

**Escola Tècnica Superior d’Enginyeria**

**Electrònica i Informàtica La Salle**

Trabajo Final de Máster

Máster en Ciberseguridad

*Malware en javascript*

Alumno Profesor Ponente

Miguel Soriano Sanz Pedro Candel Rodríguez

|  |
| --- |
| **ACTA DEL EXAMEN**  **DEL TRABAJO FINAL DE MÁSTER** |

Reunido el Tribunal calificador en el día de la fecha, el alumno

D. Miguel Soriano Sanz

Expuso su Trabajo Final de Máster, el cual trató sobre el tema siguiente:

**Malware en javascript**

Acabada la exposición y contestadas por parte del alumno las objeciones formuladas por los Sres. miembros del tribunal, éste valoró dicho Trabajo con la calificación de

|  |
| --- |
|  |

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENTE DEL TRIBUNAL

# Resumen

La distribución de *malware*[[1]](#footnote-1) y sobre todo de *ransomware*[[2]](#footnote-2) es cada vez más masiva y sofisticada siendo los *droppers*[[3]](#footnote-3) en JavaScript uno de los vectores de ataque más utilizados hoy en día. Por ello es necesario tener métodos automáticos de análisis que nos faciliten la clasificación y entender el comportamiento del *malware* para obtener indicadores de compromiso (IOCs)[[4]](#footnote-4) que nos permitan protegernos ante estos ataques.

Malware-Jail es una *sandbox*[[5]](#footnote-5) desarrollada en Node.js que permite simular la ejecución del *malware* en JavaScript y así extraer el funcionamiento del programa y el *payload*[[6]](#footnote-6) final. No obstante Malware-Jail aún sigue en desarrollo y no es capaz de simular la ejecución de las muestras de *malware* más actuales. Se pretende analizar aquellas muestras que Malware-Jail no sea capaz de ejecutar e implementar las funciones pertinentes para actualizar la *sandbox*.

Hablar de cuckoo o FAME como frontEnd

# Índice

[Resumen 3](#_Toc490041509)

[Índice 4](#_Toc490041510)

[1. Introducción 1](#_Toc490041511)

[2. Objetivos 2](#_Toc490041512)

[3. Fundamentos teóricos 3](#_Toc490041513)

[3.1. Funcionamiento del *malware* 3](#_Toc490041514)

[*3.2.* Ofuscación en el *malware* 3](#_Toc490041515)

[3.3. Herramientas 3](#_Toc490041516)

[4. Analizando *malware* a mano 4](#_Toc490041517)

[4.1. Diferencias entre muestras del mismo *malware* 4](#_Toc490041518)

[4.2. Reversing paso a paso 4](#_Toc490041519)

[5. Usando Malware-Jail 5](#_Toc490041520)

[5.1. Análisis de muestras 5](#_Toc490041521)

[5.2. Ampliando Malware-Jail 5](#_Toc490041522)

[6. Visualización de datos con 6](#_Toc490041523)

[6.1. Desarrollo del script 6](#_Toc490041524)

[6.2. Resultado final 6](#_Toc490041525)

[7. Resultados finales 7](#_Toc490041526)

[8. Coste del proyecto 8](#_Toc490041527)

[8.1. Coste temporal 8](#_Toc490041528)

[8.2. Coste económico 8](#_Toc490041529)

[9. Conclusiones 9](#_Toc490041530)

[10. Líneas de futuro 10](#_Toc490041531)

[11. Bibliografía 11](#_Toc490041532)

[Índice de figuras 12](#_Toc490041533)

[Índice de tablas 13](#_Toc490041534)

# Introducción

Cada día vemos más y más casos de *malware* que han afectado a multitud de empresas en diferentes países. Para conseguir tal número de infectados el *malware* tiene que evolucionar en nuevas formas de distribución e infección, siendo una de las más usadas en Windows los *dropper* escritos en JavaScript. Para analizar tantas muestras de *malware* al día es necesario poder automatizar ciertas tareas como la extracción del *payload* y comportamiento del *malware*. No obstante, cada vez más, el *malware* moderno se encuentra ofuscado[[7]](#footnote-7) dificultando la tarea del analista y usa funciones cada vez más avanzadas impidiendo el uso de estas herramientas automáticas. Por ello es necesario ir adaptando estas herramientas a medida que el *malware* evoluciona de tal manera que sean capaces de resolver el comportamiento incluso de las muestras más actuales.

Una de estas herramientas es Malware-Jail. Se trata de una *sandbox* desarrollada en Node.js que simula la ejecución del código en JavaScript y por lo tanto es capaz de mostrar el comportamiento de un programa y su flujo de ejecución. Gracias a esto podemos saber cuáles son las llamadas que hace un *malware* ofuscado e incluso obtener el *payload* final.

Para llevar a cabo el análisis, Malware-Jail simula un entorno de ejecución de JavaScript, para ello necesita tener definidas aquellas funciones y manejadores de clases que intentará usar el *malware*. Por lo tanto ante las nuevas técnicas empleadas por el *malware* es necesario usar técnicas de *reversing*[[8]](#footnote-8) a las muestras para ver qué funciones usan y añadirlas al conjunto que posee Malware-Jail para así poder automatizar el análisis del resto de muestras que tengan un comportamiento similar.

A lo largo de este trabajo veremos diferentes muestras de *malware* en JavaScript reales, estudiaremos sus métodos de ofuscación y desarrollaremos diferentes funciones cuando sea necesario para permitir a Malware-Jail automatizar el análisis de las muestras.

# Objetivos

Los objetivos principales sobre los que se justifica la elaboración de este proyecto son los que se detallan a continuación:

1. Investigación y análisis de diverso *malware* que sea distribuido mediante ficheros en JavaScript.
2. Clasificarlo para obtener las funciones no implementadas en malware-jail y realizar su desarrollo en Node.js
3. Integrar malware-jail en un paquete específico de análisis para cukcoo 2.0 desarrollando el módulo de procesado en Python 2.7 y creando el modelo de visualización de datos.

# Fundamentos teóricos

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

## Funcionamiento del *malware*

## Ofuscación en el *malware*

#### Motivos

#### Tipos de ofuscación

## Herramientas

#### Malware-Jail

#### Cuckoo o FAME

# Analizando *malware* a mano

## Diferencias entre muestras del mismo *malware*

## Reversing paso a paso

# Usando Malware-Jail

## Análisis de muestras

## Ampliando Malware-Jail

# Visualización de datos con

## Desarrollo del script

## Resultado final

# Resultados finales

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

…

# Coste del proyecto

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an.

## Coste temporal

El volumen de trabajo que ha comportado la realización del proyecto se puede clasificar en los siguientes apartados:

* …
* …
* …

A continuación explicamos detalladamente en qué ha consistido y cuánto tiempo ha requerido cada uno de los apartados mencionados anteriormente:

…

Figura . Coste temporal

## Coste económico

El coste económico necesario para el desarrollo del proyecto …

# Conclusiones

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

# Líneas de futuro­

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per:

* …
* …
* …

# Bibliografía

1. INSTITUTE FOR SECURITY AND OPEN METHODOLOGIES (2010). *ISECOM* [en línea]. Disponible en Internet: <http://www.isecom.org/>
2. THE PHP DOCUMENTATION GROUP (2010, 24 de Septiembre). *PHP Manual*  [en línea]: Philip Olson. Disponible en Internet:

<http://www.php.net/manual/en/>

…

…

# Índice de figuras

[Figura 1. Ejemplo de figura con autonumeración … 1](#_Toc421701638)

[Figura 2. Salida comando “ls –l” 4](#_Toc421701639)

[Figura 3. Coste temporal 7](#_Toc421701640)

# Índice de tablas

[Tabla 1. Ejemplo de tabla autonumerada 3](#_Toc421701641)

1. Malware – También llamado badware o código malicioso se refiere a aquel software cuya finalidad es causar daño a la víctima o beneficios al atacante de una manera ilícita. [↑](#footnote-ref-1)
2. Ransomware – Tipo de malware que por lo general cifra los archivos de un dispositivo privando a su dueño de acceder a ellos y pide un pago por la clave de descifrado. [↑](#footnote-ref-2)
3. Dropper – Software o programa que por él mismo no es malicioso pero que se encarga de instalar y a veces también descargar el malware real. Se usa para la evasión de antivirus y facilitar la distribución y actualización del malware. [↑](#footnote-ref-3)
4. Indicadores de compromiso – Abreviados como IOCs recogen el comportamiento que define a un malware, ya sean nombres de archivos, cambios en el OS o servidores concretos contactados con la finalidad de identificar un malware específico. [↑](#footnote-ref-4)
5. Sandbox – Entorno controlado con diferentes sensores donde se ejecuta un malware a propósito para ver su funcionamiento y poder analizarlo. [↑](#footnote-ref-5)
6. Payload – En seguridad informática el payload sería la carga o fragmento final a ejecutar por el malware si excluimos la parte del programa que se dedica a la distribución y permanencia del malware o a la explotación del dispositivo. [↑](#footnote-ref-6)
7. Ofuscar - Ofuscar un código es el acto de hacerlo ilegible para el ser humano o extremadamente difícil de entender y por lo tanto de ver su comportamiento pero siempre manteniendo la capacidad del código de ser ejecutado por un dispositivo. [↑](#footnote-ref-7)
8. Reversing - La ingeniería inversa es una modalidad por la cual partiendo de un ejecutable final se intenta averiguar la lógica del programa y cómo está construido. El nombre proviene de que en vez de escribir código para obtener un programa, se analiza el programa para averiguar el código. [↑](#footnote-ref-8)