegar a la conclusión de que la labor principal de un virus es la creación de troyanos en el ordenador víctima donde se ejecute el virus.

Otra forma muy usada por creadores de malware para atacar Windows consiste en situar troyanos con el nombre de comandos muy utilizados (como dir) en paths desde donde el usuario suela ejecutarlos, como su carpeta de usuario o el escritorio. De forma que cada vez que el usuario llame a dicho comando desde dicha carpeta, se ejecutará el troyano ahí guardado. Es una forma muy común de escalar privilegios en Windows.

Este tipo de troyanos no funcionan en Unix pues para ejecutar un programa que está en la misma carpeta que la consola se necesita indicarlo con "./".

A la hora de crear un troyano, los creadores de malware también tienen en cuenta que en caso de que este se ejecute en su propio proceso, es importante que el nombre de este proceso levante las mínimas sospechas posibles. Por ello es común es llamar al troyano con nombre de un servicio normal como init, inetd, corn, http, etc de forma que el usuario vea normal que este proceso esté siendo ejecutado.

Aunque el hecho de que haya dos procesos con el mismo nombre sí podría levantar sospechas, hay algunos procesos que de forma normal tienen el mismo nombre.

Finalmente, la última forma muy típica de crear troyanos es mediante el uso de wrappers, también llamados binders, packers o exe joiners.

Estos toman dos ejecutables, generalmente uno que el usuario esté acostumbrado a ver y en el que confíe, y otro malicioso. Una vez elegidos estos archivos, el wrapper se encarga de crear otro programa que tenga la apariencia del ejecutable confiable pero que a la hora de ser ejecutado cree dos procesos, uno del ejecutable esperado y otro del malicioso.

En estos casos se suele nombrar al ejecutable malicioso con un nombre que no llame la atención al crear su propio proceso, tal y como se ha explicado anteriormente.

Algunos wrappers típicos son: AFX File Lace, EliteWrap, Exe2vbs, PE Bundle, Perl2Exe, Saran Wrap, TOPV4, Trojan Man.

2.4.1. Esteganografía

La esteganografía también puede ser considerada un forma de creación de troyanos, pues se está introdución información inesperada en sitios que en principio podrían considerarse benignos como imágenes, vídeos, programas...

Por ejemplo Hydan es un programa que coge ejecutables para x86 y realiza cambios en estos de forma que no altera la funcionalidad original del programa pero que introduce dentro de este información (generalmente otros programas cifrados) de forma que se puede usar Hydan posteriormente con la clave de cifrado para obtener el mensaje/programa secreto contenido en el ejecutable modificado.

Hydan funciona modificando instrucciones de suma por instrucciones de resta de números negativos cuando sea necesario. De forma que codifica la información introducida dentro de un programa por medio del uso de sumas o restas.

Además, esta es una forma muy primitiva pero en ocasiones útil de crear malware polimórfico que modifique la firma que se obtiene de calcular el hash de un ejecutable.

Más adelante se detallará qué es un malware polimórfico.

2.5. RAT

Son las siglas de Remote Access Trojan.[14]

Este tipo de malware ofrece al atacante una conexión que le permite controlar completamente el ordenador víctima.

Para esto lo más típico es que la máquina víctima establezca una conexión directa con el atacante en un puerto confiable como puede ser el 80 o el 443.

Sin embargo, este tipo de conexiones son de las más fáciles de detectar, por lo que si se busca un método más sigiloso de controlar la máquina víctima se suele usar una de las siguientes técnicas.

2.5.1. ICMP

Algunos malware para comunicarse con el atacante no usan ni siquiera un puerto. Simplemente utilizan un protocolo de comunicación típicamente soportado por casi todas las máquinas en el cual se puedan introducir mensajes.

El malware estará atento a mensajes de dicho protocolo para leerlos, tomar la acción necesaria y, probablemente, enviar otro mensaje de respuesta.

Para este tipo de comunicación el protocolo elegido más utilizado es el ICMP. Por ejemplo, el atacante enviará un mensaje ICMP echo request en el que el campo de datos introduzca el mensaje “get_passwords”. Cuando este mensaje llegue a la máquina, el malware lo leerá, ejecutará la acción correspondiente con ese mensaje y enviará la respuesta probablemente mediante otro mensaje ICMP, aunque si se usasen vías distintas de comunicación para enviar peticiones y respuestas, el malware sería aún más sigiloso.

2.5.2. Port Knocking

En caso de que al atacante le guste tener una conexión directa con la víctima, existen métodos para que esta conexión se cree cuando el atacante lo necesite, por ejemplo se puede programar que al enviar 3 paquetes a 3 puertos distintos el malware crea una conexión inversa.

Un ejemplo de este tipo de acción se puede encontrar en servidores accesibles desde internet que para que permitan la conexión por ssh a una dirección IP esta antes tienes que enviar ciertos paquetes a ciertos puertos. La siguiente figura muestra un explicación más intuitiva de esta táctica de acceso.

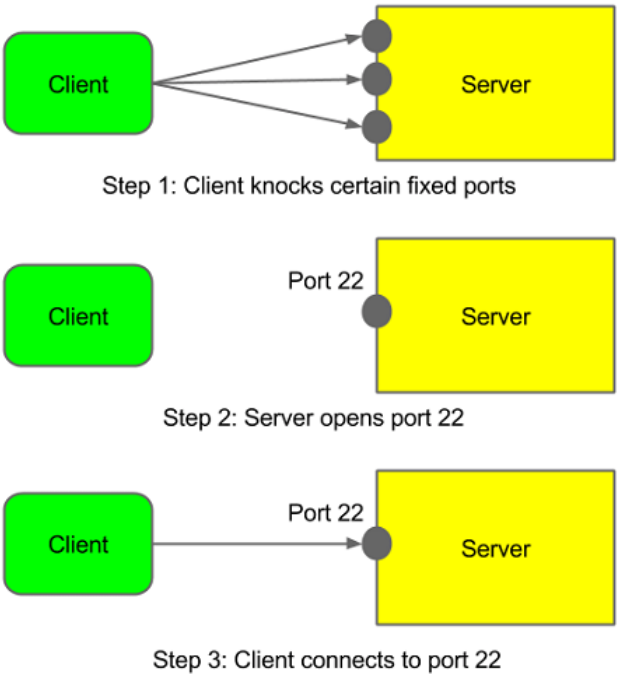


Figura 3 - Explicación de Port-Knocking

2.5.3. DNS

También existe la posibilidad de que el equipo víctima pueda recibir paquetes por la red, pero no pueda enviar casi ningún tipo de paquete. En esos casos, es normal que sí le esté permitido enviar por lo menos peticiones DNS. Esto lo saben muchos creadores de malware y para asegurarse de que podrán recibir respuesta de las máquinas infectadas pueden programar que estas respuestas sean peticiones DNS a un servidor DNS controlado por ellos, y que en estas peticiones vengan las respuestas a las órdenes enviadas.

2.5.4. Modo Promiscuo

Estos métodos sigilosos de control del equipo de forma remota son aún más interesantes si el malware logra poner la tarjeta de red de la víctima en modo promiscuo y puede escuchar todos los paquetes que llegar a la red en la que está conectado.

De esta forma, no solo no habría que abrir una conexión directa en el ordenador atacante y la víctima pues con solo enviar un paquete al ordenador víctima el malware podría leer su contenido, ejecutar la acción ordenada y enviar una respuesta. Sino que ni siquiera haría falta enviar un paquete al ordenador víctima. Se podría enviar un mensaje a cualquier otra máquina que estuviese en la misma red que la víctima y el malware también podría leer en contenido del mensaje.

Además, el malware podría responder con paquetes en los que modificara la IP de origen escribiendo la IP a la que iba dirigido el paquete del atacante. De esta forma, un observador que viera dichos paquetes y sospechara de ellos, pensaría que es la máquina cuya IP es suplantada la que está infectada con el malware.

2.5.5. Descubrir sniffers

Las formas más comunes para descubrir si alguna tarjeta de una red está actuando en modo promiscuo son:

- Enviar en un paquete una IP como 10.1.1.1 que no sea de nadie para ver si alguien pregunta al DNS por esa IP. Si alguien le pregunta, probablemente esté sniffeando.
- Enviar un paquete ping al sniffer pero con la MAC mal puesta. Si tiene la tarjeta en modo promiscuo no le importará que la MAC esté mal y responderá al ping.
- Enviar un ARP Request a una dirección que no exista. Si hay algún sniffer es probable que quiera hacerse para por dicho ordenador.

2.6. Spyware

Se puede considerar que todo malware capaz de espiar al usuario de la máquina en la que está siendo ejecutado tiene características de spyware.

Las acciones más habituales de los spywares son las de guardar las teclas del teclado que son pulsadas (keylogger), obtener capturas de pantalla y guardar el sonido y vídeo que capte por el micrófono y cámara de la máquina.

Estos métodos son tan habituales en los malwares que es común encontrarse con pegatinas tapando la cámara que viene incorporada en los portátiles.

Este tipo de malware suele estar casi siempre incorporado en un RAT. Esto es debido a que por si solo un spyware puede guardar información en el ordenador víctima pero no enviarla al atacante. Además, este tipo de utilidades suelen ser útiles a los atacantes para conocer mejor la víctima.

A pesar de esto, sí que es verdad que existen distintos programas que pueden actuar de spyware y enviar toda la información obtenida al atacante sin poder realizar ningún otro tipo de función. Estos programas vendrá también con características de backdoor para que se ejecute el spyware al arrancar el ordenador.[15]

2.7. Bucles infinitos

Las bombas lógicas consisten en código que entra en un bucle infinito y que va demandando cada vez más recursos del ordenador de forma que este al final no puede soportar el proceso de la bomba y se queda paralizado.

Estas bombas pueden estar programadas para abusar especialmente de un tipo de recurso o varios. Por ejemplo, puede realizar cada vez más operaciones complejas de forma que vaya necesitando más y más capacidad del procesador hasta que lo sature.

Puede que en vez de saturar el procesador vaya ocupando poco a poco toda la memoria RAM hasta que la máquina no pueda funcionar.

U otra posibilidad es que vaya creando y guardando grandes archivos hasta ocupar todo el espacio del disco y que el ordenador no pueda continuar funcionando.

Es la forma de malware que probablemente sea más fácil de crear y su único objetivo es producir un ataque de denegación de servicio en la máquina afectada.

2.8. Ransomware

Este tipo de malware es el que más ha proliferado en los últimos años en parte gracias a la sencillez con la que se crea y en parte gracias al asentamiento y confianza generada en las monedas digitales.

El ransomware es un tipo de malware que secuestra el ordenador y pide un rescate a cambio de liberarlo. [16]

Los primeros ransomwares eran programas que no permitían al usuario realizar ninguna tarea hasta que cumplieran con el rescate[17]. Los más usados actualmente cifran todos los archivos de una extensión seleccionada y de unas carpetas elegidas y se utilizan algoritmos de cifrado para los que aún no se conocen debilidades de forma que la única forma de conseguir descifrarlos es con la contraseña usada.

Cuando el ransomware actúa y termina de cifrar toda la información suele mostrar en pantalla explicaciones para el usuario sobre cómo conseguir la contraseña que se usó para el cifrado. Normalmente, pide hacer una transferencia de una moneda digital a un monedero controlado por el atacante. Un ejemplo de la pantalla que muestra un ransomware lo podemos encontrar en la siguiente imagen.

Imagen 1 - Ejemplo de secuestro de ransomware

Por lo tanto el ransomware actual tiene una debilidad clara, necesita una contraseña con la que cifrar la información y esta no se puede perder pues sino sería imposible descifrarla. Para ello los creadores de este tipo de malware tienen varias opciones: guardar la contraseña dentro del malware e intentar esconderla lo mejor posible de la ingeniería inversa, establecer algún tipo de comunicación con el malware por la cual enviar la contraseña a usar, programar el malware para que cree una contraseña cuando la necesite y que la guarde en algún sitio difícil de encontrar del ordenador, que genere la contraseña el malware y la envíe al atacante para que este la guarde, o que genere una contraseña cuando la necesite y luego la elimine.

En este último caso no habría forma de recuperar la información cifrada y es muy posible que se corriera la voz de que dicho ransomware no permite recuperar la información ni aunque se pague lo que se pide, por lo que los creadores de

ransomware ya no suelen eliminar la contraseña. Aunque sí que es posible que en caso de transmitirla por la red, se pierda la contraseña y no se pueda recuperar tampoco la información, por eso nunca se recomienda pagar el rescate de un ransomware.

Gracias a esta debilidad en los ransomware al cabo de un tiempo siempre suele aparecer un programa que consiga la contraseña que utilizó un determinado tipo de ransomware para cifrar la información. Normalmente esta se consigue realizando ingeniería inversa al malware y obteninedo la contraseña que tuviese guardada, dónde está guardada, o qué tipo de conexión estableció con el atacante para buscar en los logs de red la contraseña.

2.9. Exploit Kits

También llamados exploit pack son un conjunto de herramientas que automatizan la explotación de vulnerabilidades de software del lado cliente.

Su objetivo normalmente son navegadores o programas que la web puede invocar a través del navegador. Los típicos objetivos de esta clase de exploits han sido Adobe Reader, Java Runtime Environment y Adobe Flash Player. [18]

Es decir, en general es un programa que actúa como servidor web capaz de estudiar las características del software del cliente que ha establecido una conexión a él con el objetivo de averiguar si es vulnerable a alguno de los exploits que el exploit kit contiene. En caso de ser así, le envía el exploit cargado con un payload cuyo objetivo es infectar la máquina con un malware para tomar el control de esta o ejecutar algún otro tipo de malware como un ransomware.

Estos packs de malware y exploits se caracterizan por poder usarse de forma muy sencilla e incluso por personas que no son expertos de seguridad. Esto es debido a que un exploit kit normalmente ofrece una interfaz web muy fácil de usar para configurar todo lo que sea necesario y que ayuda a su dueño a controlar que tal va la campaña.

Muchas veces los grupos que crean estos exploit kits y los mantienen actualizados no los utilizan para expandir malware sino que se lo vende a la gente que esté interesada en ellos. Esto no solo provoca que sea mucho más complicado localizar a este grupo de personas, sino que fomenta que estos exploit packs sean hechos de forma que cada vez sean más fácil de usarse para aumentar el mercado en el que venderlo y que más clientes estén interesados.

Además, estas plataformas pueden estar hechas por gente de distintos países, ser vendidas a clientes de otros países distintos que la utilizarán para atacar a otros países y levantarán la plataforma en algún otro país. Lo cual complica mucho el averiguar quiénes son los miembros del grupo.

Algunos de estos exploit kits cuentan con algunos exploits día cero para atacar algún tipo de software de cliente. Sin embargo, en general lo que más utilizan son exploits para los cuales ya existen parches. Por ello la mejor defensa que se puede tener contra este tipo de ataques es usar programas actualizados y lógicamente no entran en páginas sospechosas.

2.10. Botnets

El término botnet es un nombre genérico usado para designar a un conjunto de máquinas comprometidas. Normalmente estas botnets están creadas por una persona o un grupo reducido de personas que utilizan malware y herramientas de las que se ha hablado anteriormente para poder entrar y ejecutar algún tipo de código en las máquinas que forman parte de la botnet. Estas máquinas que han sido infectadas son llamadas “bots” o “zombies”.[19]

No hay un número mínimo de máquinas infectadas para que el conjunto se denomine botnet, sin embargo, sí que es considerado que una botnet de cientos o pocos miles de máquinas es una pequeña mientras que una formada por millones de máquinas es una botnet grande.

El uso más tradicional de las botnets es el de poder coordinar a toda la botnet para lanzar ataques de denegación de servicio (DDoS) a servidores web. Un ejemplo representativo de un ataque de este tipo llevado a acabo por una botnet lo podemos encontrar en la Imagen 2. Sin embargo, el hecho de controlar tal cantidad de máquinas también permite realizar otras acciones complejas para las que se requiera mucha potencia de computación. Por ejemplo, cada vez más estas botnets se usan para minar monedas criptográficas o crackear contraseñas.

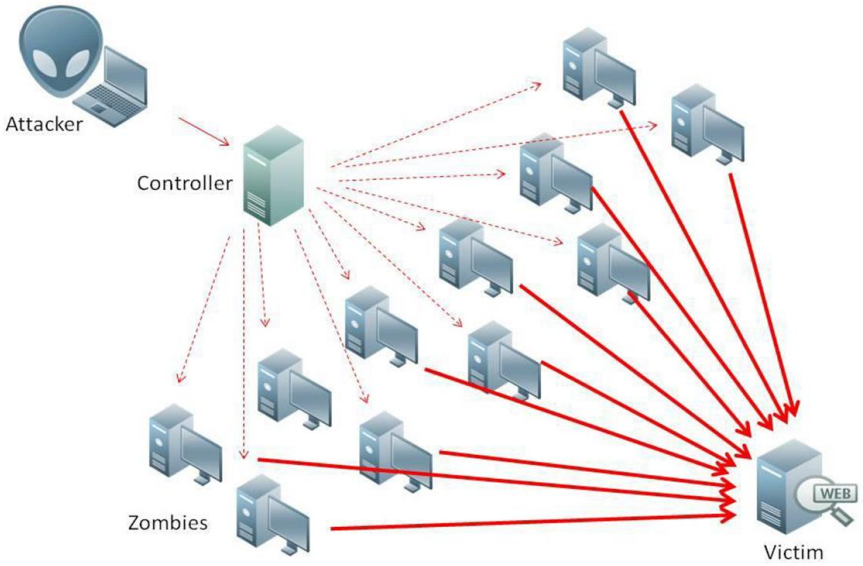


Imagen 2 - Ejemplo de botnet

En estos casos aumenta la complejidad de la sincronización de la botnet, pero creando un buen software para esto hoy en día hay botnets de millones de máquinas minando monedas y ganando dinero para sus dueños sin que los usuarios de los zombies lo sepan.

Además, este tipo de malware tiene algunas características similares a los exploit kits y es que muchos de los creadores del malware capaz de crear botnets venden este software a quien le interese utilizarlo. Además, crean interfaces que facilitan mucho el uso y configuración de la botnet así como la visualización de cómo se va expandiendo, esto hace posible que los usuarios que compran este tipo de malware no necesiten demasiado conocimiento del ámbito de la seguridad para utilizarlo.

Por este motivo en muchas ocasiones existen varias botnets del mismo tipo controladas por distintas personas.

A parte del modelo de negocio de usar la botnet para conseguir dinero o vender el software capaz de crear la botnet, hay gente que alquila botnets enormes por horas o días a otras personas que la necesite puntualmente.

2.11. RootKits

Sirven para modificar una parte sistema operativo de la víctima para ejecutar código manteniendo los privilegios obtenidos y hacer que sea más difícil al usuario detectarlo.

Son caballos de troya ya que se hacen pasar por código benigno.

Los rootkits no sirven para obtener acceso como root, sino para mantener el acceso de forma que se pueda entrar cuando quiera y a penas deje huella en el equipo. El atacante antes debe conseguir acceso root de alguna otra forma.

2.11.1. User-Mode RootKits

Son aquellos rootkits que todas las acciones las llevan a cabo desde el espacio de usuario y no tocan nada del kernel. Son más fáciles de detectar que los Kernel-Mode rootkits, pero también más fáciles de programar.

2.11.1.1. Unix

Los más conocidos usados para atacar sistemas operativos basados en Unix son:

LRK (Linux RootKit) y URK (Universal RootKit).

Típicamente los rootkits sustituyen los binarios de los comandos más usados por los administradores del ordenador. Por ejemplo se pueden sustituir el binario **sudo** para que al ser ejecutado arranque un backdoor. También se suelen sustituir los binarios de **top** y **ps** de forma que no muestren unos procesos determinados (los asociados al malware), se sustituye también **ls** para que no muestre los archivos maliciosos, **netstat** para que no muestre información de los puertos usados por el malware, etc.

2.11.1.2. Windows

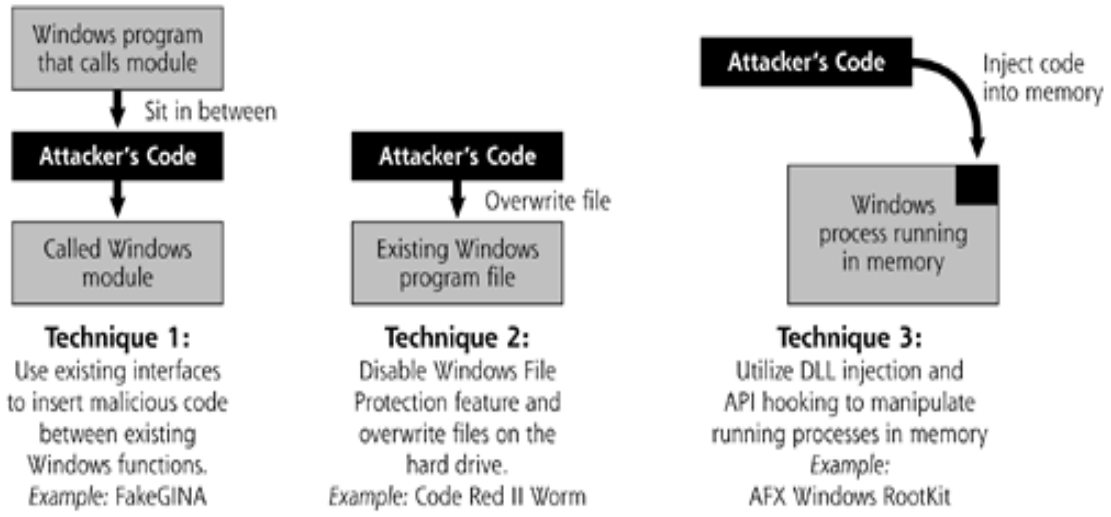


Figura 4 - Técnicas de RootKits en modo usuario para Windows

Como se puede ver en la figura 4 anterior, en el caso de Windows existen tres tipos de técnicas para aplicar un User-Mode RootKit. El primero consistiría en lograr intercalar

entre el orden de ejecución de una secuencia de programas, un código malicioso. Un buen ejemplo de esto es el caso del fakeGINA. Dicho programa se intercalaba en la ejecución normal de la autenticación inicial de Windows para robar las credenciales de usuario.

Otra forma de aplicar un User-Mode RootKit en windows es mediante la sobreescritura o sustitución de ejecutables muy comunes (como lo que se hizo en linux con los comandos). Sin embargo, para esto, hay que saltarse el WFP que es el encargado de revisar que estos archivos no sufran modificaciones, y en caso de sufrirlas las reemplaza con un backup. Pero es posible evitarlo.

También se puede realizar inyección de Dll de forma que se inyecta código en Dlls ya cargadas y usadas por programas, esto se conoce como API Hooking. De esta forma se añade código al proceso original para modificar su comportamiento y que se pueda por ejemplo esconder puertos en uso, registros, procesos… pues se inyecta código en las Dll que se encargan obtener esta información.

2.11.2. Kernel-Mode RootKits

Estos RootKits modifican el kernel de forma que todos los programas que dependen de él (prácticamente todos) pueden ser engañados para ocultar todo lo relacionado con el malware introducido.

Las tareas más comunes de este tipo de rootkits son:

- Esconder archivos y directorios.
- Esconder procesos.
- Esconder uso de modo promiscuo y de puertos.
- Redirección de ejecuciones, es decir, cuando el usuario va a ejecutar algo se ejecuta otra cosa a decidir por el atacante.
- Interceptación y control de dispositivos, como por ejemplo de teclados para crear un key-logger

De esta forma no es necesario sustituir todas las herramientas que pueden acceder a datos relacionados con el malware como se hace en los rootkits de de modo usuario, sino que todas son engañadas por defecto ya que todas acceden al kernel malicioso para obtener dicha información.

2.11.2.1. Linux

Este tipo de malware consiste en ser capaz de introducirse de alguna manera en el kernel para modificar ciertas acciones. Para ello se deben de haber obtenido privilegios de root.

Además, se puede encontrar una imagen del kernel que se está ejecutando en /dev/kmem.

Además, se puede lograr la comunicación entre un proceso corriendo en modo usuario y el módulo corriendo en el kernel mediante las system calls (/boot/System.map-4.4.0-45-generic , /boot/System.map-4.4.0-42-generic) lo que podría llegar a permitir al proceso de modo usuario ejecutar código en el kernel y con privilegios.

Como infectar el kernel con un rootkit:

Cargar un módulo malicioso en el kernel: Haciendo esto se podría alterar la tabla de los system call de forma que cada vez que se ejecute una llamada a una función, en realidad se llame a otra función maliciosa creada por el atacante de forma que probablemente oculte información al usuario pero este no lo note. De esta forma, si por

ejemplo se pide al kernel que ejecute un programa, el código malicioso puede decidir ejecutar otro.

En lugar de modificar la tabla, lo cual es fácilmente detectable, se pueden modificar las funciones de dicha tabla. Por ejemplo se puede modificar el código de SYS_execve con código malicioso de forma que cuando se haga una llamada a esta función, se ejecute también el nuevo código(es lo que se conoce como API Hooking).

Para cargar un nuevo módulo en el kernel se usa de **insmod**.

El uso de esta forma de ejecutar código en el kernel tiene dos problemas principales. El primero es que el administrador de la máquina puede mirar qué módulos están cargados en el kernel y detectarlo. Para lo cual el creador del rootkit necesitaría modificar el código de las funciones que den esta información.

El segundo y más importante es que esta forma de incluir módulos no tiene persistencia por defecto, por lo tanto hay que lograr de alguna forma que se cargue en cada arranque. Esto se suele lograr modificando el demonio init para que lo cargue.

Los rootkits de kernel más conocidos de este tipo son Adore y KIS (Kernel Intrusion System).

A pesar de todo esto también existe la posibilidad de que el administrador configure el kernel de forma que no soporte la inclusión de módulos. Entonces cada vez que quiera añadir algo tiene que reempaquetar el kernel de nuevo. Esta seguridad añadida se supera atacando /dev/kmem, el cual contiene una copia de la memoria del kernel. Con las herramientas adecuadas se puede navegar por esta y escribir de forma que se puede encontrar la system call table y modificar las funciones a las que llama o inyectarles código, siguiendo el mismo razonamiento que en el párrafo anterior.

Un rootkit que hace esto es SucKIT.

Otra forma de infectar la máquina con un rootkit sin insertar un módulo en el kernel es **crear un kernel malicioso** y sustituir al actual. De esta forma el rootkit se cargará constantemente sin hacer nada.

El kernel suele estar comprimido en /boot/vmlinuz por lo que se podría meter el código de los rootkits anteriores en el kernel, volver a comprimirlo y sustituir el kernel actual, de forma que siempre se cargaría el kernel con los rootkits y el usuario no notaría nada.

Un último método para ejecutar rootkits de este tipo es hacer **correr encima del kernel real otro kernel** dominado por el atacante. Esto es como meter al atacante dentro de una máquina virtual sin que este lo sepa habiendo sido el kernel de la virtual máquina creado por el atacante.

2.11.2.2. Windows

Para realizar peticiones al kernel de windows un programa normal sigue el siguiente camino: hace peticiones a la API Win32 DLL la cual llama a Ntdll.dll y este que invoca a the Executive el cual finalmente llama al kernel que llama a HAL (Hardware Abstraction Layer).

The system service dispatcher es lo equivalente a las system calls de linux.

Estas llamadas a funciones del kernel buscan en la system service dispatch table dónde está el código para manejar cada interrupción.

A parte del system service dispatcher Windows guarda otras similitudes con Linux. Por ejemplo el proceso smss.exe es el análogo al demonio init, es decir, es el primer proceso que comienza en User-Mode que llama al resto de procesos a ejecutar.

En el caso de lo módulos para el kernel que se pueden insertar en linux, en windows un administrador puede cambiar la funcionalidad del kernel añadiendo device drivers.

Las formas de ataque con RootKits en modo kernel en windows son muy similares a las vistas en linux:

Se puede insertar un device drivers que modifique el resultado de las llamadas al kernel como las que se hacen para listar procesos, mostrar archivos, identificar uso de UDP y TCP … Esto se puede hacer sobreescribiendo el código del kernel al que llaman las interrupciones, modificando la tabla de forma que la dirección de las interrupciones apunten al nuevo código malicioso, o se puede modificar la gestión de las interrupciones de forma que las algunas interrupciones elegidas por el nuevo driver insertado, las maneja él.

Otra opción es tocar el kernel directamente en la memoria donde está guardado, tal y como se podía hacer en linux.

También se puede parchear el kernel de windows para que el rootkit gane persistencia y se ejecute cada vez que se arranca el sistema operativo sin que el usuario lo note. Para ello se puede modificar Ntoskrnl.exe y también habría que modificar el NTLDR para que no hiciese la comprobación del archivo anterior y detectase el cambio.

Una última opción opción es abrir una máquina virtual de forma que la víctima esté haciendo todo en una máquina virtual sin que él lo sepa. Sin embargo, en windows abrir una máquina virtual es mucho más pesado que en linux y muestran más información al usuario de forma que la víctima lo notaría muy fácilmente.

2.12. Malware en BIOS y microcode

2.12.1. BIOS

Cuando el sistema se enciende la BIOS ejecuta el BIOS boot program enviando las instrucciones a la CPU. Posteriormente dicho programa localiza el master boot record del disco duro, lee su contenido y lo ejecuta.

El master boot record contiene un pequeño programa que localiza la información de arranque que puede tener cada partición del disco. Posteriormente elige la partición del disco que la contiene y esta información es usada para cargar el sistema operativo en memoria. De esta forma el BIOS boot program es un programa crítico que se encarga de todo el proceso de localizar el sistema operativo y cargarlo.

Hay un proyecto que permite guardar el kernel de linux en la BIOS y modificar la BIOS de forma que inicie Linux directamente permitiendo un arranque mucho más rápido que si se siguiera el proceso comentado anteriormente. Además el proyecto es de código libre por lo que es posible modificarlo.

Un atacante de una víctima linux podría coger el código del proyecto anterior y modificar el kernel de linux para incluir un RootKit. A continuación se modificaría la BIOS de la máquina introduciendo la nueva BIOS que arranca directamente el nuevo kernel malicioso.

De esta forma, cuando se arrancase el ordenador, este arrancaría el kernel malicioso con el rootKit y el usuario no lo notaría. El otro kernel seguiría estando guardado en la máquina, pero nunca se usaría.

En el caso de infectar la BIOS esta es una de las formas más sencillas de hacerlo pero no la única. El hecho de que la BIOS tenga la capacidad de ejecutar código abre las puertas a ataques mucho más complejos y sofisticados que hacen casi imposible la detección del ataque y el arreglo del dispositivo.

2.12.2. Microcode

El microcode es el código que viene implementado en la CPU y que le permite realizar operaciones más complicadas (como COMISS dst, src). Para implementar nuevo código este debe ser guardado en la BIOS de forma que lo añada en el CPU en cada reboot.

El hecho de poder modificar y añadir instrucciones de la CPU que puedan ser malignas hace que el tipo de malware que actúa a este nivel sea muy difícil de detectar y casi imposible de eliminar. No solo habría que modificar la BIOS sino que en algunos casos también habría que cambiar la CPU.

Lo más sencillo que se podría hacer con este tipo de malware sería provocar un DoS. Pero también se podrían realizar acciones más complejas como hacer que el CPU lance un backdoor o un rootKit escondido en algún sitio del disco de forma que a penas haya forma de saber que este está corriendo.

2.13. Técnicas antidetección de malware

2.13.1. Sigilo

Muchos virus añaden a sus archivos el atributo hidden o se esconden aprovechando el ADS.

Otra forma muy usada por los virus para intentar no ser reconocidos por los antivirus es mentirles cuando le analizan dándole una copia del ejecutable sin infectar.

2.13.2. Polimorfismo y metamorfismo

- **Polimorfismo**: Consiste en modificar de alguna forma el código del virus en cada infección. Para ello se pueden realizar varias acciones: puede cambiar el nombre de variables internas y subrutinas, puede cambiar el orden de las instrucciones o incluir órdenes que no sirven para nada. Puede ser que cifre su código y tan solo deje limpias las líneas que necesite para descifrarlo y cifre con distintas claves en cada infección (guardando la clave lógicamente).

El objetivo de este tipo de virus es intentar evitar que el antivirus pueda detectarlo mediante firmas.

- **Metamorphism**: Cambia un poco la funcionalidad del virus en cada infección, eso no significa que en cada infección realice una tarea distinta, sino que la realiza de forma distinta. Para ello puede cambiar la estructura de los archivos, modificar rutinas de mutación y cifrado, o pueden desensamblarse, modificarse y volver a ensamblarse.

2.13.3. Desactivación del antivirus

Muchos virus saben los procesos de los antivirus más usados y los desactivan. También suelen impedir la actualización de las bases de datos de virus, por ejemplo impidiendo llegar a los dominios de los antivirus.

2.14 Defensa básica contra el malware

2.14.1. Uso de antivirus

La acción más recomendable hoy en día es tener instalado al menos un antivirus en la máquina. Este sirve para analizar cada archivo nuevo que llega a la máquina por si fuese malicioso o, en caso de serlo y no detectarlo como tal, detectarlo unos pocos días cuando la firma del malware haya sido añadida a la base de datos del antivirus[20].

Los antivirus usan principalmente dos formas de detectar malware:

2.14.1.1. Firmas de virus

Cogen una parte de virus como su firma y lo guardan en una base de datos, así si pilla esa parte el antivirus, lo detectara como virus.

Esta parte del virus seleccionada puede ser desde un hash del archivo malicioso hasta unos strings que contenga el malware. El principal objetivo de la firma es que sea capaz de identificar al malware y otros de su familia pero que no provoque falsos positivos.

2.14.1.2. Heuristicos

Consiste en ver cómo se comporta el programa y en función de unos parámetros determinar si es un malware o no. Sin embargo, para esta técnica existen varias limitaciones ya que muchos de los malware están hechos para correr bajo determinadas circunstancias que no pueden ser fácilmente creadas en un entorno genérico de estudio de malware sin intervención manual.

2.14.2. Verificación de integridad

Algunos antivirus y programas para aumentar la seguridad del sistema monitorizan todos o algunos archivos. Esto lo hacen guardando algunos hashes de los archivos a monitorizar y en caso de que los archivos sufran cambios (cambien los hashes que genera el archivo) se estudia más a fondo qué ha cambiado en el archivo

2.14.3. Configuración segura

Consiste en configurar concienzudamente el equipo de forma que principalmente no se ejecuten más servicios de los necesarios y que dichos servicios y los usuarios del sistema se ejecuten con los mínimos privilegios necesarios.

2.14.4. Educación

Sin duda uno de los métodos más efectivos para evitar la entrada de malware en las máquinas es la educación de los usuarios de estos. Ya que, a pesar de haber formas de entrar en las máquinas sin que el usuario realice ninguna acción, la mayoría de las veces que un dispositivo es vulnerado es gracias a la acción del usuario. Esta puede ser abrir un archivo que se le envía mediante un correo o acceder a una página web muy sospechosa de contener un exploit kit.

2.14.5. Control de conexiones

Una de las principales formas tanto de evitar como de detectar malware es mediante un estricto control de las conexiones que realizan los equipos.

Es importante conocer qué tipo de conexiones tiene que poder realizar una máquinas y cuales no debería realiza con el objetivo de filtrar tanto conexiones salientes como entrantes según cada máquina. Por ejemplo, un servidor web nunca iniciará una conexión, por lo que bloquear todo intento de este de establecer conexiones y en caso de que ocurra notificarlo es algo importante.

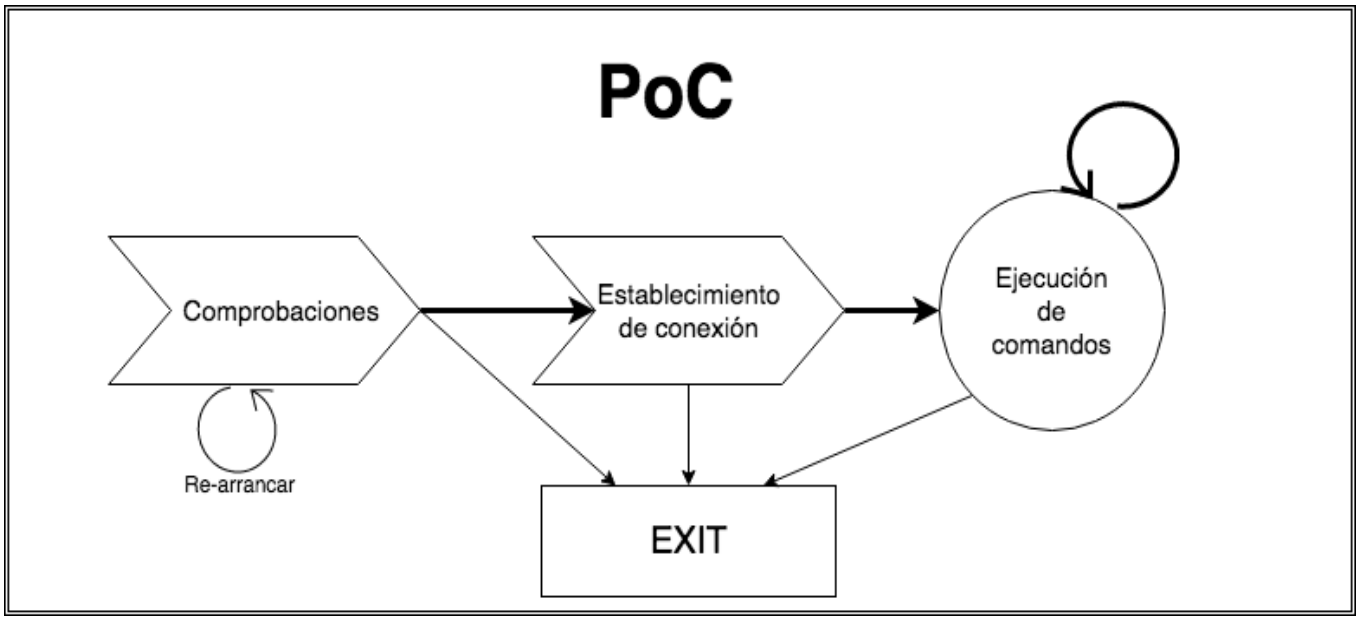
Otro control importante que se puede realizar en las máquinas es comprobar que no envían paquetes IP con direcciones IP que no son las suyas.

3. Diseño de una Prueba de Concepto de Malware

Para la prueba de concepto se ha decidido crear un único malware que reúne varias de las capacidades anteriormente comentadas. A pesar de ello, la categoría en la que mejor encajaría este malware sería en la de RAT (Remote Access Trojan) ya que como se podrá ver la funcionalidad principal del malware es obtener el control total de la máquina de forma remota. Se puede encontrar la prueba de concepto en **https://github.com/carlospolop/MalwarePoC**, para que no se pueda usar con fines maliciosos se le han añadido algunas características que hacen que esta prueba sea detectable por varios antivirus.

Durante el diseño de la PoC se decidió que se crearían tres bloques independientes que al combinarse formaran el malware. Estos tres bloques son: **Comprobaciones que realiza el malware al ejecutarse, Comunicación entre el malware y su dueño** y **Comandos propios del malware**. Se decidió separar estos tres bloques para que tanto el desarrollo como el futuro escalado del malware fuese más sencillo. De esta forma se le pueden añadir más funcionalidades sin aumentar la complejidad de añadirlas.

En la figura número 5 se muestra un resumen del camino que lleva la ejecución del



malware:

Figura 5 - Ejecución de la Prueba de Concepto

3.1. Comprobaciones que realiza el malware al ejecutarse

Esta parte del código es la primera que correrá la PoC al ejecutarla. Su función es la comprobar distintos parámetros para configurar la forma en la que debe ejecutarse el malware. Realiza comprobaciones sobre si se está ejecutando con privilegios de administrador, sobre si el malware está siendo ya ejecutado y sobre qué parámetros se le han pasado para configurar la IP y puerto al que conectarse y el método de ejecución (normal o de suplantación).

Una vez comprobados estos datos procede a configurar la ejecución y decide si proceder con una ejecución normal, una de suplantación o eliminar el proceso pues ya hay otro malware corriendo.

En caso de una ejecución normal, realizará varios intentos de conexión antes de eliminar el proceso.

3.2. Comunicación entre el malware y su dueño

Una vez que el malware ha comprobado y configurado todos los parámetros y haya optado por una ejecución normal del mismo lo primero que hace es establecer una conexión a la IP y puerto deseado.

A parte del establecimiento de la conexión, este bloque cuenta con otras dos funcionalidades importantes. La primera es la posibilidad de usar cifrado en la conexión, de forma que todo lo que entra por ese enlace se descifra y todo lo que se envía se cifra antes de ser enviado. La otra funcionalidad importante es la capacidad de decisión de si los datos recibidos son enviados al bloque de comandos de la PoC o si son enviados a una consola de comandos de windows a ser ejecutados.

Una vez que el malware ha comprobado y configurado todos los parámetros y haya optado por una ejecución normal del mismo lo primero que hace es establecer una conexión a la IP y puerto deseado.

A parte del establecimiento de la conexión, este bloque cuenta con otras dos funcionalidades importantes. La primera es la posibilidad de usar cifrado en la conexión, de forma que todo lo que entra por ese enlace se descifra y todo lo que se envía se cifra antes de ser enviado. La otra funcionalidad importante es la capacidad de decisión de si los datos recibidos son enviados al bloque de comandos de la PoC o si son enviados a una consola de comandos de windows a ser ejecutados.

3.3. Comandos propios del malware

El último bloque sin duda es el más extenso en cuanto a código. En este se recogen todas las funcionalidades programadas en el malware para que se pueden ejecutar en caso de ser solicitado por el controlador. Para llegar a ejecutar cualquiera de estas acciones la PoC tiene que haberse ejecutado siguiendo una ejecución normal al realizar las primeras comprobaciones, tiene que haber establecido una conexión con el dueño del malware y este debe enviarle el código de la funcionalidad que quiere ejecutar para que el bloque de comunicación pase este código al bloque de comandos y este lleve a cabo la acción solicitada.

El bloque de comandos es a su vez divisible en tantos bloques como categoría de comandos se han implementado. Actualmente la PoC cuenta con 8 categoría de comandos, y, como se ha explicado anteriormente, está hecho para que se puedan añadir todos los que se quiera sin aumentar la complejidad de añadir más.

3.3.1. Categoría 10X: Spyware

Esta categoría engloba todas las funciones que realizan tareas de este tipo de malware. En concreto se ha programado la tarea de iniciar un keylogger, comprobar si está activo y pararlo. También se podrían añadir otras funcionalidades como captura de sonido e imagen usando el micrófono y la cámara, o realizar una captura de pantalla.

3.3.2. Categoría 11X: Creación de troyanos mediante infección

Esta categoría es la que puede dar a la PoC la categoría de virus ya que hace posible la inyección de código en otros ejecutables, creando troyanos de este modo. Más concretamente permite infectar ejecutables de windows con una shellcode que se proporcione al malware.

En esta categoría se encuentra la opción de infectar un ejecutable, o todos los ejecutables de una carpeta.

3.3.3. Categoría 12X: Backdoor, ganar persistencia

Esta categoría engloba todas las funcionalidades de la PoC que le permite ganar persistencia. De esta forma actúa como Backdoor pues logrará iniciarse y ejecutar el código sin que el usuario lo sepa. En esta categoría están implementadas las funciones de ganar persistencia mediante la escritura en los dos registros de RUN de HKCU, la escritura en cualquier clave de HKLM, creación de un servicio que ejecute el malware y suplantación de programas de la carpeta \System32.

3.3.4. Categoría 13X: Descargar y subir

Esta categoría recoge las funciones que permiten a la PoC descargar archivos a memoria o a disco desde el ordenador del dueño del malware o desde internet. Tanto la funcionalidad vista de creación de troyanos mediante infección como la creación de procesos mediante runPE (siguiente categoría) necesitan de estas funcionalidades para descargar la shellcode o el programa en memoria.

3.3.5. Categoría 14X: Creación de procesos

Esta categoría permite crear nuevos procesos de dos formas. La primera, de la forma más sencilla, indicando el path del programa que queremos ejecutar. Y la segunda utiliza la técnica de runPE. Esta permite iniciar un proceso desde un ejecutable del cual la información en lugar de estar en el disco está en memoria. Esto permite ejecutar programas enviandoselos al malware sin que toquen el disco, lo cual es una buena forma de evitar los antivirus.

3.3.6. Categoría 15X: Escalada de privilegios

En esta categoría se engloban los métodos para escalar privilegios y, partiendo de una ejecución del malware sin privilegios, lograr ejecutarlo con ellos. En esta categoría encontramos una única funcionalidad de escalada de privilegios basada en la vulnerabilidad SilentCleanup de Windows encontrada a penas hace una semanas y que a día de este escrito aún no está corregida. Por este motivo se podría catalogar esta función como un exploit día cero.

3.3.7. Categoría 20X: Ransomware

Se puede observar que el número de las categorías a dado un salto del 15 al 20. Esto es porque el objetivo de estas categorías ya no es obtener un mejor control del sistema afectado, sino molestar al usuario de este sistema.

En concreto, en esta categoría encontramos la capacidad de cifrar un archivo o una carpeta pudiendo elegir entre tres tipos de cifrado distinto. Dos de los cuales son

fácilmente reversibles (“byte inversion” y “byte cycle”) y uno más complicado de descifrar si no se posee la clave (rc4).

Como se puede deducir esta no es la funcionalidad de un ransomware al completo pues no pide ningún tipo de rescate, pero sí ofrece la característica principal del ransonware, el cifrado de archivos.

3.3.8. Categoría 21X: Bucle Infinito

Los bucles infinitos son partes de código que entran en bucle y no acaban nunca, y además van exigiendo cada vez más recursos del ordenador hasta que lo paralizan.

En esta categoría encontramos actualmente tan solo una función que comienza a calcular decimales de pi y no acaba hasta haber encontrado 999999999 decimales. Además, esta misma función cada cierto tiempo comienza otras threads para que hagan lo mismo, por lo que al final el ordenador no soporta tanta carga y se cae.

Cuando se ejecutan este tipo de funciones generalmente es necesario un reinicio el ordenador para volver a conseguir que este funcione.

3.3.9. Categoría 22X: Molestar

En esta categoría están recogidas funciones que la única relación que guardan entre sí es la capacidad de molestar al usuario. Encontramos una función para cambiar el fondo de pantalla, otra para esconder la barra de más abajo de la pantalla de windows, otra para esconder archivos o carpetas enteras, otra para infectar ejecutables con parte de una shellcode que reconocen los antivirus y otra cambiar la hora a X años al futuro o al pasado.

3.3.10. Categoría 999: Exit

El código 999 permite cerrar el proceso del malware.

4. Desarrollo de la Prueba de Concepto de Malware

A continuación se detallará un poco más la forma en la que se ha implementado el malware creado como PoC. Se redactarán en cursiva las referencias a variables y funciones declaradas en el código.

4.1. Comprobaciones que realiza el malware al ejecutarse

Esta parte del código está formado por los archivos Source.cpp y malware.cpp y sus respectivos headers.

Como se dijo anteriormente este bloque del código se encarga de comprobar las condiciones de ejecución de la PoC y los parámetros que se pasa al malware al ejecutarlo así como de configurar su ejecución.

Las condiciones de ejecución que el malware comprueba son:

- Si el usuario tiene privilegios de administrador mediante la función *IsUserAdmin()* pues en caso de que así sea se conectará al puerto *DEFAULT_ADMIN_PORT* y no comprobará si hay otra instancia del malware corriendo.
- Si hay otra instancia del malware mediante un *mutexe.* Tan solo una thread puede tener un mutexe del mismo nombre por lo que al iniciar el malware se comprueba si algun thhread tiene un mutexe con el nombre *FAKE_NAME* y sino se crea. En caso de que ya la hubiese, el proceso se termina mediante la función *exit()*.
- Si el proceso ha sido iniciado gracias al método de suplantación de un ejecutable en la carpeta System32 (este método se explica más adelante). En ese caso el malware ejecutará en un proceso a parte el programa al que suplanta, en otro proceso volverá a llamarse a sí mismo pero pasándole una clave como parámetro para que se salte esta comprobación (ejecución de suplantación) y cerrará su proceso.

Una vez comprobadas estas condiciones y si el proceso no se ha cerrado por los motivos vistos anteriormente, se configurarán las variables en función de los parámetros pasados. La prueba de concepto soporta:

- Que se le pase un parámetro y sea el puerto al cual conectarse.
- Que se le pase un parámetro y sea la clave (*PATH_KEY*) para saltarse la comprobación de suplantación.
- Que se le pasen dos parámetros y sean el puerto y la Ip a la que conectarse.

Por si no se le pasa puerto y/o Ip a la PoC viene configurada con unos por defecto: *DEFAULT_PORT* y *DEFAULT_IPi*.

Configuradas estas variables se crea un objeto de la clase MALWARE y se procede a crear la conexión con el puerto e Ip configurados. En caso de no conseguirse se espera un tiempo (*WAIT_CONECTION*) y se vuelve a intentar hasta *NUM_RETRY* veces.

4.2. Comunicación entre el malware y su dueño

Esta parte del código está formado por los archivos Socket_functions.cpp y ciphers.cpp y sus respectivos headers.

La parte principal de este código es el establecimiento de la conexión con el atacante. Sin embargo, no es objetivo de este documento explicar paso a paso cómo se establece un enlace en C++, pero sí se explicará qué uso hace la PoC de este enlace.

Durante el establecimiento de la conexión también se configuran tres Pipes. Uno para poder enviar y recibir mensajes por el enlace, otro para poder transmitir los mensajes del enlace a un proceso de cmd.exe y enviar las respuestas de este proceso por el enlace, y otro para poder enviar los mensajes del enlace al bloque de comandos y enviar la respuesta de este por el enlace.

Por defecto los mensajes que le llegan al malware por la conexión establecida son transmitidos al bloque de comandos a no ser que el mensaje comience por *CMD*, en cuyo caso será transmitido el resto del mensaje al proceso de la consola de comandos para que lo ejecute.

Para enviar el mensaje recibido al bloque de comandos se crea un objeto de la clase COMMANDS, se guarda la longitud del comando recibido mediante la función *set_lenght_cmd()* y se pasa dicho comando mediante la función *exec()*.

Otra funcionalidad importante de la conexión es la posibilidad del uso de cifrado. En concreto se ha usado como cifrado un XOR que parte de un byte como clave (*CIPHER_CHAR_KEY*) y que usa el resultado del XOR anterior como clave para cifrar el byte siguiente. Es una algoritmo sencillo de implementar, rápido a la hora complir con su cometido y genera una gran entropía lo que hace complicado averiguar qué algoritmo es a no ser que se cuente con el código.

La conexión entre la PoC y el atacante permanecerá levantada hasta que el malware pierda conexión con Internet, el atacante la cierre o se cierre el proceso de la PoC.

4.3. Comandos propios del malware

La parte principal de este código está formada por el archivo commands.cpp y su header. En cada categoría se especifica si se hace uso de más archivos.

La función principal de los objetos de la clase COMMANDS es *exec()*. Esta recibe el mensaje de la conexión establecida, lo trata, y mediante un switch ejecuta la funcionalidad solicitada por el código recibido en el mensaje.

La PoC está hecha de forma que para correr el código de una funcionalidad de la clase COMMAND se tiene que enviar su código asociado y también se puede enviar un parámetro separado por un espacio del código.

4.3.1. Categoría 10X: Spyware

La parte principal de este código está en el archivo Keylogger.cpp y su header.

- 101 Keylogger_start(path)

Este código inicia un keylogger al que se le puede pasar el path a un archivo donde se quiera recoger la información de las teclas pulsadas, o se usará “\\k.temp” por defecto.

Para crear el keylogger se ha hecho un hook mediante la función *SetWindowsHookEx()* a *WH_KEYBOARD_LL*. Esto significa que cada vez que se pulse una tecla se llamará a la función que se le programe, en este caso a *LowLevelKeyboardProc*.

A esta función básicamente se le pasa la tecla pulsada y si ha sido pulsada o dejada de pulsar. Por lo tanto con un switch busca qué tecla ha sido pulsada y la apunta en el documento seleccionado.

Además, también mira en qué proceso ha sido escrita dicha letra (*GetWindowText*) y si ha sido en un proceso distinto al que fue escrita la anterior letra también apuntará el nombre del proceso.

Esto sirve para identificar mejor en qué contexto el usuario a escrito las teclas captadas.

En caso de haberse creado la thread correctamente este comando actualiza el valor de *_running_keylogger* a *TRUE* y devuelve *CORRECT*.

- 102 Keylogger_alive()

Mediante un dato de tipo booleano del objeto de COMMANDS creado se mantiene un registro de si se ha levantado ya un keylogger o no. De esta forma, si ya ha un keylogger corriendo se devuelve *RUNNING,* y en caso contrario, *NOT_RUNNING;*

- 103 Keylogger_stop()

Mediante las funciones de *OpenThread()* y *TerminateThread()* elimina la thread que estaba ejecutando el keylogger y actualiza el valor de *_running_keylogger.* En caso de ejecutarse correctamente devuelve *CORRECT.* En caso contrario, devuelve *INCORRECT.*

4.3.2. Categoría 11X: Creación de troyanos mediante infección

- 111 Infect_file(path, FALSE)

La parte principal de esta funcionalidad se encuentra en infectPE.cpp y su header.

El FALSE proporcionado a esta función sirve para indicar que no queremos que se use la shellcode que captan los antivirus guardada en la PoC, sino que queremos que use la shellcode que ha debido de ser descargada anteriormente en memoria. En caso de no haberse descargado ninguna se notifica mediante la devolución de *NEED_SHELLCODE.*

En caso de no recibir ningún path sobre qué archivo infectar también se notifica mediante *PARAMETER.*

Esta función comprueba que el path pasado contenga la cadena ".exe" en cuyo caso lo infectará mediante el uso de la función *infect_main()*, o la cadena ".lnk", en cuyo caso se lo pasará a *infect_link()*. En caso de no encontrar ninguna de esta cadenas devolverá *INCORRECT.*

Los programas que contienen la cadena ".lnk" son los accesos directos, los cuales guardan en su interior el path absoluto del programa al que tienen que llamar. La función *infect_link()* recorre el contenido del acceso directo y busca el path absoluto del ejecutable al que debe llamar. Una vez encontrado, le pasa este path a *infect_file()*. En caso de que no encuentre ningún path devuelve *NO_PATH_DIRECT_ACCS.*

En caso de que *infect_file()* tenga el path a un archivo ".exe" y haya una shellcode descargada, llama a *infect_main()*, a la cual pasa el path del ejecutable, la shellcode con la que infectar y le indica que debe usar esa shellcode y no la guardada.

Es importante que la shellcode no tenga valores nulos (0x0), en caso contrario no se calculará bien el tamaño de esta.

Con estos datos se procede a la infección de dicho archivo con la shellcode mediante la función *infectPE_file()*. La cual interpretará el path como si fuese de un ejecutable y buscará un número de bytes nulos en la parte del ejecutable asociada al código igual o superior al tamaño de la shellcode. En caso de encontrarlo copiará ahí la shellcode, modificará en EntryPoint de la cabecera del ejecutable para que comience ejecutando la shellcode y añadirá un JMP al final de la shellcode con dirección al EntryPoint original para que después de ejecutar la shellcode el nuevo troyano ejecute también el programa principal.

En caso de que todo esto haya funcionado devolverá *suc_msg,* y en caso de que algo haya fallado devolverá *noCave_msg* o *noOpen_msg*.

- 112 Infect_folder(path, FALSE)

Recorre los archivos guardados en el directorio actual o del path dado e intenta infectarlos pasándolos a la función *infect_file()*.

Antes comprueba que se haya descargado una shellcole y si no es así devuelve *NEED_SHELLCODE.* En caso de ejecutarse correctamente, devuelve *CORRECT,* y si algo falla devuelve *INCORRECT.*

Hay que tener en cuenta que el hecho de que *infect_folder()* devuelva *CORRECT* no significa que se haya infectado algún ejecutable, solo que la función se ha ejecutado correctamente.

4.3.3. Categoría 12X: Backdoor, ganar persistencia

Todo el código de esta categoría se encuentra en commands.cpp.

Varias de las funciones de esta categoría usan la función *writeInRegistry* la cual recibe la clave principal (solo se usan HKCU y HKLM), la subclave en la que escribir y el nombre del nuevo valor a escribir o por defecto usa *FAKE_NAME*. Siempre se escribirá como valor el path del malware obtenido mediante *GetModuleFileName*. Además, para escribir en el registro se usará *RegCreateKeyEx* y *RegSetValueEx*.

En caso de que se escriba el valor deseado en el sitio del registro deseado esta función devolverá *CORRECT*. En caso contrario devolverá *INCORRECT.*

- 121 Persis_runRegistry(val_name)

Escribe en el registro HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run llamando a la función *writeInRegistry().*

Se escribe en esta parte del registro el path al malware pues cada vez que el usuario inicie sesión se ejecutarán todos los programas aquí escritos.

- 122 Persis_runE64Registry(val_name)

Escribe en el registro HKEY_CURRENT_USER\SOFTWARE\Wow6432Node

\Microsoft\Windows\CurrentVersion\Run llamando a la función *writeInRegistry().*

Se escribe en esta parte del registro el path al malware pues cada vez que el usuario inicie sesión se ejecutarán todos los programas aquí escritos.

- 122 Persis_anyRegistryLM(sub_key)

Permite llamar a *writeInRegistry()* pasándole en la subclave indicada. Es decir, escribirá un nuevo valor en dicha subclave. Hay varios registros pertenecientes a HKLM que permiten ganar persistencia. Sin embargo, para escribir en este registro se requieren privilegios de administrador.

- 123 Persis_service(serv_name)

Llama a la función *create_a_service()* la cual crea un servicio con el nombre pasado que iniciará el malware cada vez que se encienda el ordenador. Para crear este servicio se usan las funciones OpenSCManager y *CreateService*. En caso de que el servicio se cree correctamente se devolverá *CORRECT* y en caso de que haya algún problema como la falta de privilegios, se devolverá *INCORRECT.*

- 124 Persis_path_system32(cmd_name)

Esta forma de persistencia está basada en una debilidad que existe en Windows desde sus comienzos. Consiste en que si tienes dor ejecutables en un mismo directorio uno con la extensión ".exe" y otro con la extensión ".com" (hace años que este tipo de ejecutables ya no se crean debido a sus limitaciones) y para invocar el ejecutable no escribes la extensión sino tan solo el nombre, se ejecutará el programa con extensión ".com". Por lo tanto si por ejemplo creas un ejecutable llamado "notepad.com" en C:\Windows\System32 e invocas "notepad" desde la consola, en vez de ejecutarse el programa "notepad.exe" se ejecutará el nuevo "notepad.com".

Esta funcionalidad permite seleccionar a qué ejecutable se desea suplantar de System32 y creará una copia del malware en este directorio con el mismo nombre pero con extensión ".com" y, además, lo ocultara configurando el atributo oculto (*SetFileAttribute()*).

De esta forma, cuando el malware se invoque, a no ser que se le pase la clave que permite saltarse la comprobación de si está suplantando a algún ejecutable el malware buscará si su nombre tiene extensión ".com", en caso de ser así lanzará un nuevo proceso del malware pasándole la clave que evita esta comprobación y acto seguido lanzará un nuevo proceso con el ejecutable al que está suplantando. De esta forma el malware se lanzará siempre que el usuario quiera correr el programa residente en System32 desde la consola.

4.3.4. Categoría 13X: Descargar y subir

Todo el código de esta categoría se encuentra en commands.cpp.

Algunas funciones ya comentadas o otras que aún faltan por comentar requieren de descargar shellcodes o ejecutables en memoria para después poder usarlos.

- 131 Download_to_memory(file_length)

Esta funcionalidad permite preparar la descarga en memoria de los datos que el emisor quiera enviar. Para ello lo primero que hace es comprobar si se le ha pasado la longitud del archivo y en caso de no ser así devuelve *PARAMETER*. En caso de habersela pasado, intenta liberar la memoria que podría estar usando de una llamada

anterior a esta función ejecutando *free_virtual_alloc()*. Posteriormente configura los parámetros necesarios para indicar que se va a descargar un archivo y de qué tamaño va a ser y reserva memoria para él usando *VirtualAlloc()*. En caso de que todo vaya bien, devuelve CORRECT, en caso contrario, INCORRECT.

Habiendo configurado estos parámetros el malware está preparado para que el próximo número de bytes igual al tamaño del archivo indicado se copien en memoria.

Esto es posible gracias a que la función *exec()* de la clase COMMANDS siempre que recibe algún mensaje comprueba si se está algo descargando en memoria o no. Debido a que la función *download_to_memory()* ha configurado esto, todos los datos que le lleguen a la función *exec()* serán pasados al función *save_data()* para que los guarde en la memoria reservada.

La función *save_data()* configurará que no se están descargando más datos cuando el número de bytes guardado sea igual al indicado al principio de todo el proceso. De forma que los próximos mensajes que le lleguen a la función *exec()* sean tratados de forma normal y los datos ya estén guardados en memoria.

- 132 Download_to_memory(file_length) and then to disk

Para esta funcionalidad se sigue el mismo proceso que en el apartado anterior y se realizan algunas acciones extra.

Después de llamar a la función *download_to_memory()* y antes de recibir los datos se configura la variable *is_dowloading_disk* a *TRUE.* De esta forma cuando la función *save_data()* termine de guardar los datos en memoria llamará a la función *download_to_disk()*. La cual escribe todos los datos guardados en memoria en un archivo con el nombre de *DOWNLOAD_FILE* y configura el parámetro *is_downloading_disk* a FALSE.

En caso de que se escriba correctamente devuelve *CORRECT*, en caso contrario devuelve *INCORRECT*.

- 133 Download_to_disk()

Realiza la acción de guardar los datos almacenados en memoria a disco y modificar la variable *is_downloading_disk* como se ha comentado justo antes.

En caso de que se escriba correctamente devuelve *CORRECT*, en caso contrario devuelve *INCORRECT*.

- 134 Download_file(url)

Descarga el contenido de la URL pasada en un archivo llamado *DONWLOAD_FILE* mediante la función *URLDownloadToFile()*.

En caso de que se descargue correctamente devuelve *CORRECT*, en caso contrario devuelve *INCORRECT*.

- 135 Free_virtual_alloc()

Para esta funcionalidad se llama la función ya comentafa *free_virual_alloc()* y se pasa el puntero que se usa para apuntar a la memoria reservada a *NULL.*

Devuelve siempre *CORRECT.*

- 136 Upload(path)

Esta función recibe como parámetro el path a la función que se quiere transferir. En caso de no recibir ningún path devuelve *PARAMETER*.

En caso de recibirlo, abre el archivo indicado en el path guarda todo el contenido en una variable y prepara la transmisión del archivo indicando el tamaño del mismo en la variable *_file_size_up* y poniendo a *TRUE* la variable *_is_uploading*.

En caso de que todo haya ido bien devuelve el puntero a los datos del archivo almacenados. En caso de que hubiese problemas al abrir el archivo se devuelve *noOpen_msg*.

Al enviar mensajes que vienen del bloque de comandos siempre se revisa la variable *_is_uploading*. En caso de que esta sea *TRUE* en vez de tomar la longitud de los datos a enviar por el enlace mediante la función *strlen()* la toma de la variable *_file_size_up*. Posteriormente envía los datos y pone la variable *_is_uploading* a *FALSE.*

4.3.5. Categoría 14X: Creación de procesos

- 141 runPE_memory(fake_program)

El código principal de esta funcionalidad se encuentra en runPE.cpp y su header.

Es una forma de ejecutar programas de forma que estos no necesiten tocar el disco, se hace todo desde la memoria. Además, permite dar la sensación de que en realidad se está ejecutando un programa benigno ya que se crea el proceso de el ejecutable que se quiera y este proceso es rellenado con el código descargado.

Lo primero que se necesita es descargar el ejecutable en memoria con las funcionalidades que se han visto anteriormente. En caso de invocar esta función y no tener nada descargado se devuelve *NEED_EXECUTABLE.*

A esta función se le puede pasar el path del ejecutable al que se quiere suplantar o sino, se creará un proceso con el nombre del malware y se rellenará este proceso con el código descargado.

Cuando ya se ha configurado a qué ejecutable suplantar y se ha confirmado que se ha descargado el código, se llama a la función *runPE_main()* con ambos valores. Esta será la encargada de comprobar que los datos guardados en memoria se corresponden con un ejecutable comprobando la firma de este y, en caso de ser así, crear un proceso suspendido del ejecutable elegido y descargar de forma adecuada los datos del programa guardado en memoria en la memoria de este proceso (*CreateProcess(), VirtualAlloc(), GetThreadContext(), ReadProcessMemory(), WriteProcessMemory(), SetThreadContext(), ResumeThread()*).

En caso de que el proceso se cree y se rellene con los datos descargados en memoria correctamente se devuelve *CORRECT.* En caso de que ocurra algún fallo se devuelve *INCORRECT.*

- 142 Execute_file_other_process(path)

Simplemente usa *CreateProcess()* para crear un nuevo proceso invocando al ejecutable contenido en el path indicado. En caso de no indicar ningún path devuelve

PARAMETER, en el caso de crear el proceso correctamente devuelve *CORRECT,* en caso de no lograr crearlo devuelve *INCORRECT.*

4.3.6. Categoría 15X: Escalada de privilegios

Todo el código de esta categoría se encuentra en commands.cpp y su header.

- 151 Escalation_SilentCleanup(port)

Es una método de escalar privilegios descubierto hace muy poco y que se espera que se arregle en las próximas actualizaciones de Windows.

Esta vulnerabilidad descubierta se basa en aprovecharse de la tarea programada SilentCleanup la cual ejecuta con privilegios el programa %windir%\System32\cleanmgr.exe. Como se puede apreciar el path anterior utiliza una variable de entorno cuyo valor puede ser modificado escribiendo en el registro HKCU\Environment un valor de nombre windir y con valor el path al ejecutable que queramos que se llame cada vez que se lance la tarea acabando en path con " && REM \" de forma que todo lo que se lea a continuación del path se interpretará como comentario.

De esta forma, si escribimos en este registro el path al malware y pedimos que se ejecute dicha tarea, se llamará al malware ejecutándolo con privilegios.

Además, el malware está hecho de forma que en caso de tener privilegios se conectará al puerto *DEFAULT_ADMIN_PORT* para que así el proceso anterior del malware y el nuevo con privilegios no intenten conectarse al mismo puerto.

En caso de que todo funcione correctamente se devolverá *CORRECT* y se creará una conexión al puerto *DEFAULT_ADMIN_PORT.* En caso de que algo falle se devolverá *INCORRECT.*

4.3.7. Categoría 20X: Ransomware

Parte del código de esta categoría se encuentra en commands.cpp y su header, pero la parte más importante que contiene el código de los algoritmos de cifrado se encuentra en chipers.cpp y su header.

- 201 Cypher_file_byteInversion(location)

A esta función se le indica el path del archivo que se quiere cifrar y lo sobreescribe haciendo una inversión de byte sobre cada uno de los bytes que lo forman. En caso de no recibir ningún path devuelve *PARAMETER.* En caso de que funcione correctamente devuelve *CORRECT* y en caso de que algo falle, *INCORRECT.*

- 202 Cypher_folder_byteInversion(path)

Recorre el directorio que se le indica en el path, o en caso de que no se le indique ninguno, el directorio en el que está, y llama a la función *cypher_file_byteInversion()* indicándole el path del archivo. Devuelve *CORRECT* en caso de haberse ejecutado correctamente e *INCORRECT* en caso de que algo fallase. Sin embargo, en este caso que se ejecute correctamente no significa que hay podido cifrar algún archivo.

- 203 Cypher_file_byteCycle(location)

A esta función se le indica el path del archivo que se quiere cifrar y lo sobreescribe ejecutando el algoritmo de "byte cycle" sobre cada uno de los bytes que lo forman. En caso de no recibir ningún path devuelve *PARAMETER.* En caso de que funcione correctamente devuelve *CORRECT* y en caso de que algo falle, *INCORRECT.*

- 204 Cypher_folder_byteCycle(location)

Recorre el directorio que se le indica en el path, o en caso de que no se le indique ninguno, el directorio en el que está, y llama a la función *cypher_file_byteCycle()* indicándole el path del archivo. Devuelve *CORRECT* en caso de haberse ejecutado correctamente e *INCORRECT* en caso de que algo fallase. Sin embargo, en este caso que se ejecute correctamente no significa que hay podido cifrar algún archivo.

- 205 Configurar la clave de cifrado para rc4

Se le debe enviar como parámetro una clave de más de 8 digitos, en caso de ser así la guarda y devuelve *CORRECT.* En caso de no enviar ninguna clave, o una de un tamaño menor devuelve *KEY_NEEDED.*

- 206 Borrar la clave de cifrado para rc4

Se elimina de la memoria la clave de cifrado con el objetivo de que esta exista en el ordenador víctima lo menos posible. Devuelve siempre *CORRECT.*

- 207 Cypher_file_rc4(location)

A esta función se le indica el path del archivo que se quiere cifrar y lo sobreescribe ejecutando el algoritmo rc4 sobre cada uno de los bytes que lo forman. En caso de no recibir ningún path devuelve *PARAMETER.* En caso de que funcione correctamente devuelve *CORRECT* y en caso de que algo falle, *INCORRECT.*

- 208 Cypher_folder_rc4(location)

Recorre el directorio que se le indica en el path, o en caso de que no se le indique ninguno, el directorio en el que está, y llama a la función *cypher_file_rc4()* indicándole el path del archivo. Devuelve *CORRECT* en caso de haberse ejecutado correctamente e *INCORRECT* en caso de que algo fallase. Sin embargo, en este caso que se ejecute correctamente no significa que hay podido cifrar algún archivo.

4.3.8. Categoría 21X: Bucles Inifiniros

El código principal de esta categoría se encuentra en Ininite_loop.cpp y su header, aunque también hay parte del código en commands.cpp y su header.

- 211 InifiniteLoop_pi_dec()

Esta función crea una thread que ejecuta la función *pi_dec()*. En caso de que esta se cree correctamente devuelve *CORRECT,* en caso de que ocurra algún fallo devuelve *INCORRECT.*

La función *pi_dec()* no termina hasta calcular 999999999 decimales de pi. Además, cada *Num_dec_before_thread* decimales calculados inicia otra thread que ejecute la función *pi_dec().* De esta forma, los recursos que demanda el malware son cada vez

mayores hasta que el ordenador no puede soportarlo y se congela. Será necesario un reinicio el ordenador para volver a conseguir que este funcione.

4.3.9. Categoría 22X: Molestar

El código de esta categoría se encuentra en commands.cpp y su header.

- 221 Change_wallpaper(url)

Permite modificar el fondo de pantalla. Se le indica la URL de la que descargar la imagen, la descarga llamando a *download_image()* y la pone como fondo de pantalla usando la función *SystemParametersInfo()*. En caso de que no se le indica ninguna URL devuelve *PARAMETER.* En caso de que funcione correctamente devuelve *CORRECT,* y en caso de que algo falle devuelve *INCORRECT.*

- 222 Hide_ppal_icons()

Esta función no espera ningún parámetro, tan solo elimina de la pantalla la barra inferior con los iconos que se encuentra en los escritorios de Windows. Para ello hace uso de varias funciones: *FindWindowsEx(), ShowWindow(), findWindow(), RemoveMenu(), GetSystemMenu(), GetConsoleWindow(), DrawMenuBar(), GetConsoleWindow().*

La función devuelve siempre *CORRECT.*

- 223 Hide_file(path)

Añade el atributo de oculto al archivo indicado en el path pasado, para eso se sirve de *SetFileAttributes().* En caso de no recibir ningún path devuelve *PARAMETER.* En caso de que funcione correctamente devuelve *CORRECT,* y en caso de que algo falle devuelve *INCORRECT.*

- 224 Hide_files_in_folder(path)

Recorre el directorio que se le indica en el path, o en caso de que no se le indique ninguno, el directorio en el que está, y llama a la función *hide_file()* indicandole el path del archivo. Devuelve *CORRECT* en caso de haberse ejecutado correctamente e *INCORRECT* en caso de que algo fallase. Sin embargo, en este caso que se ejecute correctamente no significa que hay podido esconder algún archivo.

- 225 Go_X_years_to_the_future(years)

Se le indica el número de años que se quiere adelantar el reloj del ordenador que está ejecutando el malware, aunque para que esto funcione el malware necesita estar corriendo con privilegios.En caso de no recibir ningún número de años devuelve *PARAMETER.* En caso de que funcione correctamente devuelve *CORRECT,* y en caso de que algo falle devuelve *INCORRECT.*

- 225 Go_X_years_to_the_past(years)

Se le indica el número de años que se quiere atrasar el reloj del ordenador que está ejecutando el malware, aunque para que esto funcione el malware necesita estar corriendo con privilegios.En caso de no recibir ningún número de años devuelve *PARAMETER.* En caso de que funcione correctamente devuelve *CORRECT,* y en caso de que algo falle devuelve *INCORRECT.*

4.3.10. Categoría 999: Exit

El código 999 permite cerrar el proceso del malware usando la función *ExitProcess().*

5. Conclusiones

5.1 Resumen

Todo empezó en 1959 con la creación de un juego basado en la teoría de Von Neuman de autómatas capaces de replicarse y desde entonces, los programas de moralidad dudosa no han hecho más que aumentar en cantidad y complejidad hasta crear lo que hoy conocemos como malware.

Los primeros en crearse fueron simples virus capaces de reproducirse, aumentar su población en una máquina y realizar alguna acción consumiendo recursos de esta, tal y como haría un virus biológico.

A continuación se crearon los worms (gusanos). Pequeños programas capaces de expandirse por la red entrando en máquinas y ejecutando en ellas algún tipo de código. Probablemente con el objetivo de infectar las máquinas con un virus.

Hasta el año 2004 estos eran los malwares que predominaban y sus creadores tan solo los hacían por diversión y obtención de reconocimiento. Sin embargo, a partir de ese año se dieron cuenta que estos programas maliciosos podían servir para algo más que obtener reconocimiento, podían servir para obtener dinero.

A partir de entonces se desarrollaron una gran cantidad de nuevas categorías de malware, y los programas maliciosos empezaron a ser mucho más complejos que los anteriores pudiendo englobarse en varias categorías de malware a la vez. Se desarrollaron los Backdoors, Troyanos, RATs, Spyware, Ransomware, Exploit Kits, Botnets, RootKits... cada tipo de malware cumplía con una característica que permitía que la unión de varios tipos de malware en uno solo hicieran de este un ejecutable malicioso complejo, difícil de detectar y capaz de realizar una gran cantidad de acciones sobre la máquina.

De este modo, las características de un virus dan a un malware la capacidad de infectar otros ejecutables con código malicioso; la característica principal de un gusano es que es capaz de expandirse por la red; un backdoor es cualquier código malicioso capaz de autoejecutarse sin la necesidad de intervención por parte del usuario; un troyano es un malware capaz de esconderse en un ejecutable benigno. Un RAT permite el control total de la máquina víctima mediante una conexión de esta con el atacante, las capacidades de un Spyware permite a un malware espiar al usuario de la máquina víctima, un ransomware es capaz de secuestrar un ordenador y pedir dinero a cambio. Los exploit kits intentan vulnerar la seguridad los programas clientes que se usan para navegar por la web y los rootkits sirven para ocultar las acciones del malware al usuario.

Desde hace años ya no es complicado encontrarse con malwares que reúnen todas esas capacidades y más, de forma que muchas veces el atacante tiene un control sobre la máquina víctima superior al que tiene el usuario.

El objetivo de la prueba de concepto desarrollada para este proyecto es darse cuenta de que un malware que reúne varias de estas capacidades no es tan difícil de crear.

5.2 Valoración

El proyecto realizado ha sido muy útil para darme cuenta de lo amplio que es el mundo del malware y lo muy avanzado que está debido en un principio a gente que le gustaba

experimentar y crear cosas curiosas, y ahora gracias a la gran cantidad de mafias y colectivos que se lucran gracias a ellos.

Es una pena que unas tecnologías con tantas buenas posibilidades como los ordenadores(y en general los dispositivos electrónicos) y las redes se utilicen para llevar a cabo estas malas de acciones. Sin embargo, no dejan de ser curiosas las técnicas tanto sociales como tecnológicas que los cibercriminales utilizan para desarrollar su negocio.

Este trabajo se ha centrado en el estudio y creación de malware para intentar entender la complejidad técnica que estos programas maliciosos llevan dentro de sí. Y aunque considero que tan solo se ha estudiado la superficie de este amplio ámbito, ha sido suficiente para imaginarse las capacidades que puede tener alguien con más conocimientos en la creación de este tipo de programas.

Esto me ha permitido entender mejor la clara preocupación que muestra la gente dedicada profesionalmente al ámbito de la seguridad informática. Debido a que un grupo de gente experta en la creación de malware y fomentada a crearlos con objetivos poco morales podría tomar el control de millones de equipos que no les pertenecen sin que sus usuarios se dieran cuenta, o controlar unos pocos equipos especializados pudiendo provocar grandes tragedias.

Además, desgraciadamente esto son cosas que ya han ocurrido.

5.3 Líneas de continuación

Una prometedora forma de continuar con este trabajo sería el profundizar más en cómo se crean y qué caracteriza a nivel técnico a los malwares más comunes, con el objetivo de poder mejorar los antivirus actuales o crear herramientas que nos protejan de estas amenazas de forma paralela a los típicos antivirus.

Bibliografía

[1] http://www.pandasecurity.com/mexico/homeusers/security-info/classic-malware/

[2] https://malware2011.blogspot.com.es/2011/04/historia-del-malware.html

[3] https://prezi.com/ojpvwczmk3ro/virus-viernes-13/?webgl=0

[4] https://es.slideshare.net/ESETLA/historia-malware

[5] https://gusanomalo.blogspot.com.es/2012/04/i-love-you.html

[6] Libro “Malware: Fighting Malicious Code” de Ed Skoudis y Lenny Zeltser

[7] https://es.wikipedia.org/wiki/Malware

[8] https://blog.kaspersky.com.mx/clasificacion-de-malwares/1608/

[9] https://www.bleepingcomputer.com/tutorials/windows-alternate-data-streams/

[10] http://www.zonasystem.com/2010/04/clasificacion-de-virus-informaticos.html

[11] https://support.kaspersky.com/sp/viruses/general/614

[12] https://www.welivesecurity.com/la-es/2015/04/17/que-es-un-backdoor/

[13] http://technology.inquirer.net/31737/a-malware-classification

[14] https://www.trusteer.com/en/glossary/remote-access-trojan-rat

[15] http://aprenderinternet.about.com/od/SeguridadPrivacidad/a/Que-Es-Spyware.htm

[16] https://www.trendmicro.com/vinfo/us/security/definition/Ransomware

[17] http://www.pandasecurity.com/spain/mediacenter/malware/que-es-un-ransomware/

[18] https://www.trendmicro.com/vinfo/us/security/definition/Exploit-Kit

[19]https://www.osi.es/es/actualidad/blog/2014/03/14/que-es-una-botnet-o-una-red-zombi-de-ordenadores

[20] Libro “Practical malware analysis” de Michael Sikorski y Andrew Honig