



universidad
de león

Departamento de Matemáticas

**MÁSTER UNIVERSITARIO EN
INVESTIGACIÓN EN CIBERSEGURIDAD**

Trabajo de Fin de Máster

**ANÁLISIS DE CRIPTOGRAFÍA USADA POR
RANSOMWARE MEDIANTE
INSTRUMENTACIÓN DINÁMICA DE
BINARIOS**

**ANALYSIS OF CRYPTOGRAPHY USED BY
RANSOMWARE USING DYNAMIC BINARY
INSTRUMENTATION**

Autor: Ignacio Samuel Crespo Martínez

Tutor: Ricardo J. Rodríguez Fernández

(Septiembre, 2019)

UNIVERSIDAD DE LEÓN
Departamento de Matemáticas
MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN CIBERSEGURIDAD
Trabajo de Fin de Máster

ALUMNO: Ignacio Samuel Crespo Martínez

TUTOR: Ricardo J. Rodríguez Fernández

TÍTULO: Análisis de criptografía usada por ransomware mediante instrumentación dinámica de binarios

TITLE: Analysis of cryptography used by ransomware using dynamic binary instrumentation

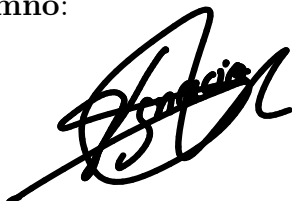
CONVOCATORIA: Septiembre, 2019

RESUMEN:

El ransomware es un tipo de malware que impide el acceso a los datos de la víctima. El ransomware se ha convertido en una de la mayores amenazas en la actualidad, adentrándose en los dispositivos de las víctimas a través de ingeniería social o simplemente mediante publicidad maliciosa. El ransomware puede dejar a las empresas sin su información más importante o incluso bloquear su producción ocasionando muchas pérdidas de dinero. Su creación no es relativamente difícil, ya que existen librerías en Windows que facilitan su construcción, por lo que la rentabilidad es máxima, ya que no requiere un esfuerzo enorme desarrollarlos y una parte de las víctimas acaba pagando el rescate pedido por recuperar su información. En este trabajo se realiza un análisis de la criptografía utilizada por el ransomware mediante técnicas de instrumentación dinámica de binarios. Así, observando las llamadas al sistema se pretende conocer características de la criptografía usada por el ransomware. Para ello, se han estudiado 99 muestras de diversas familias a lo largo de los últimos años. El objetivo es reconocer si existe algún tipo de preferencias en la criptografía usada por el malware, como en los tipos de cifrado o en las longitudes de las claves utilizadas. Los resultados muestran que el 40.4 % de las muestras de ransomware analizadas utilizan librerías de cifrado de Windows y que más de las mitad de los algoritmos usados tanto para la generación de claves como para el cifrado de los archivos son de tipo AES, siendo las parte restante de tipo RSA.

Palabras clave: Ransomware, análisis, instrumentación dinámica, CryptoAPI

Firma del alumno:



VºBº Tutor:



Digitally signed by
RODRIGUEZ FERNANDEZ
RICARDO JULIO - [REDACTED]
Date: 2019.09.03 23:31:41
+02'00'

Abstract

Ransomware is a type of malware that prevents a victim to access his/her data. Nowadays, ransomware has become one of more prevalent threat, entering victims' devices through social engineering or simply through malicious advertising. Ransomware may leave companies without their most important information or even block their production causing monetary losses. The development of ransomware is not relatively difficult, since there are libraries in Windows that facilitate its construction. Hence, the profitability is maximum since a huge effort to develop them is not required and a part of the victims pay the ransom to recover their data. In this work, we analyze the cryptography used by the ransomware using dynamic binary instrumentation techniques. Thus, observing the functions called by a ransomware sample we capture the characteristics of the cryptography used by the ransomware. We have studied 99 samples from various ransomware families over the past few years. The main goal of this project is to find the preferences in the cryptography used by malware (if any), such as in the encryption algorithms or in the key lengths. The results show that 40.4% of the ransomware samples analyzed use Windows encryption libraries and that more than half of them used AES for key generation and file encryption, being RSA the second preference.

Índice general

Abstract	I
Índice de figuras	IV
Índice de tablas	V
1. Introducción	1
2. Conceptos previos	6
2.1. CryptoAPI	6
2.2. Herramientas de instrumentación dinámica de binarios	8
2.2.1. DRAKVUF	11
2.2.2. Drltrace	11
2.2.3. WinAPIOverride	11
2.2.4. EasyHook	12
2.2.5. Frida	12
3. Sistema desarrollado	14
3.1. Sistema de análisis	15
3.2. Sistema de tratamiento de información	19
4. Experimentos y evaluación	23
5. Estado del arte	26
6. Conclusiones y trabajo futuro	29
Bibliografía	31
A. Desglose del trabajo	33
B. Integración de Cuckoo, Frida y ELK	35

Índice de figuras

1.1. Notificación WannaCry	2
1.2. Resultados de las respuestas ante el Ransomware	3
1.3. Coste del ransomware a lo largo de los años (2015 a 2019)	4
1.4. Estadísticas de los objetivos de los programas maliciosos en 2018	5
2.1. Arquitectura de la CryptoAPI	7
3.1. Diagrama de funcionamiento del sistema	14
3.2. Opciones de ejecución de análisis en Cuckoo	17
3.3. Ejemplo código javascript de instrumentación de una función	18
3.4. Ejemplo Reporting consola Web cuckoo	19
3.5. Estructura ELK	20
3.6. Filtrado de datos en Kibana	21
3.7. Representación de datos del análisis Kibana	21
4.1. Número de muestras analizadas ordenadas por año	23
4.2. Número de muestras analizadas que utilizan la CryptoAPI	24
4.3. Proveedores de servicios criptográficos usados por el ransomware	25
4.4. Algoritmos usados por las muestras de ransomware	25
A.1. Diagrama de Gantt	34

Índice de tablas

2.1. Tipos de proveedores de servicios criptográficos	9
2.2. Tipos de algoritmos de cifrado utilizados en CryptoAPI	10
3.1. Funciones instrumentadas de la CryptoAPI	22
C.1. Muestras de ransomware utilizadas para el análisis, ordenadas alfabéticamente	47

Capítulo 1

Introducción

El malware es un programa malicioso cuyo objetivo es infectar cualquier ordenador, teléfono inteligente o cualquier otro dispositivo electrónico para obtener datos confidenciales, robar dinero o cualquier otra actividad maliciosa. El ransomware es un tipo de malware que bloquea el acceso a los datos del ordenador de la víctima. Una vez impedido el acceso a los datos, notifica a la víctima que sus datos fueron bloqueados y exige un rescate para recuperar los datos. Un ejemplo se muestra en la Figura 1.1, correspondiente al ransomware WannaCry.

Existen dos tipos de ransomware [1]: *locker* y *crypto*. El primero de ellos impide el acceso al dispositivo de la víctima mediante el bloqueo de la pantalla o del teclado. El segundo bloquea el acceso a la información en el ordenador cifrando los archivos de la víctima. En ambos casos, siempre se pide un rescate económico para recuperar el acceso al dispositivo o a los archivos.

El ransomware es un tipo de malware bastante diferente al tradicional [2]. Su objetivo principal es hacer que sea imposible acceder a la información de la víctima. El ransomware no intenta permanecer de manera sigilosa en el sistema, sino que después de cifrar los archivos notifica a la víctima. Además, es relativamente fácil de desarrollar, ya que existen librerías de cifrado en Windows, como la CryptoAPI, que no requiere de conocimientos expertos de criptografía. Una librería proporciona un conjunto de funciones que pueden usarse para el desarrollo de programas y aplicaciones software. El ransomware utiliza tres vectores principales de ataque para comprometer los dispositivos de las víctimas: mediante spam/ingeniería social, descarga a través de publicidad maliciosas o mediante terceras herramientas de instalación de malware y botnets.



Figura 1.1: Notificación WannaCry

La clave necesaria para recuperar los datos puede que esté almacenada en el servidor del atacante o que haya sido generada aleatoriamente en el ordenador de la víctima y ni el atacante sepa cuál es la clave, por lo que pagar el rescate no asegura la liberación de los datos. Como se muestra en la Figura 1.2, el 53 % de las empresas o personas afectadas por el ransomware que deciden no pagar acaban recuperando sus archivos, el 19,6 % que pagaron el rescate no recuperaron sus datos, el 8 % que no pagaron no recuperaron sus archivos y el 19.1 % pagaron el rescate y recuperaron sus archivos [3].

El ransomware se ha convertido en un negocio lucrativo cada vez más popular entre los cibercriminales. Como se observa en la Figura 1.3, las pérdidas de dinero ocasionadas por el ransomware se van incrementando con el paso de los años.

El malware ataca preferentemente a los sistemas Windows. Como se puede ver en la Figura 1.4, más de la mitad de los programas maliciosos desarrollados en 2018 tenían como destino el sistema operativo Windows [4]. Por esta razón el presente trabajo se centra en Windows.

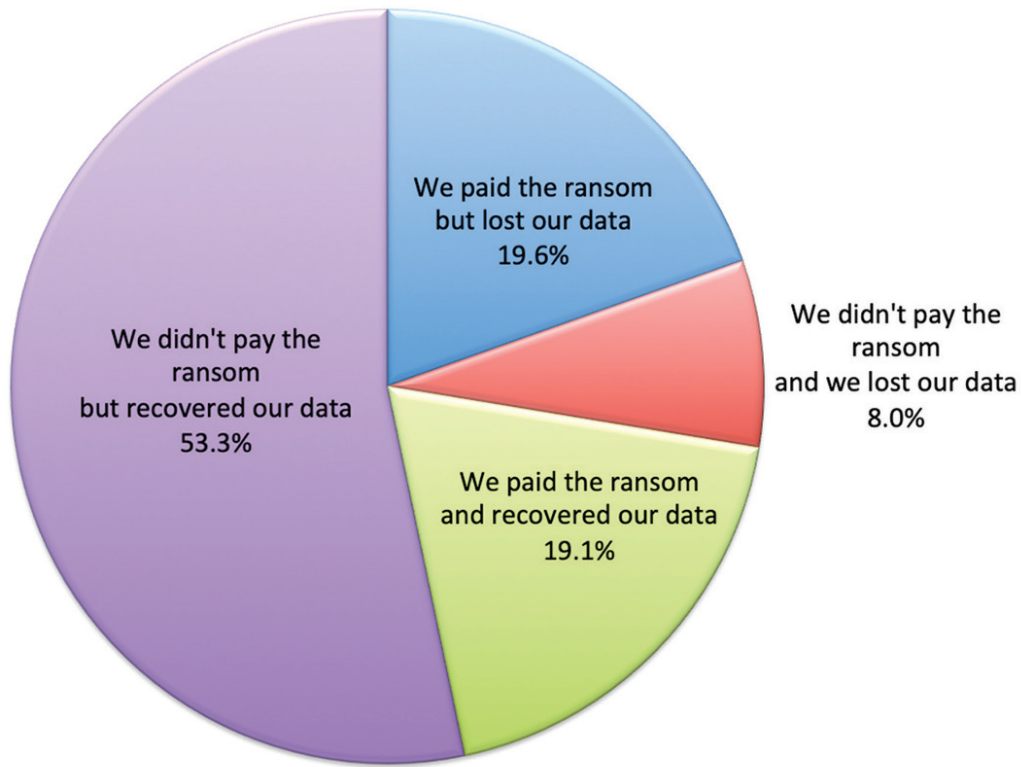


Figura 1.2: Resultados de las respuestas ante el Ransomware
Fuente: [3]

El objetivo de este proyecto es conocer las características de la criptografía usada (como los tipos de cifrado o las longitudes de las claves) por el ransomware, estudiando para ello muestras de diversas familias a lo largo de los últimos años mediante instrumentación dinámica de binarios. La instrumentación dinámica de binarios permite la inserción de código arbitrario durante la normal ejecución de una aplicación. En este caso, va a usarse para estudiar las llamadas a las funciones de la librería de cifrado de Windows usadas por las muestras de ransomware.

La estructura que se sigue en la presente memoria es la siguiente. En el Capítulo 2 se introducen los conocimientos previos para entender este proyecto. En el Capítulo 3, se explica detalladamente el sistema desarrollado. A continuación, en el Capítulo 4 se explican cuáles son los experimentos realizados y los resultados obtenidos. Después, en el Capítulo 5 se aborda el estado del arte con los principales trabajos relacionados con el proyecto. Por último, en el Capítulo 6 se exponen las conclusiones que se obtienen después de realizar el proyecto.

En cuanto a los anexos, el Anexo A muestra las tareas realizadas durante el proyecto. A continuación, en el Anexo B, se describen los pasos que hay que realizar

RANSOMWARE: TOTAL DAMAGE COST

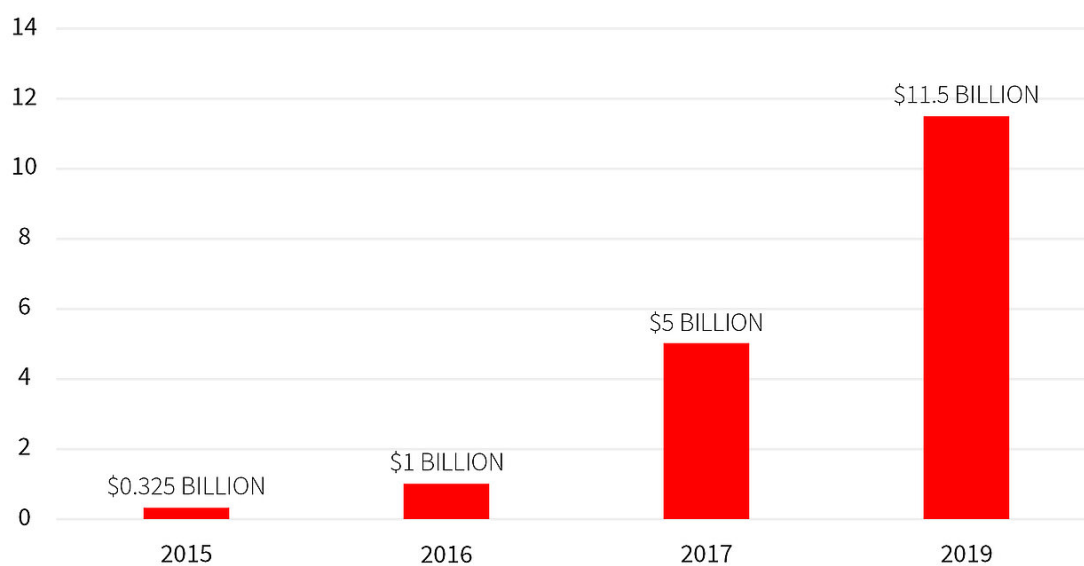


Figura 1.3: Coste del ransomware a lo largo de los años (2015 a 2019)

Fuente: <https://www.gdatasoftware.com/blog/2018/02/30439-2017-ransomware-lessons-learned>

para la construcción del sistema desarrollado y por último, en el Anexo C se muestra detalladamente el nombre y el hash de las muestras de ransomware utilizadas en este proyecto, categorizadas por el año en el que fueron descubiertas.

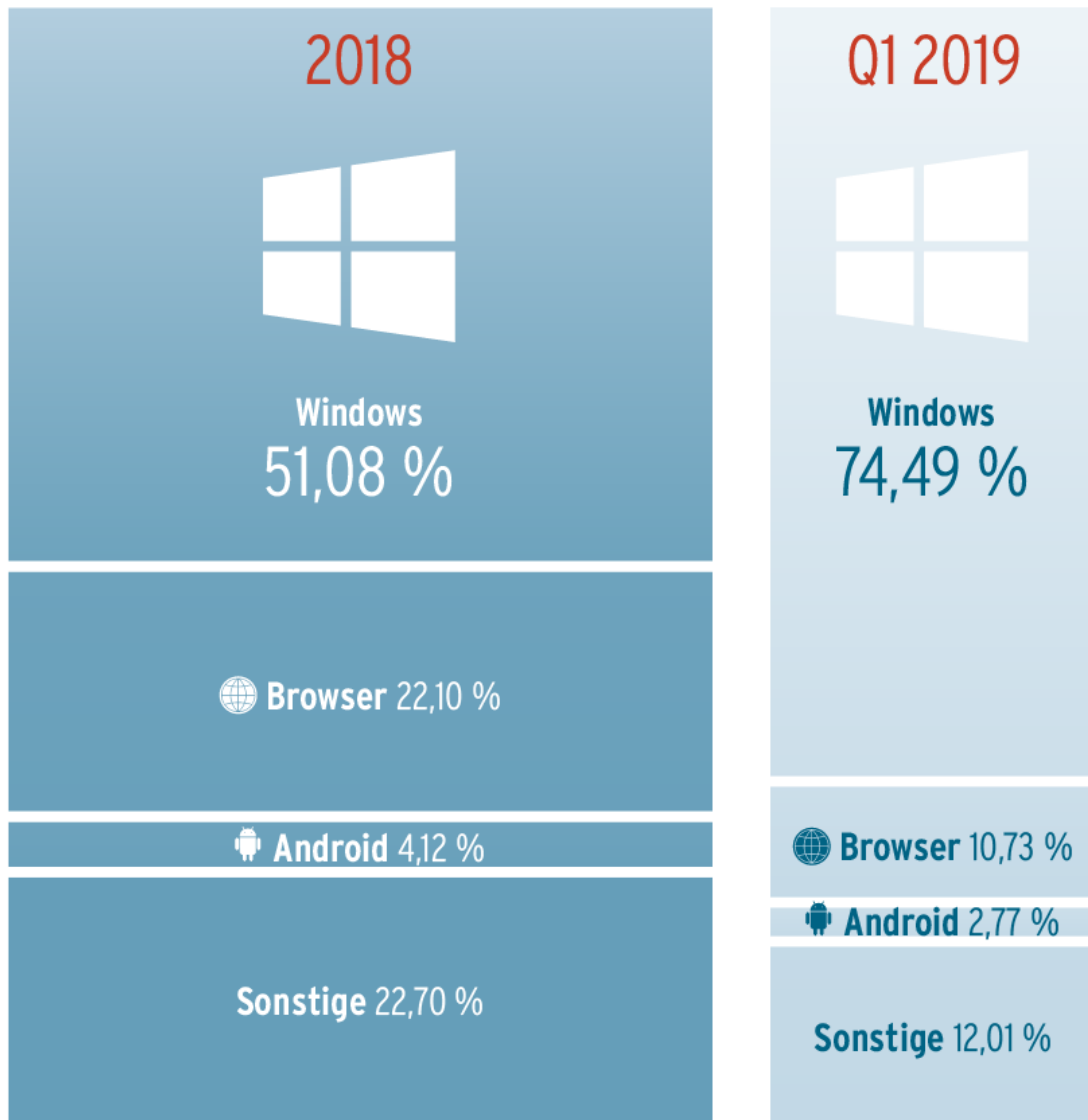


Figura 1.4: Estadísticas de los objetivos de los programas maliciosos en 2018
 Fuente: [4]

Capítulo 2

Conceptos previos

En esta sección se explican los conocimientos previos necesarios para la comprensión del proyecto desarrollado. Primero se explica la CryptoAPI, que es una librería nativa en Windows que proporciona servicios criptográficos a los desarrolladores; en segundo lugar se explican las diferentes herramientas de instrumentación dinámica de binarios probadas durante el proyecto para conocer cuál se adaptaba mejor al propósito de este proyecto.

2.1. CryptoAPI

La Microsoft CryptoAPI (MS CAPI) es una interfaz distribuida en los sistemas operativos Windows desde Windows 95, que proporciona servicios criptográficos a las aplicaciones de los usuarios. El objetivo es proporcionar una capa de abstracción para permitir a los desarrolladores securizar sus aplicaciones proporcionando un conjunto de funciones para cifrar o firmar digitalmente los datos sin ser especialistas en criptografía.

La versión 1 fue lanzada por primera vez como parte de navegador Internet Explorer 3.0 en 1996. Ese mismo año, Microsoft organizó una conferencia para recibir comentarios sobre cómo mejorar el sistema. A raíz de ello, se lanzó la versión 2 en 1997, vigente hasta la actualidad. La versión 1 contenía la funcionalidad necesaria para manejar funciones criptográficas. La versión 2 tiene la misma funcionalidad, pero además agrega la funcionalidad de administrar y usar certificados en comunicaciones seguras [5].

El sistema de CryptoAPI está formado por cinco áreas principales, que se muestran en la Figura 2.1. Estas áreas principales son:

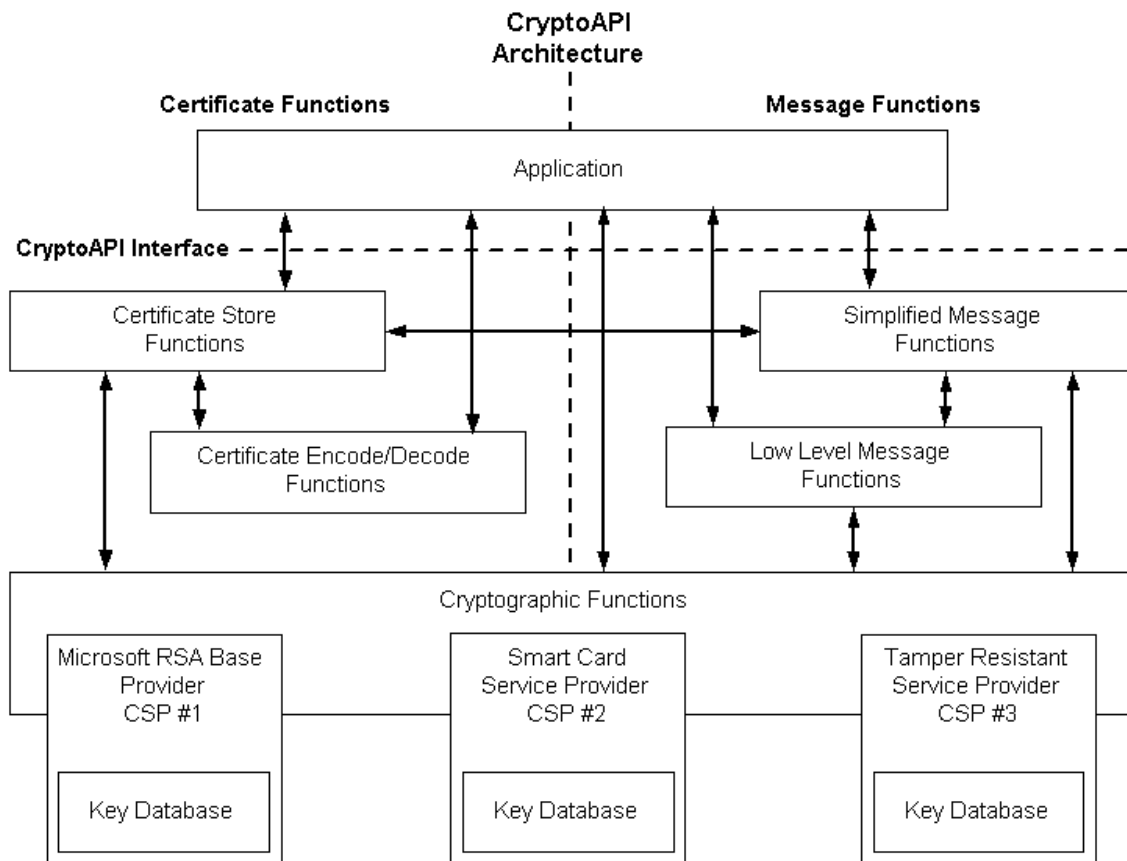


Figura 2.1: Arquitectura de la CryptoAPI

Fuente:

<https://docs.microsoft.com/en-us/windows/win32/secCrypto/Cryptoapi-system-architecture>

- Funciones de codificación/decodificación de certificados.
- Funciones para almacenar certificados.
- Funciones de mensaje simplificadas.
- Funciones de mensaje de bajo nivel.
- Funciones criptográficas base.

Una aplicación puede utilizar cualquiera de estas áreas llamando a las funciones que proporciona. El conjunto de estas funciones es lo que se conoce como CryptoAPI.

Ninguna operación criptográfica específica de un algoritmo de cifrado es parte de la CryptoAPI. En su lugar, estas operaciones son realizadas por módulos independientes conocidos como proveedores de servicios criptográficos (CSP). Las funciones tipo dependen de los CSP para proporcionar los algoritmos de cifrado y el alma-

cenamiento de seguro de cualquier sesión criptográfica o claves privadas o públicas que se generen.

Un proveedor de servicios criptográficos contiene todas las implementaciones de estándares y algoritmos criptográficos. Un CSP está compuesto por una librería de sistema (DLL) que implementa las funciones en CryptoSPI, que es una interfaz de programa del sistema. Esta interfaz facilita el desarrollo de nuevos CSP por parte de terceros, y su fácil interoperabilidad con cualquier aplicación que use la CryptoAPI.

La mayoría de los CSP contienen la implementación de todas sus funciones. Sin embargo, algunos CSP implementan sus funciones principalmente en un programa de servicio de Windows controlado por el administrador de control de servicios de Windows. Si un CSP no implementa sus propias funciones, la DLL actúa como una capa intermedia, facilitando la comunicación entre el sistema operativo y la implementación real del CSP [6].

Cuando una aplicación se conecta a un tipo de CSP, cada una de las funciones de la CryptoAPI funcionará de la manera predeterminada de la familia a la que corresponda ese tipo de CSP. Algunas aplicaciones pueden conectarse a más de un CSP a la vez, pero la gran parte de las aplicaciones generalmente utilizan un único CSP. Los tipos de proveedores predefinidos se muestran en la Tabla 2.1.

Respecto a los algoritmos de cifrado, el algoritmo de cifrado es utilizado en las funciones CryptoAPI y se especifica mediante el parámetro ALG_ID. Este parámetro indica el tipo de algoritmo que se va a utilizar tanto en la generación de claves como en el cifrado, ya que el algoritmo de cifrado viene determinado por la clave a utilizar. La Tabla 2.2 enumera los diferentes indicadores de algoritmos definidos actualmente. Un desarrollador que cree su propio CSP puede definir nuevos valores de algoritmos.

2.2. Herramientas de instrumentación dinámica de binarios

La instrumentación dinámica de binarios es una técnica para alterar el comportamiento de una función determinada mediante la inserción de código arbitrario durante la normal ejecución de una aplicación.

En el proyecto se probaron diferentes herramientas que utilizan esta técnica para saber cuál de ellas se ajusta mejor al propósito del proyecto y utilizarla para la

Tabla 2.1: Tipos de proveedores de servicios criptográficos

Fuente: <https://docs.microsoft.com/es-es/windows/win32/secCrypto/Cryptographic-provider-types>

Tipo	Propósito	Algoritmos soportados
PROV_RSA_FULL	Key Exchange	RSA
	Firma	RSA
	Cifrado	RC2, RC4
	Hashing	MD5, SHA
PROV_RSA_AES	Key Exchange	RSA
	Firma	RSA
	Cifrado	RC2, RC4, AES
	Hashing	MD2, MD4, MD5, SHA-1, SHA-2
PROV_RSA_SIG	Key Exchange	None
	Firma	RSA
	Cifrado	None
	Hashing	MD5, SHA
PROV_RSA_SCHANNEL	Key Exchange	RSA
	Firma	RSA
	Cifrado	RC4, DES, Triple DES
	Hashing	MD5, SHA
PROV_DSS	Key Exchange	None
	Firma	DSS
	Cifrado	None
	Hashing	MD5, SHA
PROV_DSS_DH	Key Exchange	DH
	Firma	DSS
	Cifrado	CYLINK_MEK
	Hashing	MD5, SHA
PROV_DH_SCHANNEL	Key Exchange	DH
	Firma	DSS
	Cifrado	DESTriple DES
	Hashing	MD5, SHA
PROV_FORTEZZA	Key Exchange	KEA
	Firma	DSS
	Cifrado	Skipjack
	Hashing	SHA
PROV_MS_EXCHANGE	Key Exchange	RSA
	Firma	RSA
	Cifrado	CAST
	Hashing	MD5
PROV_SSL	Key Exchange	RSA
	Firma	RSA
	Cifrado	Varios
	Hashing	Varios

Tabla 2.2: Tipos de algoritmos de cifrado utilizados en CryptoAPI

Fuente: <https://docs.microsoft.com/en-us/windows/win32/secCrypto/alg-id>

Identificador	Valor	Descripción
CALG_3DES	0x6603	Algoritmo de cifrado Triple DES.
CALG_3DES_112	0x6609	Cifrado triple DES de dos claves con una longitud de clave efectiva igual a 112 bits.
CALG_AES	0x6611	Estándar de cifrado avanzado (AES).
CALG_AES_128	0x660e	128 bit AES.
CALG_AES_192	0x660f	192 bit AES.
CALG_AES_256	0x6610	256 bit AES.
CALG_AGREEDKEY_ANY	0xaa03	Identificador de algoritmo temporal para identificadores de claves acordadas por Diffie-Hellman.
CALG_CYLINK_MEK	0x660c	Un algoritmo para crear una clave DES de 40 bits.
CALG_DES	0x6601	Algoritmo de cifrado DES.
CALG_DESX	0x6604	Algoritmo de cifrado DESX.
CALG_DH_EPHEM	0xaa02	Algoritmo temporal de intercambio de claves de Diffie-Hellman.
CALG_DH_SF	0xaa01	Diffie-Hellman almacena y reenvía el algoritmo de intercambio de claves.
CALG_DSS_SIGN	0x2200	Algoritmo de firma de clave pública DSA.
CALG_ECDH	0xaa05	Curva elíptica Diffie-Hellman algoritmo de intercambio de claves.
CALG_ECDH_EPHEM	0xae06	Algoritmo de intercambio de claves Diffie-Hellman de Curva elíptica temporal.
CALG_ECDSA	0x2203	Algoritmo de firma digital de Curva elíptica.
CALG_ECMQV	0xa001	Curva elíptica Menezes, Qu y Vanstone (MQV) algoritmo de intercambio de claves.
CALG_HASH_REPLACE_OWF	0x800b	Algoritmo hash de función unidireccional.
CALG_HUGHES_MD5	0xa003	Algoritmo hash Hughes MD5.
CALG_HMAC	0x8009	Algoritmo hash con clave HMAC.
CALG_KEA_KEYX	0xaa04	Algoritmo de intercambio de claves KEA (FORTEZZA).
CALG_MAC	0x8005	Algoritmo hash con clave MAC.
CALG_MD2	0x8001	Algoritmo hash MD2.
CALG_MD4	0x8002	Algoritmo hash MD4.
CALG_MD5	0x8003	Algoritmo hash MD5.
CALG_NO_SIGN	0x2000	Sin algoritmo de firma.
CALG_RC2	0x6602	Algoritmo de cifrado en bloque RC2.
CALG_RC4	0x6801	Algoritmo de cifrado de flujo RC4.
CALG_RC5	0x660d	Algoritmo de cifrado en bloque RC5.
CALG_RSA_KEYX	0xa400	Algoritmo de intercambio de clave pública RSA.
CALG_RSA_SIGN	0x2400	Algoritmo de firma de clave pública RSA.
CALG_SEAL	0x6802	Algoritmo de cifrado SEAL.
CALG_SHA	0x8004	Algoritmo de hash SHA.
CALG_SHA1	0x8004	Igual que CALG_SHA.
CALG_SHA_256	0x800c	Algoritmo de hash SHA de 256 bits.
CALG_SHA_384	0x800d	Algoritmo de hash SHA de 384 bits.
CALG_SHA_512	0x800e	Algoritmo de hash SHA de 512 bits.

creación del sistema a desarrollar. El sistema ha de permitir la monitorización de las llamadas a funciones de interés (principalmente, las funciones de la CryptoAPI) realizadas por la aplicación analizada.

2.2.1. DRAKVUF

Drakvuf es un sistema que permite mostrar todas las llamadas a funciones del sistema de los binarios ejecutados en una máquina virtual, sin la necesidad de instalar nada dentro de la máquina utilizada para el análisis [7]. Drakvuf se basa en la utilización de Rekall y Xen, además de sus propios módulos. Rekall es un framework para el análisis de la memoria, aprovechando la información de depuración que es proporcionada por los creadores del sistema operativo. Xen es un sistema de virtualización de código abierto que solo soporta Linux como sistema anfitrión.

Esta herramienta presenta unos requerimientos de hardware específicos sin los cuales no funciona. Debe ejecutarse sobre un equipo con procesador Intel con soporte de virtualización (Vt-x) y no soporta procesadores AMD. Esta herramienta se descartó por su alta dependencia de hardware concreto.

2.2.2. Drltrace

Drltrace es una herramienta construida sobre el DynamicRio, una plataforma de instrumentación dinámica [8] que tracea todas las llamadas a las funciones del sistema que hace la aplicación analizada. El inconveniente de esta aplicación es que registra todas las llamadas a las funciones del sistema, sin poder elegir qué funciones se quieren registrar.

Esto hace que la ejecución de la herramienta tarde mucho tiempo, a la vez de que una vez finalizado el análisis hay que hacer un escrutinio de los resultados para quedarse con lo que interesa. Además, no permite lanzar un análisis a un proceso que ya está en ejecución.

Todas estas desventajas hicieron que esta herramienta no fuera seleccionada.

2.2.3. WinAPIOverride

WinAPIOverride es una herramienta que monitoriza las llamadas a las funciones del sistema de procesos tanto de arquitecturas de 32 bits como de 64 bits [9]. Esta herramienta además es de pago, por lo que la versión que se probó no contenía todas las potenciales funcionalidades que puede contener la versión completa. La

versión gratuita solo permite ser instalada en un equipo Windows desde la versión XP no virtualizada, por lo que para montar un sistema de máquinas virtualizadas no serviría.

De nuevo, esta herramienta no permite elegir qué funciones se han de monitorizar, con lo que hace que el análisis se haga complicado al tener que ir buscando las funciones que interesan además de que solo permite monitorizar procesos que están ya en ejecución, por lo que algunas llamadas hechas al principio de la ejecución no quedan registradas.

Se descartó esta herramienta porque no permite ser instalada en máquinas virtualizadas y es de pago.

2.2.4. EasyHook

EasyHook es una herramienta de instrumentación dinámica que permite ver las llamadas a las funciones de Windows de los procesos que están en funcionamiento en el sistema, no permitiendo lanzar el proceso y después instrumentarlo, desde un entorno administrado [10].

Requiere de conocimiento de C++, dado que permite inyectar funciones de instrumentación escritas en C++ tanto en procesos de 32 bits como de 64 bits, pero únicamente de las aplicaciones creadas para .NET Framework 3.5/4.0+, así como librerías DLL nativas en Windows XP SP2 de 32 ó 64 bits, Windows Vista x64, Windows Server 2008 x64, Windows 7, Windows 8.1, y Windows 10.

Esta herramienta se descartó porque requiere un alto conocimiento de C++ , además de que solo funciona para aplicaciones creadas para .NET Framework 3.5/4.0+ o librerías DLL nativas.

2.2.5. Frida

Frida es una herramienta de instrumentación dinámica de código cuyo núcleo está escrito en C y que permite inyectar código JavaScript en el proceso analizado. Frida tiene así un completo acceso a la memoria del proceso, permitiendo la instrumentación de funciones e incluso la invocación de llamadas al sistema dentro del propio proceso [11]. Frida permite instrumentar varias librerías del sistema a la vez, o solamente funciones específicas de librerías, o una mezcla de ambas.

Al funcionar a través de JavaScript y Python, facilita el proceso de instrumentar cualquier proceso ya en ejecución o cualquier ejecutable que se quiera lanzar. Ade-

más, la integración en un sistema Windows es fácil, ya que solo se necesita tener Python instalado en el sistema.

Además de mostrar las llamadas al sistema o a funciones propias del ejecutable, permite mostrar los parámetros y sus valores de retorno, permitiendo también cambiar los valores de estos.

Los resultados de un análisis pueden ser manejados de varias formas, desde mostrar los resultados por línea de comandos hasta volcar la salida a un fichero, lo que le da una alta versatilidad.

Frida fue la herramienta elegida debido a su versatilidad, su facilidad de uso y porque permite elegir las funciones que se han de monitorizar.

Capítulo 3

Sistema desarrollado

En la siguiente sección se detalla el sistema desarrollado en este proyecto. Primero se describe el sistema de manera breve, para luego detallar en profundidad cada subsistema.

En la Figura 3.1 se muestra el diagrama del sistema, el cual tiene dos partes fundamentales bien diferenciadas. Por una parte está el sistema de análisis, que abarca tanto la recogida de muestras, su ejecución instrumentada y la generación de los resultados del análisis. Por otra parte, está el sistema de tratamiento de información de los resultados del análisis.

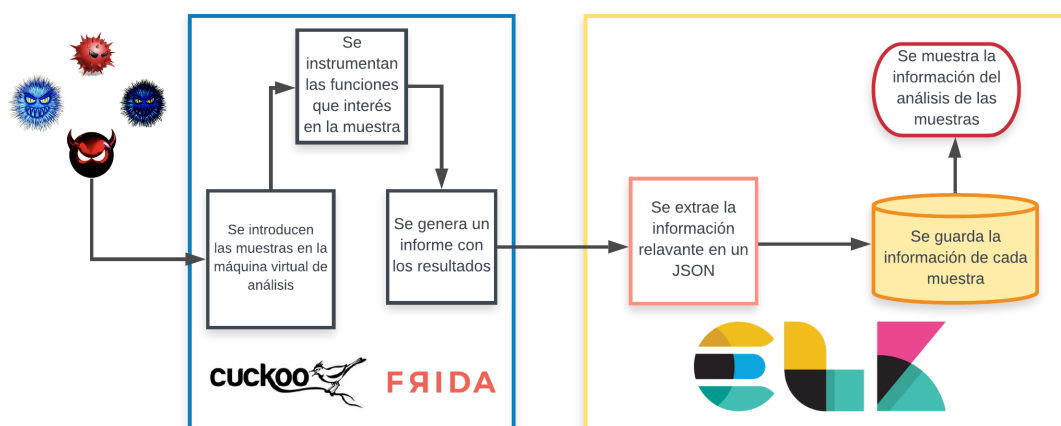


Figura 3.1: Diagrama de funcionamiento del sistema

3.1. Sistema de análisis

El sistema de análisis está compuesto por dos tecnologías principales, Cuckoo Sandbox y Frida. El entorno de instrumentación dinámica de Frida ya se ha explicado anteriormente en la Sección 2.2.5.

Cuckoo es un sistema avanzado de análisis malware automatizado, modular y de código abierto [12]. En este trabajo, Cuckoo se ha montado sobre una máquina Linux Ubuntu 18.04.2 LTS y trabaja juntamente con VirtualBox para realizar los análisis sobre una máquina virtual Windows 7 x86. Se eligió esta máquina por ligereza de hardware, ya que solo necesita 2GB de RAM para funcionar correctamente. En la máquina de análisis se instaló como software adicional Python 2.7 y Frida, requisitos del sistema de análisis.

Cuckoo está estructurado en módulos independientes que funcionan conjuntamente para realizar el análisis de las muestras. Los módulos de los que está compuesto Cuckoo son los siguientes:

- *Auxiliary*, que define los procedimientos que hay que realizar a la vez que se ejecuta el procesos de análisis.
- *Machinery*, que define cómo debe interactuar Cuckoo con el software de virtualización escogido. Si no es compatible de forma predeterminada, se debe escribir un propio módulo en Python de cómo debe interactuar Cuckoo.
- *Analysis Package* son clases estructuradas de Python que describen cómo el analizador de Cuckoo debe llevar a cabo el procedimiento de análisis para un fichero dado en la máquina de análisis.
- *Processing* permite definir la forma de analizar los resultados generados por el módulo *Analysis Package* y agregar información al contenedor que luego serán utilizados por los módulos de *Reporting*.
- *Reporting* utilizará los resultados de análisis procesados por los módulos de *Processing* para hacerlos accesibles y consumibles en diferentes formatos.

En el sistema desarrollado los módulos que se modificaron son *Analysis Package*, *Processing* y *Reporting*. A continuación se describen las modificaciones realizadas.

El módulo de *Analysis Package* coge una muestra de ransomware y la copia a la máquina virtual de análisis. Dentro de la máquina, es Frida el encargado de ejecutar la muestra mediante un script escrito en Python, a través del cual inyecta un código

JavaScript encargado de instrumentar las funciones de la CryptoAPI relevantes para el análisis.





En la interfaz web de Cuckoo, cada análisis se crea con unas opciones determinadas mostradas en la Figura 3.2. El paquete elegido es el exe, el cual se modificó para que tuviera la funcionalidad antes descrita. El tiempo de análisis máximo son 120 segundos, ya que en un análisis más corto cabe la posibilidad de que la muestra no se llegue a ejecutar de manera completa, y un análisis más largo aumentaría el tiempo de análisis del conjunto de muestras significativamente sin obtener mejores resultados. Las opciones restantes no se utilizan ya que aumentarían el tiempo de análisis y de nuevo la información resultante no sería relevante.

Por cada análisis de una muestra se crea en Cuckoo una estructura de carpetas y ficheros. Se genera un fichero de nombre “analysis.log”, que contiene el resultado de todas las acciones realizadas en la máquina virtual, y otro fichero “cuckoo.log”, que contiene todas las acciones realizadas durante el análisis. En una carpeta llamada “shots” se guardan las capturas de pantalla que se realizan durante el análisis y en otra carpeta de nombre “files”, se guarda el resultado del análisis de Frida.

Las funciones que se consideran más relevantes para ser instrumentadas [13] en cada análisis de una muestra son las que se muestran en la Tabla 3.1, junto con una breve descripción. Las funciones `CryptAcquireContextA()` y `CryptAcquireContextW()` indican qué proveedores de servicios criptográficas son utilizados. `CryptDeriveKey()`, `CryptGenKey()` y `CryptCreateHash()` señalan qué algoritmos son los utilizados para la creación de las claves y el cifrado de los ficheros. Las demás funciones son típicamente utilizadas por el ransomware [13] e indican que las muestras de ransomware usan la CryptoAPI de alguna otra manera (las anteriores funciones mencionadas no son de uso obligatorio si se usa la librería CRYPTOAPI).


Para cada función, se ha desarrollado un código en JavaScript para interceptar todas las llamadas y sus parámetros. Un ejemplo de este código de intercepción se muestra en la Figura 3.3. En la llamada a la función (función `onEnter`) se están escribiendo todos los parámetros a un fichero en particular, mientras que tras su ejecución no se hace nada (función `onLeave`).

Toda la información se guarda en archivo de nombre “frida.br” dentro de la propia máquina virtual. El archivo tiene la extensión .br, ya que archivos con extensiones comunes para archivos de texto, como pueden ser .txt, .log u otras extensiones conocidas, corren el riesgo de ser cifrados por el ransomware y por tanto no se obtendría el resultado del análisis.

cuckoo  Dashboard  Recent  Pending  Search


submit-file >> configure >> analyze

Configure your Analysis

Network Routing 

NONE DROP INTERNET INETSIM TOR

VPN via

Package  Priority

LOW MEDIUM HIGH

Timeout

SHORT MEDIUM LONG ...

60 120 300 ...

Options

- Remote Control Enables Guacamole UI for VM
- Enable Injection Enable behavioral analysis.
- Process Memory Dump
- Full Memory Dump If Volatility has been enabled, process an entire VM memory dump with it.
- Enforce Timeout
- Enable Simulated Human Interaction disable this feature for a better experience when using Remote Control

EXTRA OPTIONS [What can I use?](#)

F0 **I0** CryptoGod-85b0e492fb1019f3696714a33a62a5f1692af379a98c87de

Figura 3.2: Opciones de ejecución de análisis en Cuckoo

```

Interceptor.attach(Module.findExportByName(null, 'CryptAcquireContextA'),
{
  onEnter: function (args) {
    var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
    file.write('CryptAcquireContextA() attach\n');
    file.write("[+] HCRYPTPROV *phProv: " + args[0] + "\n");
    file.write("[+] LPCSTR      szContainer: " + args[1] + "\n");
    file.write("[+] LPCSTR      szProvider: " + args[2] + "\n");
    file.write("[+] DWORD       dwProvType: " + args[3] + "\n");
    file.write("[+] DWORD       dwFlags: " + args[4] + "\n\n");
    file.close();
  },
  onLeave: function (retval) {
  }
});

```

Figura 3.3: Ejemplo código javascript de instrumentación de una función

Cuando finaliza la ejecución del módulo *Analysis Package*, empieza la ejecución del módulo *Processing*, encargado de tratar la información contenida en el archivo resultante del análisis.

Este módulo construye un JSON, estándar basado en texto plano para el intercambio de información, con la información más relevante del análisis. Contiene los siguientes campos:

- *Name*: Contendrá el nombre de la muestra.
- *Hash*: El hash SHA256 de la muestra.
- *Date*: El año en el que fue descubierta la muestra.
- *Algorithm*: El algoritmo utilizado para la generación de clave y el cifrado de los ficheros.
- *ProviderType*: El tipo de proveedor de servicios criptográficos utilizado por la muestra.
- *Functions*: Las funciones de la CryptoAPI que utiliza la muestra en su ejecución.
- *Full_Log*: el resultado completo del análisis de la muestra.

Los campos *Name*, *Hash* y *Date* se extraen del nombre de la muestra, dado que se asume un cierto formato determinado. El formato empieza con el nombre de la muestra, después su hash SHA256 y por último el año en el que fue descubierta la muestra, todos esperados por un guión.

El contenido de cada JSON es almacenado en un único archivo denominado “samples.json”, que contiene todos los JSON creados a partir de las muestras ejecutadas. Este archivo es utilizado posteriormente por el sistema de tratamiento de información.

Inmediatamente después de que finalice la ejecución del módulo *Processing*, se ejecuta el módulo *Reporting*. Este se encarga de recoger la información recibida por parte del módulo *Processing* (archivo “samples.json”) y mostrarlo en la consola web de Cuckoo de la manera en la que se muestra en la Figura 3.4.

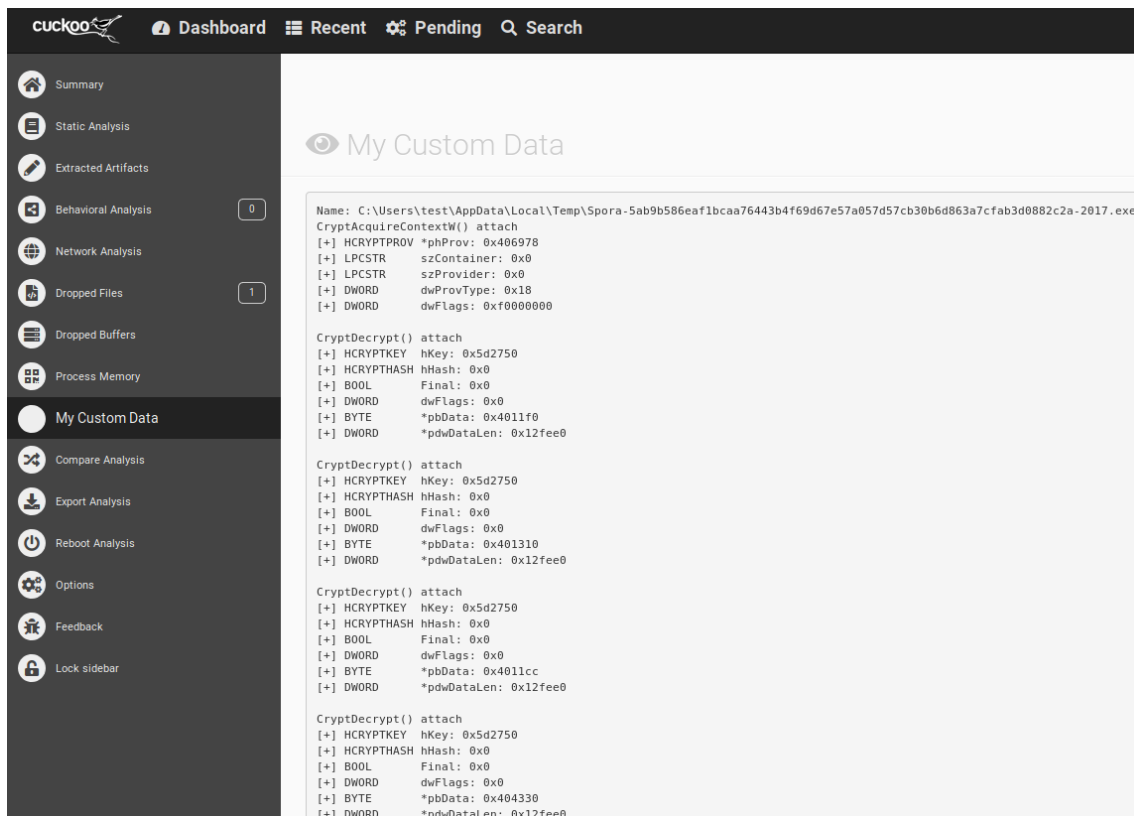


Figura 3.4: Ejemplo Reporting consola Web cuckoo

En el anexo B, se describen cuales son los pasos realizados para la integración entre Cuckoo Sandbox, Frida y ELK. El código está libremente accesible en <https://github.com/nacho94/Dynamic-instrumentation-system-with-Cuckoo-Frida-and-ELK> bajo licencia GPLv3.

3.2. Sistema de tratamiento de información

Este sistema está compuesto por un ELK que sigue la estructura de la Figura 3.5. ELK es un grupo de herramientas de código abierto formado por Logstash,

Elasticsearch y Kibana que funcionan conjuntamente para crear una herramienta que permite la monitorización y gestión en tiempo real de registros y logs.

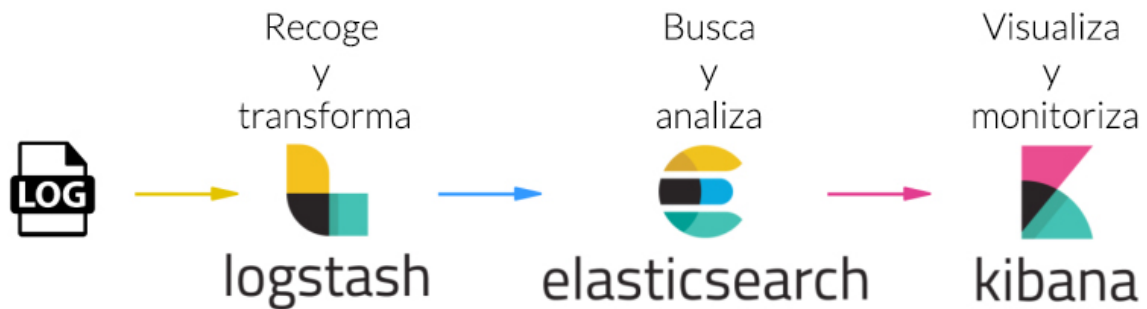


Figura 3.5: Estructura ELK

Fuente: <https://openwebinars.net/blog/que-es-elk-elasticsearch-logstash-y-kibana/>

En el sistema desarrollado, Logstash es el encargado de recolectar los datos almacenados en el archivo “samples.json” mencionado anteriormente y enviarlos a Elasticsearch. Elasticsearch indexa los datos recibidos por parte de Logstash parseando los campos para convertirlos en opciones de filtrado y búsqueda. Kibana crea un índice a partir de el índice de Elasticsearch y permite acceder a la información de cada muestra, pudiendose aplicar filtros y realizar búsquedas para mostrar la información que se considere oportuna en cada momento, como por ejemplo filtrar por el nombre de la muestra, el algoritmo que utiliza para cifrar o las funciones utilizadas de la CryptoAPI (véase la Figura 3.6).

Con todos los datos del análisis recolectados e indexados, Kibana permite crear diferentes visualizaciones, posibilitando representar los resultados del análisis total de las muestras, para sacar las conclusiones adecuadas, como se puede observar en la Figura 3.7.

Discover

Available fields

- Popular
- t Date
- @timestamp
- t @version
- t Flags
- t Hash
- t ProviderType
- t _id
- t _index
- # _score
- t _type
- t host
- t message
- t path
- t type
- t zFull_Log

Name	Algorithm	Functions
RansomWarrior	-	-
Cerber	-	CryptEncrypt(), CryptHashData(), CryptAcquireContextW(), CryptStringToBinaryA(), CryptGetKeyP
TorrentLocker	-	-
Cerber	-	CryptEncrypt(), CryptAcquireContextW(), CryptStringToBinaryA(), CryptGetKeyParam()
Meteoritan	-	-
Locky	-	CryptHashData(), CryptAcquireContextA()
smrss32	-	CryptAcquireContextW()
DontWorry	-	CryptAcquireContextW()
AVcrypt	-	-
Blackout	-	CryptEncrypt(), CryptHashData(), CryptAcquireContextW()
TotalWipeOut	-	CryptHashData(), CryptAcquireContextW()
Sigma	CALG_AES_256	CryptHashData(), CryptDeriveKey(), CryptDecrypt(), CryptAcquireContextW()
ShinoLocker	-	-
Ryuk	-	-
GPGWerty	-	-
WhiteRose	-	CryptEncrypt(), CryptHashData(), CryptAcquireContextW()
FLKR	-	-
GandCrabV4	CALG_RSA_KEYX	CryptEncrypt(), CryptExportKey(), CryptAcquireContextW(), CryptGenKey(), CryptGetKeyParam()
LittleFinger	RSA_KEY_EXCHANGE	CryptEncrypt(), CryptHashData(), CryptExportKey(), CryptAcquireContextW(), CryptGenKey(), Cry
E0E0	CALG_AES_256	CryptEncrypt(), CryptHashData(), CryptDeriveKey(), CryptAcquireContextA(), CryptGetKeyParam()

Figura 3.6: Filtrado de datos en Kibana

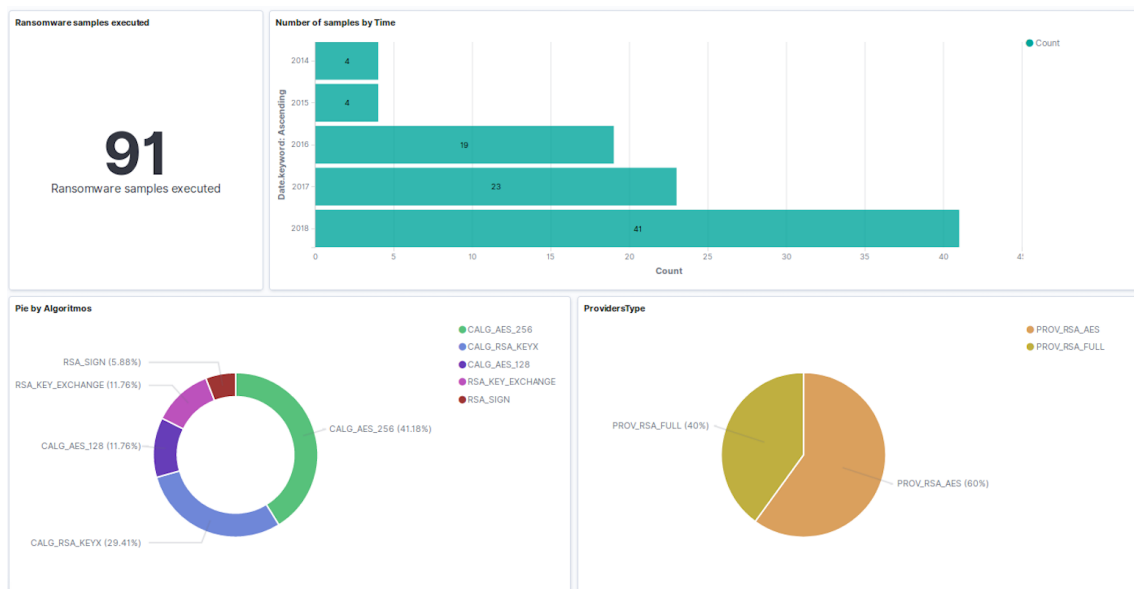


Figura 3.7: Representación de datos del análisis Kibana

Tabla 3.1: Funciones instrumentadas de la CryptoAPI

Funciones	Descripción
<code>CryptEncrypt()</code>	Función encargada de cifrar los datos. La clave designa el algoritmo utilizado para cifrar los datos.
<code>CryptHashData()</code>	Función que agrega datos a un objeto hash especificado.
<code>CryptDeriveKey()</code>	Función que genera claves de sesión criptográficas derivadas de un valor de datos base. Esta función garantiza que cuando se utiliza el mismo CSP y algoritmos, las claves generadas a partir de los mismos datos base son idénticas. Los datos base pueden ser una contraseña o cualquier otro dato de usuario.
<code>CryptDecrypt()</code>	Función que descifra los datos previamente cifrados mediante la función <code>CryptEncrypt()</code> .
<code>CryptAcquireContextA()</code>	Función utilizada para adquirir un identificador para un contenedor de claves particular, dentro de un CSP en particular. Este identificador devuelto se usa en llamadas a funciones de CryptoAPI que usan el CSP seleccionado.
<code>CryptAcquireContextW()</code>	Igual que <code>CryptAcquireContextA()</code> , pero el formato de las cadenas es UNICODE en vez de ASCII.
<code>CryptExportKey()</code>	Función que exporta una clave criptográfica o un par de claves de un CSP de manera segura.
<code>CryptGenKey()</code>	Función que genera una clave de sesión criptográfica aleatoria o un par de claves pública/privada.
<code>CryptSetKeyParam()</code>	Función que personaliza varios aspectos de las operaciones de una clave de sesión.
<code>CryptStringToBinaryA()</code>	Función que convierte una cadena formateada en una matriz de bytes.
<code>CryptGetKeyParam()</code>	Función que recupera los datos que gobiernan las operaciones de una clave.
<code>CryptCreateHash()</code>	Función que inicia el hash de un flujo de datos. Crea y devuelve a la aplicación de llamada un identificador para un objeto hash del CSP.

Capítulo 4

Experimentos y evaluación

En este capítulo se detallan los experimentos llevados a cabo y los resultados obtenidos.

Los experimentos se llevaron a cabo con un conjunto de 99 muestras de ransomware. En la Tabla C.1, disponible como anexo, se detallan el nombre y el hash de las muestras consideradas, categorizadas por el año en el que fueron descubiertas. Como se muestra en la Figura 4.1, en total se evaluaron 41 muestras del año 2018, 23 del año 2017, 19 del año 2016 y 4 de los años 2015 y 2014. La cantidad de muestras probadas son más cuanto más reciente es el año, ya que la posibilidad de encontrar las muestras de ransomware más viejas es más difícil.

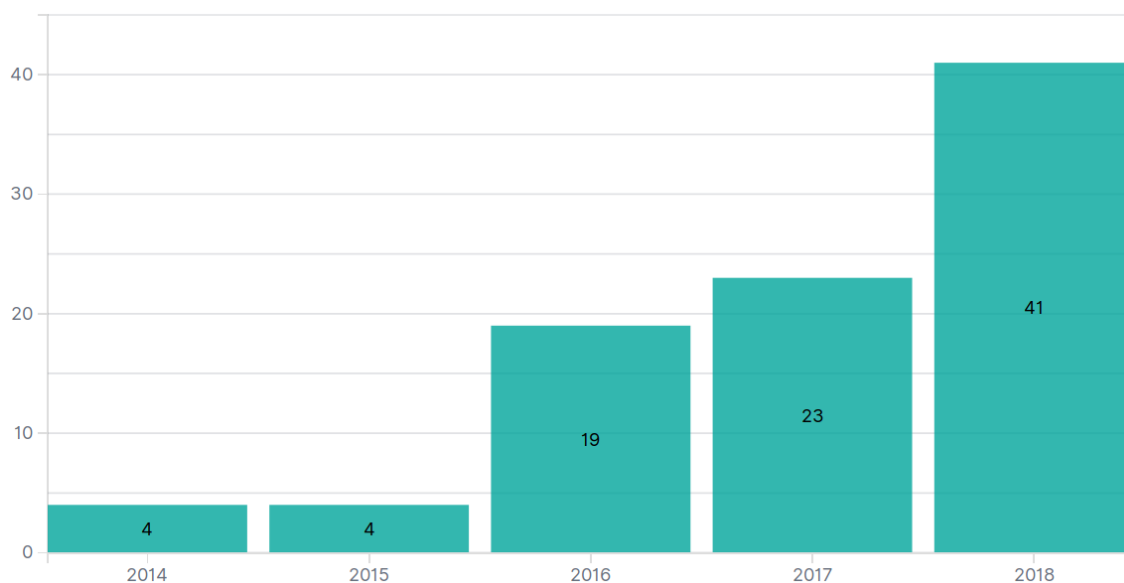


Figura 4.1: Número de muestras analizadas ordenadas por año

Para automatizar el proceso de análisis, se ejecuta un script en Python que se encarga de coger una a una las muestras de ransomware e introducir las en el sistema de análisis desarrollado.

Los resultados del análisis muestran que de las 99 muestras ejecutadas, 40 utilizaban la CryptoAPI para realizar sus ataques. Esas 40 muestras de ransomware se dividen en 17 muestras del año 2018, 16 del año 2017 y 7 del año 2016, como se ve en la Figura 4.2. Este incremento en los últimos años parece indicar la preferencia de la CryptoAPI por parte de los desarrolladores de malware.

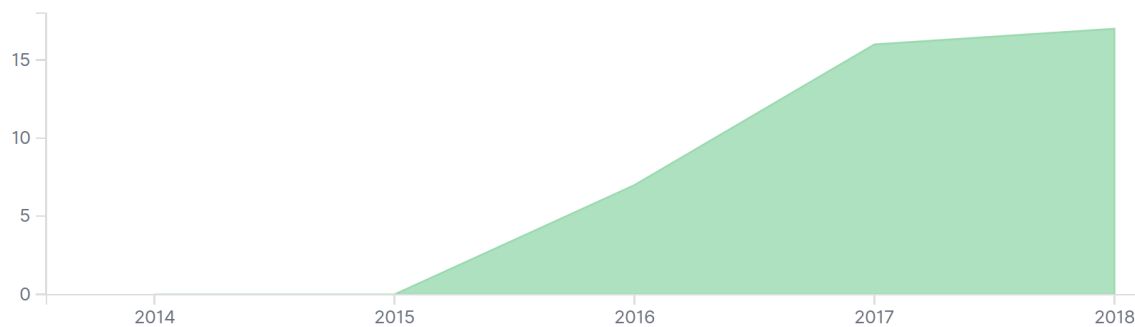


Figura 4.2: Número de muestras analizadas que utilizan la CryptoAPI

De las muestras que utilizaban la CryptoAPI, el 40 % de las muestras utilizaban el proveedor de servicios criptográficos PROV_RSA_FULL (véase la Figura 4.3), indicando que utilizaban solo algoritmos definidos en la Tabla 2.2 para la generación de claves y el cifrado de los archivos. El 60 % restante utilizaban el CSP PROV_RSA_AES, lo que indica que pueden utilizar AES aparte de RSA como algoritmo para la generación de claves y de cifrado de archivos.

Los resultados del análisis muestran también que más de la mitad de los algoritmos usados tanto para la generación de claves como para el cifrado de los archivos son de tipo AES (concretamente, un 52.94 %). Estos resultados se observan en la Figura 4.4, y están relacionados directamente con los datos anteriores en los que se ve que la mayoría de las muestras utilizaban el proveedor PROV_RSA_AES.

Con los datos del proveedor de servicios criptográficos y el algoritmo utilizado por las muestras de ransomware extraídos del análisis, se observa que solo se utilizan algoritmos RSA y AES, dejando de lado otros algoritmos débiles como pueden ser RC4 o MD5. El algoritmo involucrado en el cifrado no siempre está presente en el resultado del análisis de todas las muestras ejecutadas que utilizan la CryptoAPI, ya que pueden utilizar la CryptoAPI parcialmente, cifrando los archivos utilizando su propia clave ya generada o utilizando funciones propias.

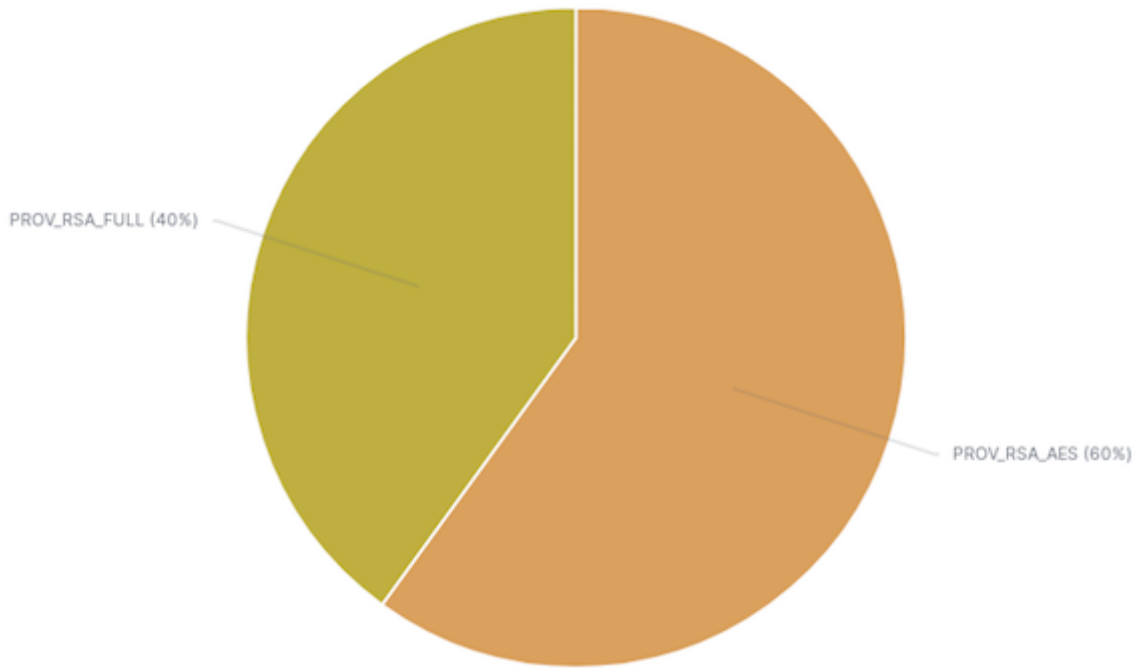


Figura 4.3: Proveedores de servicios criptográficos usados por el ransomware

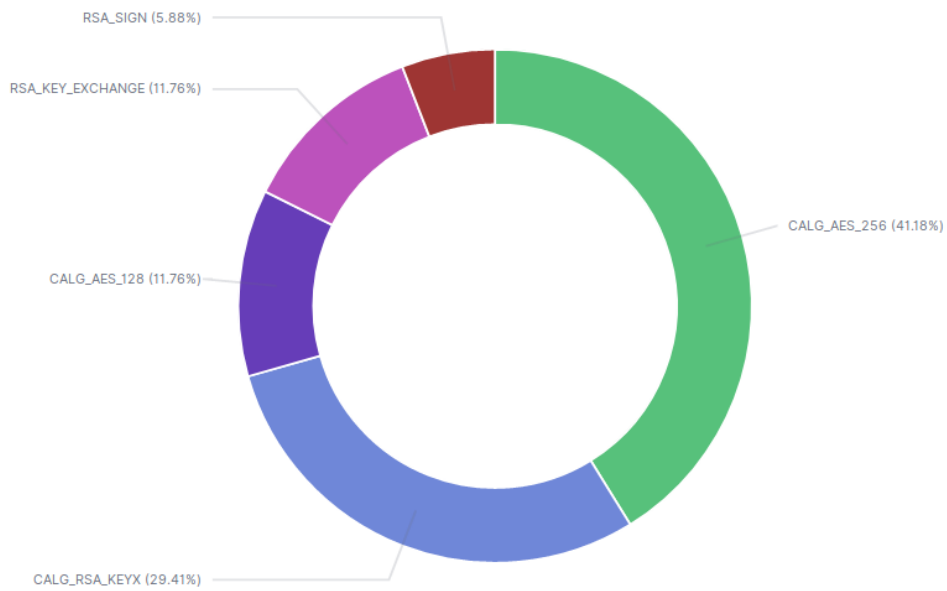


Figura 4.4: Algoritmos usados por las muestras de ransomware

Capítulo 5

Estado del arte

En este capítulo se lleva a cabo una revisión de los trabajos que guardan alguna relación con este proyecto.

Existen trabajos que presentan contramedidas contra el ransomware permitiendo a las víctimas de un ataque de este tipo descifrar sus archivos.

En [14], se presentan dos contramedidas. La primera se aprovecha del modo de operación débil al cifrar utilizado por algunos ransomware. La segunda intercepta las llamadas realizadas a la CryptoAPI de Microsoft mediante la implementación de un CSP, a diferencia de lo realizado en este proyecto que intercepta las llamadas a la CryptoAPI mediante instrumentación dinámica. Ambos métodos tienen que funcionar al mismo tiempo antes de que ocurra un ataque y ninguna de las dos contramedidas funciona de manera genérica. La evaluación realizada muestra unos resultados de un 50 % de eficacia.

En [15] los autores proponen ShieldFS, que monitoriza la actividad del sistema de ficheros de bajo nivel para mantener actualizado un conjunto de modelos adaptativos sobre la actividad del sistema a lo largo del tiempo. Así, cuando un proceso no concuerda con estos modelos, sus acciones se consideran maliciosas y los cambios originados en el sistema de ficheros son revertidos.

Los autores de [16] presentan un software denominado Redemption, una defensa que aumenta la resistencia del sistema operativo contra los ataques de ransomware. Este software requiere de una modificación del sistema operativo para mantener un búffer de todas las operaciones de entrada/salida (E/S). Simultáneamente, el sistema monitoriza los patrones de solicitud de E/S de las aplicaciones para detectar comportamientos semejantes a los de un ransomware. Si el sistema observa patrones

que indican actividad de ransomware, los procesos involucrados se pueden detener, incluso finalizar y restaurar los datos. La evaluación presentada muestra cero pérdida de datos contra las familias de ransomware actuales sin afectar a la experiencia de usuario.

En [17], los autores presentan un enfoque en el que no solo se pretende detectar de manera temprana al ransomware, sino bloquear por completo sus acciones. Para ello, implementa un conjunto de archivos trampa en el entorno de destino. En lugar de ser archivos normales, los archivos trampa son tipo FIFO, es decir, los archivos pueden ser abiertos por cualquier proceso para leer o escribir, pero tiene que estar abierto simultáneamente por ambos extremos. Es decir, cuando el ransomware abre el archivo para leer se queda bloqueado hasta que otro proceso abra ese mismo archivo para escribir en él. Aparte de frustrar el objetivo del ransomware, la solución presentada puede lanzar contramedidas de manera automática para acabar con el ataque. La solución presentada en [17] ha sido creada para plataformas Linux como prueba de concepto, denominada R-Locker, no necesita de privilegios especiales y no afecta al funcionamiento general del sistema.

En [18] se presenta PayBreak, una propuesta que se basa en que el cifrado seguro de los archivos emplea un cifrado híbrido donde se usan claves de sesión simétricas en el ordenador de la víctima. PayBreak observa el uso de estas claves y las mantiene bajo resguardo, pudiendo descifrar posteriormente los archivos. PayBreak se evaluó contra 20 familias de ransomware, demostrando que el sistema puede restaurar todos los archivos cifrados en 12 muestras de esa familias con una sobrecarga de rendimiento inapreciable.

Otros artículos exploran los esfuerzos de investigación en ransomware y resaltan los desafíos y problemas a los que se enfrenten las soluciones existentes.

En [19] se presenta una nueva taxonomía del ransomware desde diferentes perspectivas. Posteriormente, se definen los factores que conducen a un ataque de ransomware, incluyendo soluciones de análisis, prevención, predicción y detección.

Los autores de [20] presentan los resultados de un estudio a largo plazo de los ataques de ransomware observados entre los años 2006 y 2014, y también proporcionan una visión de cómo los ataques de ransomware han evolucionado, analizando 1.359 muestras pertenecientes a 15 familias de ransomware diferentes. Las evaluaciones realizadas muestran que a pesar de la mejora continua en la técnicas de cifrado en las principales familias de ransomware, una gran cantidad de muestras solo bloquean

el acceso al escritorio del ordenador de la víctima o intentan cifrar los archivos de la víctima utilizando técnicas superficiales.

Además, en [20] también se muestra que monitorizando la actividad anormal del sistema de ficheros es posible diseñar un sistema que pueda detener una gran cantidad de ataques de ransomware, incluso con aquellos ataques que utilizan cifrados fuertes. También sugiere que al observar las solicitudes de E/S y proteger la tabla maestra de archivos, que es un archivo en el que se almacena la información que le indica al sistema operativo cómo manejar el archivo o directorio contenido en el sistema de archivos NTFS de Windows, es posible detectar y prevenir un gran número de ataques de ransomware.

La mayoría de los trabajos presentan soluciones para protegerse contra el ransomware detectando anomalías en la actividad normal de las aplicaciones, pero ninguno se centra en la criptografía usada por el ransomware. Con este proyecto se pretende aportar una visión de las características de la criptografía usada por el ransomware estudiando muestras de diversas familias a lo largo de los últimos años.

Capítulo 6

Conclusiones y trabajo futuro

Este capítulo se presentan las conclusiones alcanzadas de la elaboración de este proyecto, así como los trabajos futuros que pueden ser realizados a partir de este proyecto.

En este trabajo se ha llevado a cabo un análisis de las características de la criptografía usada (como los tipos de cifrado o las longitudes de la claves utilizadas) por el ransomware estudiando muestras de diversas familias a lo largo de los últimos años mediante instrumentación dinámica de binarios, estudiando las funciones de la librería de cifrado de Windows (llamada CryptoAPI) normalmente usada por las muestras de ransomware.

Después de los resultados obtenidos, se observa que la CryptoAPI sigue siendo usada por el ransomware a pesar de que está en desuso y que Microsoft recomienda que se utilice la siguiente versión de la librería, llamada Cryptography API: Next Generation (CNG). Como es una librería cuyo uso está desaconsejado, tal vez tendría cabida su eliminación permanente en los sistemas operativos más modernos de Microsoft, obligando a los desarrolladores de software a utilizar la siguiente versión. Esta eliminación supondría que algunas aplicaciones dejaran de funcionar. Si se mantiene, como es una librería de fácil uso, se pueden construir muestras de ransomware de manera sencilla y sin tener dependencias de librerías externas.

También se observa que los algoritmos utilizados tanto para la generación de claves como para el cifrado de los archivos son RSA y AES, siendo los algoritmos más robustos soportados por esta librería. Esto indica que los desarrolladores de ransomware no se arriesgan a que sus ataques puedan quedar sin efecto al usar esquemas de cifrado débiles.

Como trabajo futuro conviene realizar este análisis pero con la evolución de CryptoAPI, la CNG, comparando los resultados con los de este proyecto. Los resultados de análisis de este trabajo también pueden servir para definir patrones de comportamiento de ransomware en base a la traza de las funciones criptográficas llamadas, desarrollando alguna herramienta que monitorice la actividad de las aplicaciones y detenga su ejecución, en caso de detectar alguna actividad sospechosa.

Bibliografía

- [1] V. Kotov and M. S. Rajpal, “Understanding Crypto-Ransomware,” Bromium, Tech. Rep., 2014.
- [2] A. Kharraz, W. Robertson, and E. Kirda, “Protecting against Ransomware: A New Line of Research or Restating Classic Ideas?” *IEEE Security Privacy*, vol. 16, no. 3, pp. 103–107, May 2018.
- [3] A. Fagioli, “Zero-day recovery : the key to mitigating the ransomware threat,” *Computer Fraud & Security Bulletin*, vol. 2019, no. 1, pp. 6–9, 2019.
- [4] “La situación de riesgo se agrava: todos los datos en el informe sobre seguridad de AV-TEST 2018/2019 | AV-TEST.” [Online]. Available: <https://www.av-test.org/es/noticias/la-situacion-de-riesgo-se-agrava-todos-los-datos-en-el-informe-sobre-seguridad-deav-test20182019>
- [5] A. Fuchsberger, “Microsoft CryptoAPI,” *Information Security Technical Report*, vol. 2, no. 2, pp. 74 – 77, 1997.
- [6] “Cryptographic Service Providers - Windows applications | Microsoft Docs.” [Online]. Available: <https://docs.microsoft.com/es-es/windows/win32/seccrypto/cryptographic-service-providers>
- [7] “DRAKVUF™ Black-box Binary Analysis System.” [Online]. Available: <https://drakvuf.com/>
- [8] “mxmssh/drltrace.” [Online]. Available: <https://github.com/mxmssh/drltrace>
- [9] “WinAPIOverride : Free Advanced API Monitor, spy or override API or exe internal functions.” [Online]. Available: <http://jacquelin.potier.free.fr/winapioverride32/http://jacquelin.potier.free.fr/winapioverride32/index.php>
- [10] “EasyHook.” [Online]. Available: <https://easyhook.github.io/>
- [11] “Frida.” [Online]. Available: <https://www.frida.re/>

- [12] “Cuckoo Sandbox - Automated Malware Analysis,” 2014. [Online]. Available: <https://cuckoosandbox.org/>
- [13] “CryptoAPI in Malware - Blueliv.” [Online]. Available: <https://www.blueliv.com/blog/research/cryptoapi-in-malware/>
- [14] A. Palisse, H. Le Boudier, J.-L. Lanet, C. Le Guernic, and A. Legay, “Ransomware and the Legacy Crypto API,” in *The 11th International Conference on Risks and Security of Internet and Systems - CRiSIS 2016*, ser. Risks and Security of Internet and Systems, F. Cuppens, N. Cuppens, J.-L. Lanet, and A. Legay, Eds., vol. 10158. Roscoff, France: Springer, Sep. 2016, pp. 11–28.
- [15] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barengi, S. Zanero, and F. Maggi, “ShieldFS: A Self-healing, Ransomware-aware Filesystem,” in *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ser. ACSAC ’16. New York, NY, USA: ACM, 2016, pp. 336–347.
- [16] A. Kharraz and E. Kirda, “Redemption: Real-Time Protection Against Ransomware at End-Hosts,” in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis, and M. Antonakakis, Eds. Cham: Springer International Publishing, 2017, pp. 98–119.
- [17] N. Engineering and S. Group, “R-Locker : Thwarting ransomware action through a honeyfile-based approach,” *Computers & Security*, vol. 73, pp. 389–398, 2018.
- [18] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, “PayBreak: Defense Against Cryptographic Ransomware,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS ’17. New York, NY, USA: ACM, 2017, pp. 599–611.
- [19] B. A. S. Al-rimy, M. A. Maarof, S. Zainudeen, and M. Shaid, “Ransomware threat success factors , taxonomy , and countermeasures : A survey and research directions,” *Computers & Security*, vol. 74, pp. 144–166, 2018.
- [20] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, “Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks,” in *Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148*, ser. DIMVA 2015. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 3–24.

Anexo A

Desglose del trabajo

El proyecto tiene una duración de 759 horas y se divide en las tareas que se muestran en la Figura A.1. La primera tarea es la de investigación (147 horas), en la que se recopila la información relacionada con el proyecto y se estudian todos los trabajos relacionados. Después, en la segunda tarea (97 horas) se estudian y prueban diferentes herramientas de instrumentación dinámica de binarios y se elige cuál es la más adecuada a este proyecto. A continuación, en la tercera tarea (81 horas), se comienza a recopilar información sobre la CryptoAPI y las diferentes tecnologías involucradas en el desarrollo del sistema de análisis. Finalmente, en la cuarta tarea (158 horas), se desarrolla el sistema de análisis con sus pertinentes pruebas para corroborar que funciona correctamente.

Posteriormente, en la quinta tarea (84 horas), se comienza a realizar las experimentaciones con las muestras de ransomware, ejecutándolas y modificando el sistema en función de las ejecuciones de estas. Luego, en la sexta tarea (63 horas) se evalúan los resultados y por último, en la tarea final (129 horas) se realiza la memoria del proyecto.

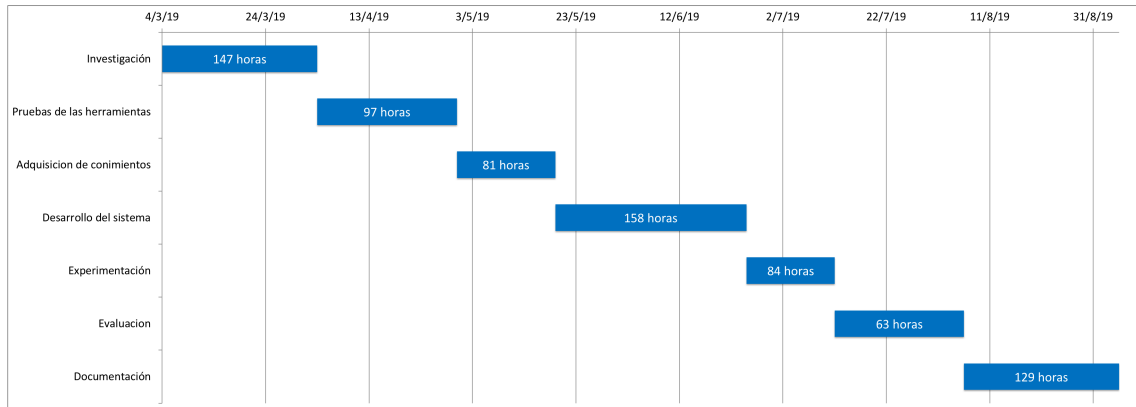


Figura A.1: Diagrama de Gantt

Anexo B

Integración de Cuckoo, Frida y ELK

Este anexo explica los pasos necesarios para realizar la integración entre Cuckoo Sandbox, Frida y ELK.

La instalación de todo el conjunto se ha llevado a cabo sobre un Linux Ubuntu 18.04.2 LTS, en la cual se requiere tener instalado Python 2.7 y VirtualBox. Como máquina virtual, se ha usado una Windows 7 x86 en la que tiene que estar instalado el entorno de análisis Cuckoo Sandbox, además de Python 2.7 y Frida. Las versiones usadas en este proyecto son Cuckoo Sandbox 2.0.7, VirtualBox 6.0.8, Python 2.7, Frida 12.6.11 y ELK 7.3.0.

En el resto de la sección se hacen referencia a 2 carpetas diferentes de la siguiente manera:

- **CUCKOO:** “/usr/local/lib/python2.7/dist-packages/cuckoo/” (ruta de instalación de Cuckoo).
- **CWD:** “/opt/cuckoo/”.

Primero se modifican los módulos *Analysis Package* exe y zip, que están localizados en “CWD/analyzer/windows/modules/packages”.

El módulo exe es el archivo “exe.py” y tiene el Código B.1. Este es el encargado de lanzar el análisis.

```
1 import os
2 import shlex
3 import sys
4 import logging
5
6 from lib.common.abstracts import Package
7 from lib.common.results import NetLogFile
8 from lib.common.results import upload_to_host
```

```

9
10 log = logging.getLogger(__name__)
11
12 class Exe(Package):
13     """EXE analysis package."""
14
15     PATHS = [
16         ("HomeDrive", "Python24", "python.exe"),
17         ("HomeDrive", "Python25", "python.exe"),
18         ("HomeDrive", "Python26", "python.exe"),
19         ("HomeDrive", "Python27", "python.exe"),
20         ("HomeDrive", "Python32", "python.exe"),
21         ("HomeDrive", "Python33", "python.exe"),
22         ("HomeDrive", "Python34", "python.exe"),
23     ]
24
25     def start(self, path):
26         args = self.options.get("arguments", "")
27
28         name, ext = os.path.splitext(path)
29         if not ext:
30             new_path = name + ".exe"
31             os.rename(path, new_path)
32             path = new_path
33
34         pathdir = os.getcwd()
35         pathpython = pathdir + "\\modules\\packages\\zip.py"
36         python = self.get_path("Python")
37         log.debug("debug " + python)
38         args = [pathpython] + [path]
39         return self.execute(python, args=args, trigger="file:%s" % pathpython)
40
41     def finish(self):
42
43         return upload_to_host("C:\\Users\\test\\Desktop\\frida.br", os.path.join("
         files", "frida.log"))

```

Código B.1: Fichero “exe.py”

El módulo zip es el archivo “zip.py” que contiene el Código B.2 y es el encargado de lanzar Frida.

```

1 import frida
2 import sys
3 import os
4
5 print ("Name: " + sys.argv[1] + "\n")
6 f = open ("C:\\Users\\test\\Desktop\\frida.br", "a")
7 f.write("Name: " + sys.argv[1] + "\n")
8 f.close()
9
10 pid = frida.spawn(sys.argv[1])
11 session = frida.attach(pid)
12
13 pathdir = os.path.dirname(os.path.realpath(__file__))
14 pathjava = pathdir + "\\instrumentation.js"

```

```

15 contents = open(pathjava).read()
16 script = session.create_script(contents.decode('utf-8'))
17 script.load()
18 frida.resume(pid)
19 sys.stdin.read()

```

Código B.2: Fichero “zip.py”

Una vez modificados los módulos de *Analysis Package*, se crea el fichero “instrumentation.js” con el Código B.3 en la misma carpeta de los módulos de *Analysis Package*. Este fichero contiene las funciones de la CryptoAPI que se instrumentan.

```

1  Interceptor.attach(Module.findExportByName(null, 'CryptEncrypt'),
2  {
3      onEnter: function (args) {
4          var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
5          file.write('CryptEncrypt() attach\n');
6          file.write("[+] HCRYPTKEY hKey: " + args[0] + "\n");
7          file.write("[+] HCRYPTHASH hHash: " + args[1] + "\n");
8          file.write("[+] BOOL Final: " + args[2] + "\n");
9          file.write("[+] DWORD dwFlags: " + args[3] + "\n");
10         file.write("[+] BYTE *pbData: " + args[4] + "\n");
11         file.write("[+] DWORD *pdwDataLen: " + args[5] + "\n");
12         file.write("[+] DWORD dwBufLen: " + args[6] + "\n\n");
13         file.close();
14     },
15     onLeave: function (retval) {
16     }
17 });
18
19 Interceptor.attach(Module.findExportByName(null, 'CryptHashData'),
20 {
21     onEnter: function (args) {
22         var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
23         file.write('CryptHashData() attach\n');
24         file.write("[+] HCRYPTHASH hHash: " + args[0] + "\n");
25         file.write("[+] const BYTE *pbData: " + args[1] + "\n");
26
27         file.write("[+] DWORD dwDataLen: " + args[2] + "\n");
28         file.write("[+] DWORD dwFlags: " + args[3] + "\n\n");
29         file.close();
30     },
31     onLeave: function (retval) {
32     }
33 });
34
35 Interceptor.attach(Module.findExportByName(null, 'CryptDeriveKey'),
36 {
37     onEnter: function (args) {
38         var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
39         file.write('CryptDeriveKey() attach\n');
40     }
41 });
42
43

```

```

44     file.write("[+] HCRYPTPROV hProv: " + args[0] + "\n");
45     file.write("[+] ALG_ID     Algid: " + args[1] + "\n");
46     file.write("[+] HCRYPTHASH hBaseData: " + args[2] + "\n");
47     file.write("[+] DWORD     dwFlags: " + args[3] + "\n");
48     file.write("[+] HCRYPTKEY   *phKey: " + args[4] + "\n\n");
49     file.close();
50 },
51
52     onLeave: function (retval) {
53     }
54 });
55
56 Interceptor.attach(Module.findExportByName(null, 'CryptDecrypt'),
57 {
58     onEnter: function (args) {
59         var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
60         file.write('CryptDecrypt() attach\n');
61         file.write("[+] HCRYPTKEY   hKey: " + args[0] + "\n");
62         file.write("[+] HCRYPTHASH hHash: " + args[1] + "\n");
63         file.write("[+] BOOL     Final: " + args[2] + "\n");
64         file.write("[+] DWORD     dwFlags: " + args[3] + "\n");
65         file.write("[+] BYTE     *pbData: " + args[4] + "\n");
66         file.write("[+] DWORD     *pdwDataLen: " + args[5] + "\n\n");
67         file.close();
68     },
69
70     onLeave: function (retval) {
71     }
72 });
73
74 Interceptor.attach(Module.findExportByName(null, 'CryptAcquireContextA'),
75 {
76     onEnter: function (args) {
77         var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
78         file.write('\n\nCryptAcquireContextA() attach\n');
79         file.write("[+] HCRYPTPROV *phProv: " + args[0] + "\n");
80         file.write("[+] LPCSTR     szContainer: " + args[1] + "\n");
81         file.write("[+] LPCSTR     szProvider: " + args[2] + "\n");
82         file.write("[+] DWORD     dwProvType: " + args[3] + "\n");
83         file.write("[+] DWORD     dwFlags: " + args[4] + "\n\n");
84         file.close();
85     },
86
87     onLeave: function (retval) {
88     }
89 });
90
91 Interceptor.attach(Module.findExportByName(null, 'CryptExportKey'),
92 {
93     onEnter: function (args) {
94         var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
95         file.write('CryptExportKey() attach\n');
96         file.write("[+] HCRYPTKEY   hKey: " + args[0] + "\n");
97         file.write("[+] HCRYPTKEY   hExpKey: " + args[1] + "\n");
98         file.write("[+] DWORD     dwBlobType: " + args[2] + "\n");
99         file.write("[+] DWORD     dwFlags: " + args[3] + "\n");

```

```

100     file.write("[+] BYTE      *pbData: " + args[4] + "\n");
101     file.write("[+] DWORD     *pdwDataLen: " + args[5] + "\n\n");
102     file.close();
103 },
104
105     onLeave: function (retval) {
106     }
107 });
108
109 Interceptor.attach(Module.findExportByName(null, 'CryptGenKey'),
110 {
111     onEnter: function (args) {
112         var file = new File("C:\\\\Users\\test\\Desktop\\frida.br", "a");
113         file.write('CryptGenKey() attach\n');
114         file.write("[+] HCRYPROV hProv: " + args[0] + "\n");
115         file.write("[+] ALG_ID     Algid: " + args[1] + "\n");
116         file.write("[+] DWORD     dwFlags: " + args[2] + "\n");
117         file.write("[+] HCRYPIKEY *phKey: " + args[3] + "\n\n");
118         file.close();
119     },
120
121     onLeave: function (retval) {
122     }
123 });
124
125 Interceptor.attach(Module.findExportByName(null, 'CryptSetKeyParam'),
126 {
127     onEnter: function (args) {
128         var file = new File("C:\\\\Users\\test\\Desktop\\frida.br", "a");
129         file.write('CryptSetKeyParam() attach\n');
130         file.write("[+] HCRYPIKEY hKey: " + args[0] + "\n");
131         file.write("[+] DWORD     dwParam: " + args[1] + "\n");
132         file.write("[+] const BYTE *pbData: " + args[2] + "\n");
133         file.write("[+] DWORD     dwFlags: " + args[3] + "\n\n");
134         file.close();
135     },
136
137     onLeave: function (retval) {
138     }
139 });
140
141 Interceptor.attach(Module.findExportByName(null, 'CryptStringToBinaryA'),
142 {
143     onEnter: function (args) {
144         var file = new File("C:\\\\Users\\test\\Desktop\\frida.br", "a");
145         file.write('CryptStringToBinaryA() attach\n');
146         file.write("[+] LPCSTR pszString: " + args[0] + "\n");
147         file.write("[+] DWORD     cchString: " + args[1] + "\n");
148         file.write("[+] DWORD     dwFlags: " + args[2] + "\n");
149         file.write("[+] BYTE     *pbBinary: " + args[3] + "\n");
150         file.write("[+] DWORD     *pcbBinary: " + args[4] + "\n");
151         file.write("[+] DWORD     *pdwSkip: " + args[5] + "\n");
152         file.write("[+] DWORD     *pdwFlags: " + args[6] + "\n\n");
153         file.close();
154     },
155

```

```

156     onLeave: function (retval) {
157     }
158 });
159
160 Interceptor.attach(Module.findExportByName(null, 'CryptGetKeyParam'),
161 {
162     onEnter: function (args) {
163         var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
164         file.write('CryptGetKeyParam() attach\n');
165         file.write("[+] HCRYPIKEY hKey: " + args[0] + "\n");
166         file.write("[+] DWORD     dwParam: " + args[1] + "\n");
167         file.write("[+] BYTE      *pbData: " + args[2] + "\n");
168         file.write("[+] DWORD     *pdwDataLen: " + args[3] + "\n");
169         file.write("[+] DWORD     dwFlags: " + args[4] + "\n\n");
170         file.close();
171     },
172
173     onLeave: function (retval) {
174     }
175 });
176 Interceptor.attach(Module.findExportByName(null, 'CryptCreateHash'),
177 {
178     onEnter: function (args) {
179         var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
180         file.write('CryptCreateHash() attach\n');
181         file.write("[+] HCRYPIPROV hProv " + args[0] + "\n");
182         file.write("[+] ALG_ID     Algid " + args[1] + "\n");
183         file.write("[+] HCRYPIKEY hKey " + args[2] + "\n");
184         file.write("[+] DWORD     dwFlags " + args[3] + "\n");
185         file.write("[+] HCRYPIHASH *phHash " + args[4] + "\n\n");
186         file.close();
187     },
188
189     onLeave: function (retval) {
190     }
191 });
192
193 Interceptor.attach(Module.findExportByName(null, 'CryptAcquireContextW'),
194 {
195     onEnter: function (args) {
196         var file = new File("C:\\Users\\test\\Desktop\\frida.br", "a");
197         file.write('CryptAcquireContextW() attach\n');
198         file.write("[+] HCRYPIPROV *phProv: " + args[0] + "\n");
199         file.write("[+] LPCSTR     szContainer: " + args[1] + "\n");
200         file.write("[+] LPCSTR     szProvider: " + args[2] + "\n");
201         file.write("[+] DWORD     dwProvType: " + args[3] + "\n");
202         file.write("[+] DWORD     dwFlags: " + args[4] + "\n\n");
203         file.close();
204     },
205
206     onLeave: function (retval) {
207     }
208 });

```

Código B.3: Fichero “instrumentation.js”

Los archivos “exe.py”, “zip.py” e “instrumentation.js” son copiados a la máquina virtual al iniciar el análisis. El fichero exe ejecuta el fichero zip, que es el encargado de lanzar Frida.

A continuación, se crea un modulo *Processing* de nombre “custom”. Los módulos *Processing* se localizan en “**CUCKOO**/processing”. En esta carpeta se crea un archivo llamado “custom.py” y tiene el Código B.4, que es el encargado de procesar los datos del análisis.

```
1 import os
2 import json
3 import io
4
5 from cuckoo.common.abstracts import Processing
6 from cuckoo.common.exceptions import CuckooProcessingError
7
8
9
10 class Custom(Processing):
11     """Analysis custom information."""
12
13     def searchFunction(self, custom_log):
14         functions = ["CryptEncrypt()", "CryptHashData()", "CryptDeriveKey()", "
15                     CryptExportKey()", "CryptGenRandom()", "CryptDecrypt()", "
16                     CryptAcquireContextA()", "CryptAcquireContextW()", "CryptGenKey()", "
17                     CryptSetKeyParam()", "CryptStringToBinaryA()", "CryptGetKeyParam()"]
18         json = "Functions":[
19
20             for function in functions:
21
22                 with io.open(custom_log, 'r', encoding='utf-8') as ObjFichero:
23
24                     for line in ObjFichero:
25
26                         PosicionTexto = line.find(function)
27
28                         if PosicionTexto >= 0:
29
30                             json += '"' + function + ','
31
32                             break
33                         else:
34                             json += ""
35
36         json = json[:-1]
37         json += '], '
38
39         return json
40
41     def searchProviderType(self, custom_log):
42         providers = {"0x1" : "PROV_RSA_FULL", "0x18" : "PROV_RSA_AES", "0x2" : "
43                     PROV_RSA_SIG", "0xc" : "PROV_RSA_SCHANNEL", "0x3" : "PROV_DSS", "0xd" : "
```

```

41         PROV_DSS_DH", "0x12" : "PROV_DH_SCHANNEL", "0x4" : "PROV_FORTEZZA", "0x5" :
42         "PROV_MS_EXCHANGE", "0x6" : "PROV_SSL"}
43     json = ""
44
45     with io.open(custom_log, 'r', encoding='utf-8') as ObjFichero:
46
47         for line in ObjFichero:
48
49             PosicionTexto = line.find("dwProvType")
50
51             if PosicionTexto >= 0:
52                 print line[27:-1]
53
54                 json = '"ProviderType": '
55                 json += '"' + providers[line[27:-1]] + ','
56
57                 break
58             else:
59                 json = ""
60
61         return json
62
63     def searchAlgorithm(self, custom_log):
64         algorithms = {"0x6603": "CALG_3DES", "0x6609": "CALG_3DES_112", "0x6611": "
65         CALG_AES", "0x660e": "CALG_AES_128", "0x660f": "CALG_AES_192", "0x6610": "
66         CALG_AES_256", "0xaa03": "CALG_AGREEDKEY_ANY", "0x660c": "CALG_CYLINK_MEK",
67         "0x6601": "CALG_DES", "0x6604": "CALG_DESX", "0xaa02": "CALG_DH_EPHEM", "0
68         xaa01": "CALG_DH_SF", "0x2200": "CALG_DSS_SIGN", "0xaa05": "CALG_ECDH", "0
69         xae06": "CALG_ECDH_EPHEM", "0x2203": "CALG_ECDSA", "0xa001": "CALG_ECMQV", "
70         0x800b": "CALG_HASH_REPLACE_OWF", "0xa003": "CALG_HUGHES_MD5", "0x8009": "
71         CALG_HMAC", "0xaa04": "CALG_KEYA_KEYX", "0x8005": "CALG_MAC", "0x8001": "
72         CALG_MD2", "0x8002": "CALG_MD4", "0x8003": "CALG_MD5", "0x2000": "
73         CALG_NO_SIGN", "0xffffffff": "CALG_OID_INFO_CNG_ONLY", "0xffffffffe": "
74         CALG_OID_INFO_PARAMETERS", "0x4c04": "CALG_PCT1_MASTER", "0x6602": "
75         CALG_RC2", "0x6801": "CALG_RC4", "0x660d": "CALG_RC5", "0xa400": "
76         CALG_RSA_KEYX", "0x2400": "CALG_RSA_SIGN"}
77     json = ""
78
79     with io.open(custom_log, 'r', encoding='utf-8') as ObjFichero:
80
81         for line in ObjFichero:
82
83             PosicionTexto = line.find("ALG_ID")
84
85             if PosicionTexto >= 0:
86                 json = '"Algorithm": '
87                 json += '"' + algorithm[line[22:-1]] + ','
88
89                 break
90             else:
91                 json = ""
92
93         return json
94
95     def searchName(self, custom_log):
96         json = ""
97
98         with io.open(custom_log, 'r', encoding='utf-8') as ObjFichero:

```

```

83
84     for line in ObjFichero:
85         PosicionTexto = line.find("Name")
86
87         if PosicionTexto >= 0:
88             var = line[39:-5].split("-")
89             json = '"Name":' + var[0] + ','
90             json += '"Hash":' + var[1] + ','
91             json += '"Date":' + var[2] + ','
92         return json
93
94 def fullLog(self , custom_log):
95     json = '"zFull_Log":'
96     with io.open(custom_log, 'r', encoding='utf-8') as ObjFichero:
97         for line in ObjFichero:
98             PosicionTexto = line.find("Name")
99
100            if PosicionTexto >= 0:
101                json += " "
102            else:
103                json += line[:-1] + " \n"
104    json += "' "
105    return json
106
107 def createJson(self , custom_log):
108
109     name = self.searchName(custom_log)
110     functions = self.searchFunction(custom_log)
111     algorithm = self.searchAlgoritmo(custom_log)
112     provider = self.searchProviderType(custom_log)
113     fullLog = self.fullLog(custom_log)
114     return "{" + name + algorithm + provider + functions + fullLog + "}\n"
115
116 def escribirJson(self , custom_log):
117
118     f = open ("/home/nacho/Documents/Projects/Frida/archives/samples.json", "a")
119     f.write(self.createJson(custom_log))
120     f.close()
121
122 def run(self):
123
124     self.key = "custom"
125     try:
126         custom_log = os.path.join(self.dropped_path, "frida.log")
127         self.escribirJson(custom_log)
128
129         with io.open(custom_log, 'r', encoding='utf-8') as report_file:
130             data = report_file.read()
131     except Exception, e:
132         raise CuckooProcessingError(str(e))
133     return data

```

Código B.4: Fichero “custom.py” del módulo *Processing*

Posteriormente se habilita el nuevo módulo *Processing* agregando la sección del Código B.5 en “CWD/conf/processing.conf”.

```
1 [custom]
2 enabled = yes
```

Código B.5: Sección custom en el fichero "processing.conf"

Y por último se agrega el campo del Código B.6 al diccionario definido en “CUCKOO/common/config.py”.

```
1 "processing": {
2     "analysisinfo": {
3         "enabled": Boolean(True),
4     },
5     "apkinfo": {
6         "enabled": Boolean(False),
7         "decompilation_threshold": Int(5000000),
8     },
9     "baseline": {
10        "enabled": Boolean(False),
11    },
12    "behavior": {
13        "enabled": Boolean(True),
14    },
15    "custom": {
16        "enabled": Boolean(True),
17    },
18    ...
19 },
```

Código B.6: Campo "custom" del módulo *Processing* en el fichero "config.py"

Una vez creado el módulo *Processing*, se crea el modulo *Reporting*, denominado “custom”. Los módulos de *Reporting* están localizados en “CUCKOO/reporting”, donde se crear el archivo “custom.py” con el Código B.7, que es el encargado de mostrar el resultado del análisis en la consola web de Cuckoo.

```
1 import os
2 import json
3 import codecs
4
5 from cuckoo.common.abstracts import Report
6 from cuckoo.common.exceptions import CuckooReportError
7
8 class Custom(Report):
9
10     def run(self, results):
11
12         try:
13
14             with codecs.open(path, "w", "utf-8") as report:
15                 json.dump(results["custom"], report, sort_keys=False, indent=4)
```

```

16     except (UnicodeError, TypeError, IOError) as e:
17         raise CuckooReportError("Failed to generate JSON report: %s" % e)

```

Código B.7: Fichero “custom.py” del módulo *Reporting*

Después se habilita el módulo de *Reporting* agregando la sección del Código B.8 en “**CWD**/conf/reporting.conf”.

```

1 [custom]
2 enabled = yes

```

Código B.8: Sección custom en el fichero “reproting.conf”

Por último, se agrega el campo del Código B.9 al diccionario definido en “**CUC-KOO**/common/config.py”.

```

1     "reporting": {
2         "feedback": {
3             "enabled": Boolean(False),
4         },
5         "jsondump": {
6             "enabled": Boolean(True),
7             "indent": Int(4),
8             "calls": Boolean(True),
9         },
10        "custom": {
11            "enabled": Boolean(True),
12        },
13        ...
14    },

```

Código B.9: Campo “custom” del módulo *Reporting* en el fichero “config.py”

Una vez integrado Frida con Cuckoo, se integra ELK cambiando la configuración de Logstash (véase Código B.10) que se localiza en “/etc/logstash/config.d/frida.conf” para coger los resultados del análisis y enviarlos a Elasticsearch.

```

1 input {
2     file {
3         type => "json"
4         path => "/home/nacho/Documentos/Projects/Frida/archives/samples.json"
5         start_position => "beginning"
6         sincedb_path => "/dev/null"
7     }
8 }
9
10 filter {
11     json {
12         source => "message"
13     }
14 }
15
16

```

```
17 | output {
18 |   elasticsearch {
19 |     hosts => ["localhost:9200"]
20 |     index => "test"
21 |   }
22 |   stdout { codec => rubydebug }
23 | }
```

Código B.10: Fichero “frida.conf”

Anexo C

Muestras de ransomware analizadas

Tabla C.1: Muestras de ransomware utilizadas para el análisis, ordenadas alfabéticamente

Nombre	Hash	Año
7zipper	7b009c2b16bbb35326d64b2d905d61cad-f09989f6d60cfdaf256d58ab400a639	2017
ABCLocker	e0e7ea7f9c0cd61b7211459b2d9b51d-558e4bb35ee63ce2f43fe9a6c3a96bb54	2017
Apocalypse	478383fb588665c254d416b7c50a12-4f82291124b002d9bad9fd758a59fd728f	2016
ApolloLocker	e9311c94648d1b96052eabd5f6cb-30ba8338c968f303c67ee173ee457f0dcaf8	2017
Aurora	fdb9e49213b880cf9a570a1279e36f6fef-80901e4367f73b674144f7fb4946dc	2018
AVcrypt	58c7c883785ad27434ca8c9fc20b02885-c9c24e884d7f6f1c0cc2908a3e111f2	2018
BadBlock	f58a3cca57107313fbb2eff231ebe52d-74feabf0097e53343efd2a1f9ed97775	2016
BitCrypt	3c95f5b3e07449bb05c2cacafbb12ca6-9cecf2496328b1c700ec1073ad52e99a	2014
Bitpaymer	43984eb5b8f35d5e89cefb755a679b-78824dd81a2ecf27829b56ff11cb293cc	2017
Bitshifter	3e63a8e13c2363bb3002c04152cbbc-908f237b9ca487b3f0681a09e8aa9451fd	2017

Blackout	10010b61af8d21c6c43f9d7fb99795a0-77224780af70450f1d6e30030e176d1a	2017
BTCWare	880d25776e08769a75c43bf9a69f9f7ca-fcc46546690270fa36785195f327d97	2017
CBTLocker	8cde925b91846358d7d0976b0ee0961-3fd8911622979cbb8cc471be430e79950	2014
Cerber	408fd7edadfdbdaab161e04afcfc115c464-916e99aaba8b036f52c57c3ade49c5	2016
Cerber	c6f29582e489506ccb14f19fdfa7c169b3-63246a44b760484716e7a3e15b0fb9	2016
Chimera	1dacdc296fd6ef6ba817b184cce990190-1c47c01d849adfa4222bfabfed61838	2015
Cryptgh0st	f763a8115f7be167ecb4825c6f26a2-5b1f62c32f3ae5259f54b58928f9be89da	2018
CryptoGod	85b0e492fb1019f3696714a33a62a5f-1692af379a98c87debc8a50c47950160c	2018
CryptoHasYou	7d66e29649a09bf3edb61618a61f-d7f9fb74013b739dfc4921eefece6c8439bb	2016
CryptoLocker	a2bc3059283d7cc7bc574ce32cb6-b8bfd27e02ac3810a21bd3a9b84c17f18a72	2013
Cryptomix	004cdc6996225f244aef124edc72f90-434a872b3d4fa56d5ebc2655473733aef	2016
Dcrtr	333ab4a2d64a023ce61d0c2f4ce109b3c08-f8e50b34156b7f7613ac575a97cd0	2018
DeathNote	50dca038c2306d0d7cc9833216461f9-79be25421cb67c7b033c30e33ba4b432a	2018
Dharma	c2ab289cbd2573572c39cac3f234d77fdf-769e48a1715a14feddaea8ae9d9702	2016
DontWorry	9a522b3b5a51b5e021977ad55780f52-7d58193d587b1c01588e07cc40a69f884	2018
Donut	2f40011df85d75556816ac944d805b6313d-a44c73c80778af62be5727c005811	2018
EggLocker	a7c1e2ab0ab45f07a1ccceca5d48939-727e8116d8a5de57af5883c8a2650115a	2018
EOEO	0cf5c4b3399d094fa0c58399fac521e4b290-2db7ae1692aa502f5784ac755d49	2018

EverbeV2	18dfcf81046272e08f6ef3230df83008- cb78eb30cda341c59ceb33c5be542d85	2018
FLKR	dad59e4617ab8e4adcf5a1c5b749b4792d0- 3894b3ed3b9018f388dc1f212146	2016
GandCrab	5d53050a1509bcc9d97552fa52c1105b- 51967f4ccf2bde717b502605db1b5011	2018
GandCrabV4	ef7b107c93e6d605a618fee82d5aeb- 2b32e3265999f332f624920911aabe1f23	2018
GlobeImposter	3a9d5976fbf41daf80f0eb9e6b7- ce52a82fe9609984ef7f8ea166048547	2017
gpcode	628f3906ccfa4ede0ac6466bdef8e8a79f- 53ae9e26b165ab2f0c1569c5ede2c9	2005
GPGQwerty	f5cd435ea9a1c9b7ec374ccbd08cc6c- 4ea866bc438ea8f1523251966c6e88b	2018
Halloware	007c1f11afb195d77c176891d54b9cf- d37c87b13dfe0ab5b6c368125e4459b8c	2017
Jaff	0746594fc3e49975d3d94bac8e80c0cdaa96- d90ede3b271e6f372f55b20bac2f	2017
Jigsaw	80bc27ef05d51164ba3a66e6353a84e9c9- 1de6f9961a15c99afe47be7a16e10b	2016
JobCrypter	052c942192cd20241fefa76f2942ca- 2959b151b8eda0e510b9c4b1abb991e126	2016
Katyusha	d00ee0e6eab686424f8d383e151d2200- 5f19adbda5b380a75669629e32fe12a6	2018
KeyPass	ee74c63faa2eb9709b1d738762e28072a- ece2e7b9effc5913eb6a5fd1564752	2018
KillRabbit	92c50cd253de42823a2e1a59f2551a- a315ceb12b8f741820bdbc14b5ebe1dfb9	2018
Kraken	528e4d24c18160b6bdd73c9a612d38a78f- c58bd40c8ab415973a94429b321dfc	2016
LightningCrypt	0a3697a4a3476b61b54a039875- 2d1daffb4da24c6cd25bdcebfaaac05706d288	2017
LittleFinger	6243ddb5f1337118e0a5cb17c326- d30d6b90237c316463312e93c45cbe713346	2018
Locker	c8b31a92d2ab8bb163cfb66b7d461acd79- 41c5c17314220ecdee6abdcd005a8e	2015

Locky	17c3d74e3c0645edb4b5145335b342d2929-c92dff856cca1a5e79fa5d935fec2	2016
LongTermMemoryLoss	25c284b66ff946307eea7a-ce2c9abef767cb9a3aaaa3d2e69d11db041ad3727d	2018
Magniber	2e6f9a48d854add9f895a3737fa5fcc9-d38d082466765e550cca2dc47a10618e	2017
Matrix	522e7328b76109502231e9f80a2c83fb0c-57c28db98b54e0e7f401e368401046	2016
Meteoritan	576cb374221b4ede43c70d43b74f34-5008b4fac9e21657b6ef4d3ddca35a8906	2017
Nemucod	9f6519bade9ec587d38bd21c74b04b9c5-e4b3e4cbc8651db806c78720afc42dd	2016
NotAHero	e4e9ae0de9f95d6e5334e6bb36c832db-cd172adfe521a7b04f889af2eda59880	2017
Paradise	2d20b610c64a5b400478fdff089178e4-67a6c3e70674c94351af777a706b7488	2017
Petya	26b4699a7b9eeb16e76305d843d4ab05e94-d43f3201436927e13b3ebafa90739	2016
PGPSnippet	4a75763b0d8493b3d66aa9b2c57ffb-240a020036d07aef2a6d985f0d024af5c	2018
PoisonFang	60ec60033665f4da6fd0d9bd0fef4-4897c68a94bc3f6c97cfa08fa6ddf7eed3	2018
PopCornTime	fd370e998215667c31ae1ac6ee812-23732d7c7e7f44dc9523f2517adffa58d51	2016
PrincessLockerEvolution	1408a24b74949922cc65164eea0780449c-2d02bb6123fd992b2397f1873afd21	2018
PUBG	74c1538e894dddc5af711374795fc78594f8-c3de9ad056fd3cf705143f56f290	2018
RansomWarrior	825a42a32624644933cf5513064-ce3e2995d5f0dde1e524c5c8950d240f2945a	2018
Rapid	1db83903514d064201d54bf2fbb2dab9a37-3c3c399be45be1ecaf4559c6ed16b	2018
RaRansomware	58e363a5cab119d5adc2d0ba7e0f-ba85155871c95a8894c8449f615a8722c954	2018
Reyptson	9481f211d34d562303313173fb1ac829-a286c0a75de708f17ec3a6453cbc9480	2017

Ryuk	8d3f68b16f0710f858d8c1d2c699260e6f43-161a5510abb0e7ba567bd72c965b	2018
Satana	683a09da219918258c58a7f61f7dc4161a-3a7a377cf82a31b840baabfb9a4a96	2016
Saturn	9e87f069de22ceac029a4ac56e6305d2df-54227e6b0f0b3ecad52a01fbade021	2018
ScarabBomber	5062d41b0b91014042c8df5bfbeb-451d8a8584649e9c48cf8a53b1c5c8c49a8b	2018
ShinoLocker	2516f9eedf86130926a8042e7faf3-aa739eaa70248fbc7a3d5439b1eb9af8f20	2016
Shrug	c89833833885bafdcfa1c6ee84d7dbcf238-9b85d7282a6d5747da22138bd5c59	2018
Sigma	b81c7079fd573304bb8fb177898dfbf6acd-b16ff32632dfa9ebb9c3da2a59864	2017
Sigrun	664b482e22e0f108660cf03fb7d1507d92-9e8242eb6c5762e577096a50a8cc5b	2018
SilentSpring	a0fa26c2f79fd4e0f3f25b3fcb7a-a02bb0194b34f13aa8d721b114432a00b05a	2018
smrss32	17d2c0d0fbc3c37cb467930d29ad93598-8edb1d0c727778ef31a4f9f91998e25	2016
Spartacus	ef25bdbcf05fa478df3ddc5f4f717c0-70e443da04cfc590d44409c815f237cb3	2018
Spora	5ab9b586eaf1bcaa76443b4f69d67e57a05-7d57cb30b6d863a7cfab3d0882c2a	2017
STOP	5cd8748313f911a2f13cf555b21c022433fa-895ec968c530924386714e1604ac	2017
SYSDOWN	280f1c9db2325a7f66e76eaf10bb4bfe9-2bd90e96ac01525f9ee86a902382fd1	2018
Termite	021ca4692d3a721af510f294326a31780-d6f8fcd9be2046d1c2a0902a7d58133	2018
TeslaCrypt	650e8daef2695675d65655764d18f4-765aa2d134ba32bc397a3961318d5a75e4	2015
TeslaCrypt	ca7cb56b9a254748e983929953df32-f219905f96486d91390e8d5d641dc9916d	2015
Thanatos	97d4145285c80d757229228d13897820-d0dc79ab7aa3624f40310098c167ae7e	2017

TorrentLocker	c0b6953bf1306d0697b716f1999- eb6da518924132e041a050e9e5eb2f1d545a8	2014
TotalWipeOut	54ef5dd5a99a13b476f3673a0bce- 5219186a06d5d1a8c1769827267c421b6b65	2018
TRON	071af25cf4b989176f1f61d7636f5eb5004e- 13594ccb8f8fc659e289ef4ae0191	2018
Unlock92	53e48f638cfab80fc5217d0833a697ea- b133071630058dea754502bfe00ab26c	2018
UselessFiles	d6e44edef14d894b837019be4235- b6d0d61e2526cf6e088f6955c8fd0c7d4230	2018
Velso	d34077e03b35a77db9bca1f7fc285993edf- 98e03a4d6481169bcc18d17f8c0f8	2018
VenusLocker	d31afd6d582a666e121a1e1df8bc1- 5a93a8793c46c2814730da2e56fdce3ba4a	2016
Virlock	efdcd96ef514481ce200b9853eb9f33be- b21b2454874a2504e2b99373a563883	2014
Vurten	583aabffbdb69f611557f8289059792e4f- f0aeb7ce6d7dc812dbd3b93079b1c9	2018
Wannacry	ed01ebfbc9eb5bbea545af4d01bf5f10- 71661840480439c6e5babe8e080e41aa	2017
WannaSmile	6e354e4ea3df8d5d903a367748b0ad- e52a3310664b85175aaec86f5b9d88068d	2017
WhiteRose	9614b9bc6cb2d06d261f97ba25743a8- 9df44906e750c52398b5dbdbcb66a9415	2018
Wooly	39564d97064f438d8682d41403a321dcfc9- 1f1d28ac83de3060adf3d380bee05	2017
XiaoBa	ec663b43bce75ac8950ff33cd929038678- 8d1cdbe889710ee68b3512920ff5ce	2017
Xorist	b601f2907210f5c30dfd3b123ac98f2146- 8bcecbffd4f6506b0c60e754b4f835	2016
YYTO	79946df6b53fbc8346b956cfda3448c97f8- 81e48cd325583b6497fca8a0a3e6	2017
Zoldon	515696aaf6ea74dd0f4038b24336f636a9- 25eacb7b770aa469bda22ba9d4d13b	2018