Universidad Zaragoza
1542

Trabajo Fin de Grado

Grado en Ingeniería Informática

Caracterización de mecanismos de interacción humana en malware y desarrollo de módulos de simulación para CAPEv2

Characterization of human interaction mechanisms (Reverse Turing Tests) in malware and development of simulation modules for CAPEv2

Autor

Pablo Nicolás Fabra Roque

Directores

Ricardo Julio Rodríguez Fernández

Javier Carrillo Mondéjar

ESCUELA DE INGENIERÍA Y ARQUITECTURA
SEPTIEMBRE 2026

AGRADECIMIENTOS

La finalización de este Trabajo Fin de Grado es el punto final de cuatro años de esfuerzo, y no habría sido posible sin quienes me han acompañado durante todo ese camino.

Gracias a mis padres, por su apoyo constante y por haber estado presentes siempre que lo he necesitado. Todo lo que he conseguido tiene detrás la educación y los valores que ellos me han dado. Si soy la persona que soy hoy, es gracias a ellos.

Gracias a mis hermanos, por estar siempre ahí de forma incondicional, sin necesidad de pedirlo. Son de esas presencias que uno da por sentadas hasta que se da cuenta de lo mucho que sostienen.

Gracias también a mis compañeros de universidad, con los que he compartido estos cuatro años. Las horas de prácticas y los agobios de época de exámenes se llevan de otra manera cuando se comparten. Buena parte de lo que he aprendido durante la carrera lo he aprendido con ellos.

Por último, quiero dar las gracias a mis directores, Ricardo y Javier. Sin su orientación este trabajo no habría llegado tan lejos.

DECLARACIÓN DE USO RESPONSABLE DE IA

Durante el desarrollo de este trabajo se ha recurrido a Microsoft Copilot, Claude Sonnet 4.6 y Claude Opus 4.8 como herramientas de apoyo para la programación de algunos scripts de automatización de procesos, en concreto los empleados en el estudio de prevalencia, y para la depuración y detección de errores. El autor ha revisado, adaptado y verificado todo el material generado, haciéndose responsable del contenido íntegro de este trabajo, incluidas las partes en las que se usó esta ayuda.

RESUMEN

El análisis dinámico en entornos aislados constituye la técnica de referencia para el estudio de *malware*. Su eficacia se ve comprometida, sin embargo, por los mecanismos de comprobación de presencia humana, conocidos como *Reverse Turing Tests*. Mediante estos mecanismos se condiciona la ejecución de la carga maliciosa a la detección de un usuario real, de modo que en un entorno automatizado la muestra permanece inerte y el informe generado no recoge actividad alguna.

En el presente Trabajo de Fin de Grado se aborda este problema desde dos vertientes complementarias. En la primera se caracteriza el fenómeno y se mide su prevalencia. Para ello, se desarrolló un conjunto de siete reglas con las que se detectan estos mecanismos de forma estática, aplicadas sobre un corpus de 653 muestras confirmadas de *malware* y contrastadas frente a un grupo de control de 700 binarios legítimos. Sobre las muestras en las que el análisis por importaciones resulta aplicable, el 53,7 % del *malware* implementa algún mecanismo de este tipo, frente al 10,1 % del software legítimo comparable.

En la segunda vertiente se desarrolló un módulo de simulación de interacción para el entorno de análisis CAPEv2. En él se reproducen las propiedades del movimiento humano, se generan clics como eventos físicos de entrada y se atienden los cuadros de diálogo mediante dos técnicas de confirmación encadenadas. El módulo se validó sobre cuatro casos de estudio con mecanismos de naturaleza distinta, en los cuales se logró activar el comportamiento malicioso. Frente al módulo incorporado por defecto en la herramienta, con el realizado se mejoran y superan (pasando de 3 a 7 detecciones exitosas) las comprobaciones de interacción de una batería de referencia.

ABSTRACT

Dynamic analysis in isolated environments is the reference technique for malware research. Its effectiveness is nevertheless undermined by human presence checks, commonly referred to as reverse Turing tests. Through these mechanisms, the execution of the malicious payload is made conditional on the detection of a real user, so that in an automated environment the sample remains inert and the resulting report shows no activity at all.

This Bachelor's Thesis addresses the problem from two complementary angles. In the first one, the phenomenon is characterized and its prevalence is measured. To this end, a set of seven rules was developed for the static detection of these mechanisms. The rules were applied to a corpus of 653 confirmed malware samples and contrasted against a control group of 700 legitimate binaries. Among the samples for which import-based analysis is applicable, 53.7 % of the malware implements at least one such mechanism, compared to 10.1 % of comparable legitimate software.

In the second angle, an interaction simulation module was developed for the CAPEv2 analysis environment. The module reproduces the properties of human movement, generates clicks as physical input events and handles dialog boxes by chaining two different confirmation techniques. It was validated against four case studies covering mechanisms of different nature, and the malicious behaviour was successfully triggered in all of them. Compared to the module included by default in the tool, the one developed here passes all seven interaction checks of a reference test suite, instead of three.

Índice

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El análisis dinámico de *malware* se apoya en una premisa sencilla, y es que basta con ejecutar una muestra en un entorno controlado para observar lo que hace. Buena parte del *malware* actual ha convertido esa premisa en su punto débil. En lugar de ocultar su comportamiento malicioso, sencillamente no lo ejecuta mientras sospecha que está siendo observado [1]. En este trabajo se aborda una de las formas más eficaces de comprobarlo, que consiste en detectar si hay una persona al otro lado de la pantalla.

1.1. Contexto y motivación del trabajo

El análisis de *malware* se aborda tradicionalmente desde dos enfoques complementarios: análisis estático y análisis dinámico [2]. En el análisis estático se examina el binario sin ejecutarlo, mediante la inspección de su código, sus cadenas de texto y su tabla de importaciones. En el análisis dinámico, en cambio, se ejecuta la muestra en un entorno aislado y se registra su comportamiento real, que incluye las llamadas a la *Application Programming Interface* (API) del sistema operativo, la actividad de red y las modificaciones sobre ficheros y registro.

El análisis dinámico se ha consolidado como la técnica de referencia porque resiste bien las contramedidas que inutilizan al análisis estático [3]. Un binario empaquetado, cifrado o que resuelve sus funciones en tiempo de ejecución revela poco al inspeccionarlo mediante análisis estático, pero muestra su comportamiento real al ejecutarlo. Los entornos aislados y vigilados (denominados *sandboxes*) automatizados, como CAPEv2, son la implementación más extendida de este enfoque y con ellas se analizan grandes volúmenes de muestras sin intervención manual [4].

Este éxito ha provocado una reacción esperable. La evasión de *sandboxes* se ha convertido en una capacidad habitual del *malware* moderno, hasta el punto de que existen catálogos sistemáticos de las técnicas empleadas [5]. En consecuencia, la eficacia del análisis dinámico depende de que el entorno no sea reconocido como tal.

Con las técnicas dependientes de detección se buscan activamente indicios que revelen el entorno de análisis. La más extendida es el *fingerprinting*, en el que se rastrean artefactos como claves de registro propias de VMware o VirtualBox, cantidades de memoria o de núcleos poco verosímiles, o la presencia de herramientas de monitorización [1]. Frente a este tipo de comprobaciones cabe una respuesta reactiva, que consiste en eliminar o falsear cada artefacto conocido. El resultado es una escalada continua, pero al menos se dispone de un procedimiento claro de mitigación.

El *Reverse Turing Test* (RTT) es otra táctica de esta misma categoría y plantea un problema bastante más complejo. En lugar de buscarse rastros de la *sandbox*, se comprueba si el sistema está siendo utilizado por una persona. Si no se detecta esa presencia, la carga maliciosa no llega a ejecutarse. La denominación invierte la prueba clásica de Turing, ya que aquí es la máquina la que evalúa si su interlocutor es humano. La variedad casi ilimitada de formas de diseñar estas comprobaciones convierte al RTT en un reto difícil de contrarrestar, cuyo uso se describe como una tendencia al alza [1].

La razón de esa dificultad es que no existe una lista de artefactos que se puedan modificar o falsear para engañar a la comprobación. Lo que se busca no es un dato falseable, sino un recurso del que la *sandbox* carece por definición, que es un usuario real que mueva el ratón, pulse teclas y responda a las ventanas que aparecen. De este modo, una muestra con un mecanismo RTT no se comporta de forma anómala ni levanta sospechas, sino que sencillamente permanece inerte. Desde el punto de vista del analista, el resultado es peor que un fallo evidente, porque el informe se genera sin errores y sin actividad maliciosa, de modo que la muestra puede acabar clasificada como inofensiva (falso negativo)[6].

Esta limitación es conocida y en las *sandboxes* se incorporan módulos con los que se simula actividad del usuario durante el análisis. Sin embargo, esa simulación suele reducirse a desplazar el cursor y generar algún clic, lo que resulta fácilmente identificable por el *malware* evasivo [6]. En los mecanismos RTT documentados en los últimos años se verifican propiedades bastante más exigentes, como la geometría del movimiento [7] o su distribución estadística [8]. La distancia entre lo que se simula en la *sandbox* y lo que se exige desde el *malware* es precisamente el espacio en el que se sitúa este trabajo.

1.2. Objetivo y alcance

En este trabajo se persiguen dos objetivos complementarios. El primero consiste en caracterizar y medir el uso de mecanismos RTT en el *malware* actual. Para ello se identifica qué comprobaciones concretas se emplean, se agrupan y se determina con qué

frecuencia aparecen sobre un conjunto representativo de muestras recientes. Con este objetivo se responde a una cuestión aún abierta [1], que es si se trata de una práctica extendida o si por el contrario es un rasgo específico de ciertas familias de *malware*. El segundo consiste en desarrollar un módulo para CAPEv2 con el que se superen esos mecanismos, de modo que las muestras que los implementan lleguen a ejecutar su carga útil durante el análisis. En el módulo debe simularse la interacción de un usuario con un realismo suficiente para satisfacer las comprobaciones documentadas, y debe hacerse de forma reproducible para que los análisis resulten comparables entre sí.

El alcance queda acotado en tres aspectos. Se aborda únicamente *malware* para Windows, que es la plataforma predominante y aquella para la que están documentados los mecanismos estudiados [1]. La parte dinámica se desarrolla sobre CAPEv2, sin que se pretenda que las soluciones sean directamente portables a otras *sandboxes*, aunque los principios sí lo sean. Por último, la caracterización se limita a los mecanismos basados en interacción del usuario, de modo que el resto de técnicas de evasión, que constituyen un campo mucho más amplio, queda fuera del estudio.

1.3. Estructura de la memoria

El resto de la memoria se organiza como sigue. En el capítulo 2 se presentan los conceptos necesarios para seguir el trabajo. En el capítulo 3 se describe el trabajo realizado. En el capítulo 4 se recogen los resultados obtenidos. En el capítulo 5 se sitúa el trabajo respecto a la investigación previa, recorriendo la evolución de los mecanismos RTT documentados y los trabajos en los que se aborda el mismo problema. Por último, en el capítulo 6 se exponen las contribuciones y se plantean las líneas de trabajo futuro.

Se incluyen además dos anexos. En el Anexo A se recoge la dedicación empleada y su distribución temporal, y en el Anexo B se detalla el contenido del repositorio en el que se publica el material desarrollado.

Capítulo 2

Conceptos previos

El trabajo desarrollado se apoya en dos elementos que se describen a continuación. El primero es la distinción entre los dos enfoques de análisis de *malware*, ya que de sus respectivas limitaciones surge la necesidad de este trabajo. El segundo es la arquitectura de CAPEv2, y en particular la forma en que registra el comportamiento de una muestra, porque determina qué puede observarse durante un análisis y qué no.

2.1. Análisis de *malware* estático y dinámico

En el análisis estático se estudia el binario sin llegar a ejecutarlo. En el caso de Windows, los ejecutables siguen el formato *Portable Executable* (PE), que además del código máquina incluye una serie de estructuras en las que se describe cómo debe cargarse el programa [9]. De todas ellas, la relevante para este trabajo es la tabla de importaciones. En un programa rara vez se implementan por sí mismas operaciones como abrir un fichero o consultar la posición del cursor, sino que se solicitan al sistema operativo. Para que el cargador de Windows resuelva esas llamadas, el binario declara qué funciones necesita y de qué biblioteca de enlace dinámico (del inglés, *Dynamic-Link Library* o DLL) proceden [9]. Esa tabla es, en la práctica, una lista de las capacidades que va a utilizar el programa, y puede obtenerse sin ejecutarlo.

Sobre esa idea se apoya la detección estática de este trabajo. Si en una muestra se importa `GetCursorPos`, resulta razonable sospechar que se consulta la posición del ratón, y si se importa `MessageBoxW`, que se muestra un cuadro de diálogo. No se trata de una prueba concluyente, ya que la presencia de una función no garantiza que llegue a llamarse ni revela con qué propósito, pero es un indicio aplicable a grandes volúmenes de muestras.

El problema es que esa tabla no refleja necesariamente todas las funciones que un programa utiliza. Durante la ejecución pueden realizarse llamadas a funciones que no aparecen en ella. Para ello, en Windows se dispone de la función `GetProcAddress`,

con la que puede obtenerse la dirección de cualquier función del sistema a partir de su nombre. En una muestra que resuelve así sus llamadas (importación dinámica) no es necesario declararlas de antemano, de modo que su tabla de importaciones queda prácticamente vacía. Esta técnica es habitual en el *malware*, y a menudo se combina con el empaquetado o el cifrado del código, con los que se ocultan también las cadenas de texto y el resto del contenido del binario [3].

Por contra, en el análisis dinámico se adopta el enfoque contrario y se ejecuta la muestra en un entorno controlado para observar su comportamiento [2]. En lugar de analizar la lista de funciones declaradas, se registran las que realmente se invocan durante la ejecución, junto con la actividad de red, los ficheros creados o modificados y los cambios en el registro de Windows.

Su principal ventaja frente al estático es que las contramedidas anteriores dejan de ser efectivas [3]. La resolución dinámica de APIs deja de ser un obstáculo, ya que la llamada acaba produciéndose y puede registrarse.

Su principal desventaja es que solo se observa una única ejecución [4]. Todo lo que no se llegue a hacer durante ese intervalo queda fuera del registro, y es precisamente ahí donde operan los mecanismos estudiados en este trabajo. Si no se detectan las condiciones esperadas, el código malicioso no llega a ejecutarse y puede interpretarse erróneamente como ausencia de comportamiento malicioso.

A ello se añade el coste temporal. Mientras que el análisis estático puede completarse en segundos, cada ejecución dinámica requiere mantener la muestra en funcionamiento durante uno o dos minutos, lo que limita el volumen de muestras que pueden procesarse en un tiempo dado.

Ambos enfoques son, por tanto, complementarios. Con el estático se obtiene una cobertura amplia, pero se degrada frente a la ofuscación. Con el dinámico se resiste la ofuscación, pero el resultado depende de que la muestra se decida a actuar.

2.2. *Sandboxes* y arquitectura de CAPEv2

En una *sandbox* de análisis se automatiza el proceso descrito en el apartado anterior [4]. Se recibe una muestra, se ejecuta en un entorno aislado, se recoge su comportamiento y se genera un informe, todo ello sin intervención manual y de forma repetible.

CAPEv2 es la *sandbox* empleada en este trabajo [10]. Deriva de Cuckoo Sandbox, de la que hereda su arquitectura general, y se organiza en dos partes bien separadas. Por un lado, una máquina anfitriona en la que se ejecuta el *framework*, se gestiona la cola de tareas, se controla el hipervisor y se procesan los resultados. Por otro, una o

varias máquinas virtuales invitadas en las que se detonan las muestras.

La comunicación entre ambas se realiza mediante un agente que se ejecuta en la máquina invitada, y en el que se reciben del anfitrión la muestra a analizar y los módulos que deben acompañarla. Cada análisis parte de un estado limpio de la máquina virtual, de modo que ninguna ejecución hereda los efectos de la anterior. Al terminar, la máquina se restaura a ese mismo estado.

El comportamiento se registra mediante el monitor, una biblioteca de enlace dinámico (del inglés, *Dynamic-Link Library* o DLL) llamada `capemon` que se inyecta en el proceso de la muestra al arrancarla. Su funcionamiento se basa en la técnica de *hooking*, que consiste en interceptar las llamadas a funciones del sistema operativo para registrarlas antes de dejar que sigan su curso [2].

Para cada función que se desea vigilar, el monitor implementa una función sustituta en la que se registra la llamada junto con sus argumentos y a continuación se cede el control a la original. La muestra de *malware* ejecuta su código con normalidad, sin percibir diferencia alguna, mientras que el monitor anota cada invocación.

El recorrido desde la llamada hasta el informe final atraviesa dos capas. La primera es el propio monitor, escrito en C y distribuido como una DLL ya compilada, en el que residen el motor de interceptación así como el comportamiento de cada función interceptada. La segunda es una tabla de traducción en Python, situada en el lado del anfitrión, con la que se convierten los eventos registrados por el monitor en las entradas legibles del informe. Para que una llamada aparezca en el informe final debe existir tanto en la primera capa como la entrada correspondiente en la segunda. El recorrido completo se representa en la Figura 2.1, en la que se destacan además los dos puntos de la arquitectura sobre los que se ha actuado en este trabajo.

En CAPEv2 se admiten módulos auxiliares que se ejecutan en paralelo a la muestra en la máquina invitada durante el análisis. Su cometido es preparar o alterar el entorno de ejecución, y se activan mediante las opciones indicadas al enviar la tarea.

Entre ellos figura el módulo de simulación de interacción que se incorpora por defecto, con el que se genera movimiento del cursor y clics mientras la muestra se ejecuta. Este es el mecanismo previsto en CAPEv2 para contrarrestar las comprobaciones de presencia humana y foco de estudio y mejora en este TFG.

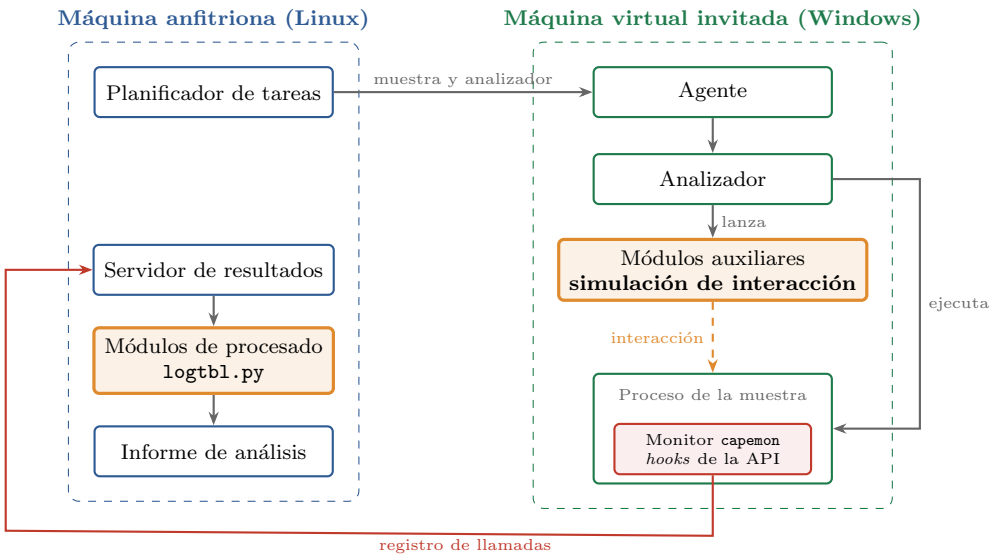

Figura 2.1: Arquitectura de CAPEv2 y recorrido de una llamada a la API hasta el informe.

Capítulo 3

Análisis, diseño y desarrollo

El desarrollo no partió de una especificación previa, sino del análisis de muestras reales. Antes de construir ningún componente se determinó qué comprueba exactamente cada familia y bajo qué condiciones ejecuta su carga útil, y de ahí surgieron los requisitos que condicionaron todo el diseño posterior. Ese es el punto de partida de este capítulo, del que se derivan los tres componentes desarrollados.

Este capítulo se organiza en cinco partes. En la primera se recoge la caracterización experimental de las muestras y los requisitos que de ella se derivan. En la segunda se presenta la clasificación de mecanismos RTT que resulta de ese análisis y del estudio de la literatura. Las tres siguientes describen los componentes desarrollados, que son el módulo de simulación de interacción humana, el conjunto de reglas de detección estática y la corrección aplicada sobre la instrumentación dinámica de CAPEv2. El capítulo se cierra indicando dónde se publica todo el material.

3.1. Caracterización experimental del problema

Se seleccionaron cuatro casos, cada uno representativo de un mecanismo de naturaleza distinta. `Gozi/Ursnif`, en el que se emplea el desplazamiento del cursor como material criptográfico. `LummaC2 v4.0`, en el que se analiza la geometría del movimiento. `LummaC2 2025`, en el que se condiciona la ejecución a la confirmación de un cuadro de diálogo. Por último, se ha considerado `Pafish` [11], herramienta de referencia con la que se dispone de siete comprobaciones de interacción independientes. De este análisis inicial se desprenden los requisitos que condicionaron el diseño del módulo.

El primero es que no basta con que el cursor cambie de posición. En `Gozi/Ursnif` se descarta el movimiento cuando se detectan coordenadas repetidas, de modo que cada paso debe producir un cambio real de posición.

El segundo es que la trayectoria debe ser geométricamente plausible. En `LummaC2`

`v4.0` se muestrean posiciones consecutivas del cursor y se calculan los ángulos entre los vectores que forman, con lo que se descartan los recorridos con cambios de dirección abruptos.

El tercero es que la simulación no debe interrumpirse durante intervalos prolongados. El tiempo transcurrido desde la última entrada del usuario se emplea también como indicador de presencia, de modo que una pausa larga equivale a un sistema desatendido. La restricción no consiste en mantener una actividad ininterrumpida, puesto que en el manejo real de un equipo también se producen pausas. Lo que se requiere es que esas pausas sean breves e irregulares, de modo que no se alcance el umbral de inactividad sin renunciar por ello a la variabilidad propia de una persona.

El cuarto es que los clics deben generarse como eventos físicos de entrada y no como mensajes dirigidos a una ventana. Las comprobaciones más exigentes se apoyan en *hooks* de bajo nivel, con los que solo se captura la entrada que el sistema considera real.

El quinto, y el más restrictivo, es que la confirmación de diálogos exige dos técnicas distintas y mutuamente excluyentes. En `Pafish` se requiere un clic físico con el cursor situado sobre la ventana del diálogo, y se rechaza el mensaje lógico de pulsación. En `LummaC2 2025` ocurre lo contrario, ya que solo se reacciona al mensaje lógico y se ignora el clic físico posicional. Ninguna de las dos técnicas es suficiente por separado.

3.2. Taxonomía de mecanismos *Reverse Turing Test*

Existen cinco mecanismos de RTT. La distinción entre ellos no es meramente descriptiva, ya que cada uno exige una respuesta distinta por parte de la simulación y deja un rastro diferente en el binario.

Consulta de la posición del cursor. Se consultan de forma repetida las coordenadas del puntero mediante `GetCursorPos` y se comparan las lecturas sucesivas. La ausencia de variación indica que nadie está manejando el equipo. En las variantes más elaboradas no se comprueba únicamente que haya cambios, sino que se analiza la geometría del recorrido [7].

Consulta del estado de teclas o botones. Se comprueba si en un instante dado hay alguna tecla o botón pulsado, mediante `GetAsyncKeyState` o `GetKeyState`. A diferencia del anterior, no se observa una trayectoria sino un estado puntual, por lo

que se requiere que la pulsación coincida en el tiempo con la comprobación [1].

Espera de confirmación mediante diálogo. Se muestra un cuadro de diálogo con `MessageBox` o `DialogBox` y se condiciona la continuación de la ejecución a que alguien pulse uno de sus botones. Es un mecanismo especialmente eficaz porque la llamada bloquea la ejecución hasta obtener respuesta, de modo que en un entorno sin usuario no se llega nunca a la siguiente instrucción [1].

Medición del tiempo de inactividad. Se consulta cuánto tiempo lleva el sistema sin recibir entrada del usuario mediante `GetLastInputInfo`. El razonamiento es que en un equipo en uso se registra actividad con frecuencia, mientras que un entorno de análisis suele permanecer desatendido. Si el tiempo transcurrido supera cierto umbral, se concluye que no hay nadie y se suspende la actividad [12].

Instalación de *hooks* de entrada de bajo nivel. En lugar de sondearse el estado del sistema, se instala un *hook* sobre la cola de entrada mediante `SetWindowsHookEx`, indicando con una constante que debe interceptarse la entrada a bajo nivel [13]. A partir de ese momento el programa se limita a esperar, y es el sistema el que le notifica cada clic o pulsación que se produzca. Al no realizarse consultas repetidas, este mecanismo apenas deja rastro en el registro de llamadas, ya que la única invocación relevante es la que instala el *hook*.

Los cuatro primeros mecanismos comparten un rasgo que conviene destacar, y es que en todos ellos el programa consulta al sistema por el estado de la interacción. En el quinto se invierte esa relación, ya que el programa se limita a esperar y es el sistema quien le notifica cada evento. Esta diferencia condiciona la simulación. Una posición del cursor modificada por software basta para satisfacer las comprobaciones que consultan, pero no genera el evento que el sistema debería notificar, de modo que el mecanismo basado en *hooks* continúa esperando indefinidamente.

3.3. Módulo de simulación de interacción humana

El módulo de simulación de interacción humana se ejecuta como módulo auxiliar de CAPEv2 dentro de la máquina virtual de análisis, en paralelo a la muestra, y se activa mediante la opción `human_improved=1` al lanzar el análisis. Toda su lógica reside en un único hilo en el que se ejecuta un bucle continuo. En cada iteración se realizan de forma secuencial tres acciones, que son un desplazamiento del cursor, con cierta probabilidad un clic simple o doble, y la comprobación de si ha aparecido algún cuadro de diálogo

pendiente de confirmar. En la Figura 3.1 se representa mediante un diagrama UML de secuencia el comportamiento del módulo a lo largo de una iteración del bucle principal.

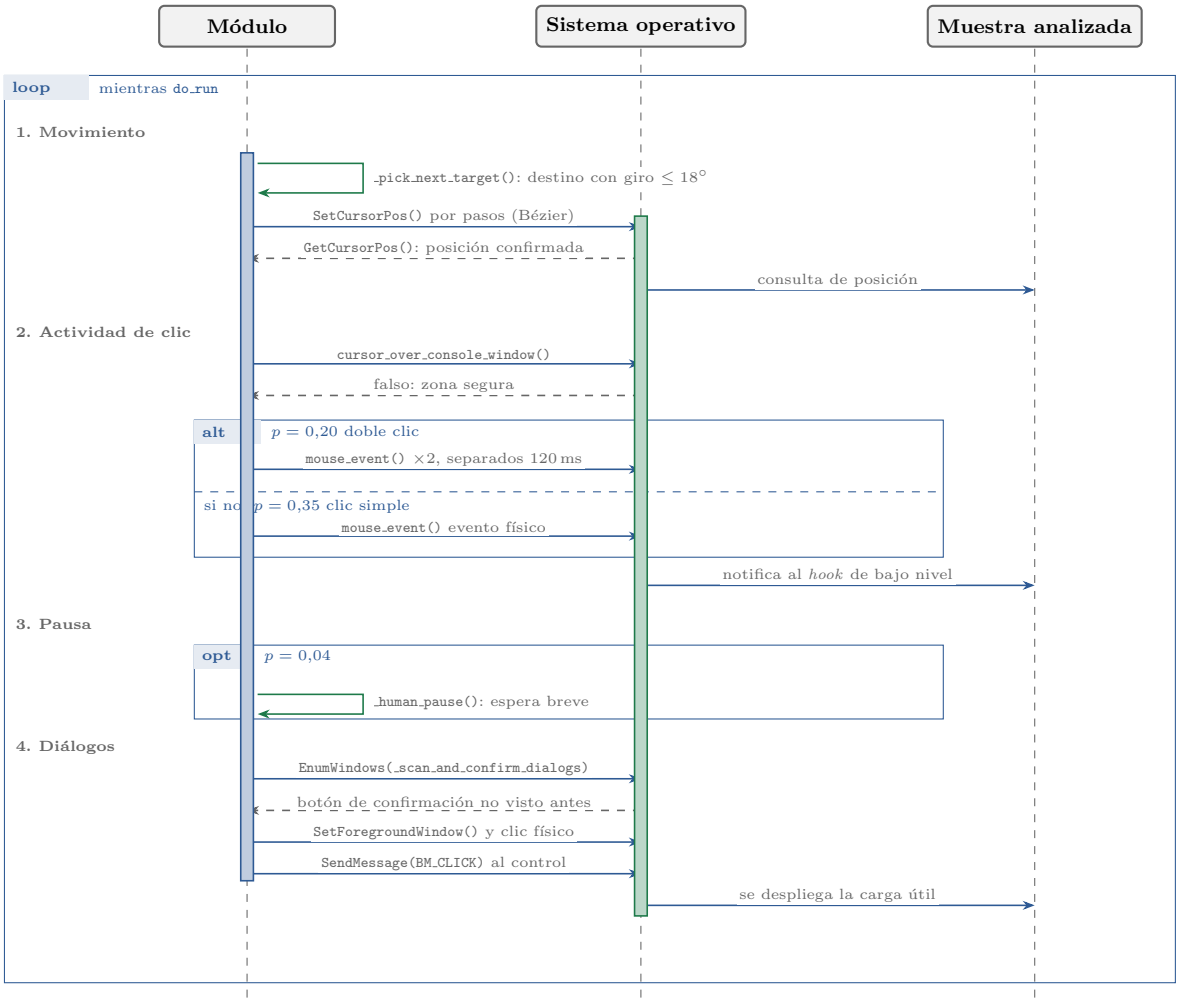

Figura 3.1: Diagrama UML de secuencia del módulo en una iteración del bucle principal.

La decisión de integrar todo en un solo hilo fue la más relevante del diseño y se adoptó como consecuencia de un problema detectado experimentalmente. En una versión previa, el movimiento del cursor y la gestión de diálogos residían en dos módulos independientes que se ejecutaban de forma concurrente. Al compartir ambos un único cursor físico se producía una condición de carrera, ya que mientras en un módulo se llevaba el cursor sobre el botón del diálogo para confirmarlo, en el otro se desplazaba a una zona distinta de la pantalla y la acción quedaba invalidada.

El efecto observado era que se superaban las comprobaciones de movimiento o las de diálogo, pero nunca ambas de forma simultánea. La ejecución secuencial en un único hilo resuelve el conflicto de raíz, puesto que en cada instante solo se dispone de un componente actuando sobre el cursor.

El motor de movimiento del cursor es el componente central y el más elaborado. Su cometido es generar un movimiento del ratón indistinguible del de una persona, ya

que en varias familias no se comprueba únicamente que el cursor se desplace, sino que se analiza la naturaleza de ese desplazamiento.

Cada trazo del cursor sigue una curva de Bézier cuadrática en lugar de una línea recta, dado que en un movimiento humano nunca se traza una recta perfecta. Con esa ligera curvatura se introduce una desviación natural en el recorrido. Este mismo enfoque se emplea en otros trabajos que abordan la simulación de interacción humana frente a sistemas de detección automática [14].

Sobre esa trayectoria se modela la velocidad según el modelo de mínimo tirón de Flash y Hogan [15], con el que se describe cómo se aceleran y desaceleran los movimientos humanos de alcance. El cursor arranca despacio, acelera en el tramo central y frena de forma suave al llegar al destino. La duración de cada trazo se calcula en función de la distancia a recorrer siguiendo la ley de Fitts [16], y se le añade una pequeña variación aleatoria para que no se repitan dos trazos idénticos. El perfil resultante se representa en la Figura 3.2.

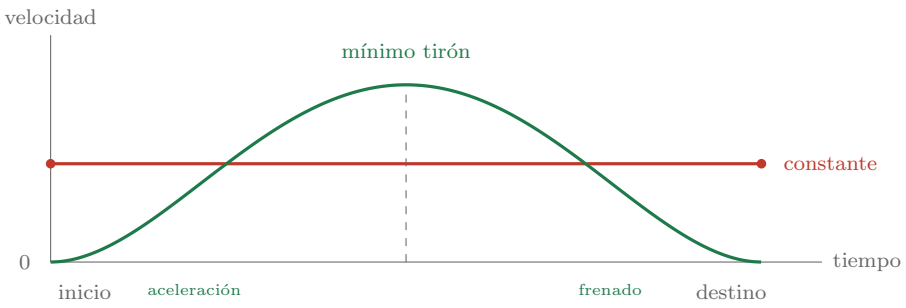

Figura 3.2: Perfil de velocidad a lo largo de un trazo simulado por el módulo siguiendo el modelo de mínimo tirón.

La continuidad entre trazos se controla de forma explícita. El cambio de dirección de un trazo al siguiente se acota a un máximo de 18°. Este detalle resultó determinante para `LummaC2 v4.0`, en cuya comprobación se calculan los ángulos entre vectores de posiciones consecutivas y se rechaza el movimiento si alguno supera los 45°. El valor de 18°se fijó muy por debajo de ese umbral, de modo que cualquier trayectoria generada lo satisface con margen suficiente y la activación deja de depender del azar. En la Figura 3.3 se muestra un ejemplo de trayectoria generada por el módulo.

Por último, se garantiza que en cada paso se produzca un cambio efectivo de coordenadas sin repeticiones. Este aspecto, aparentemente menor, era necesario para `Gozi/Ursnif`, donde el movimiento se descarta si se detectan coordenadas repetidas.

Algunas comprobaciones no se apoyan en el movimiento, sino en la existencia de pulsaciones del ratón. En el módulo se generan clics simples y dobles con una probabilidad calibrada en cada iteración del bucle. La zona en la que se generan los

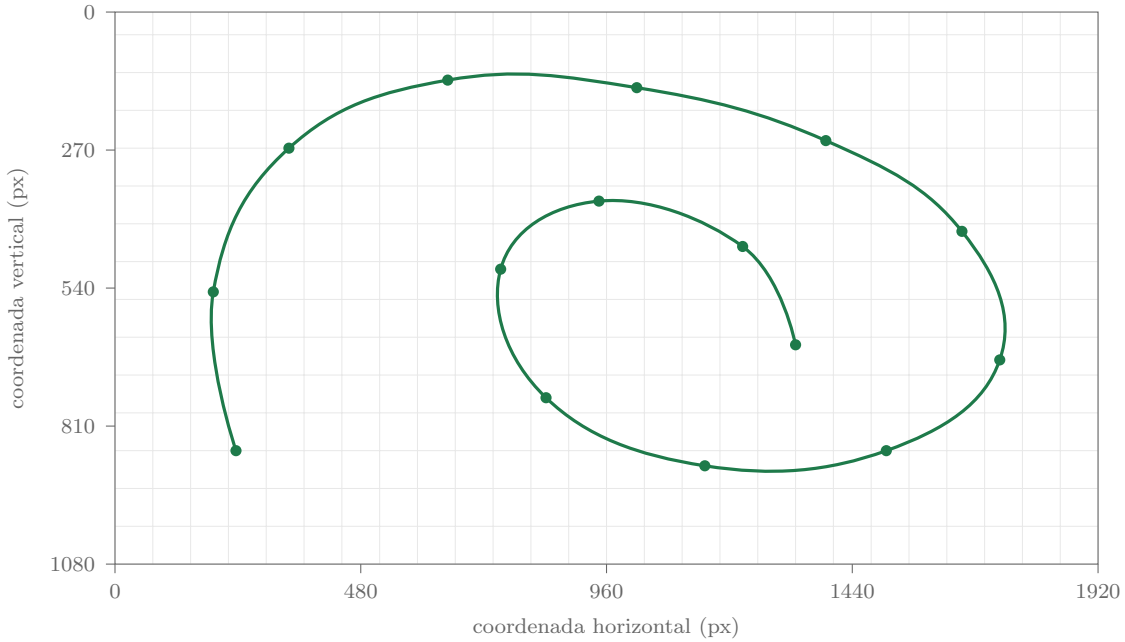

Figura 3.3: Trayectoria del cursor generada por el módulo a lo largo de la pantalla. Cada punto marca el final de un trazo.

clics también está acotada. Se reserva un margen de 12 píxeles respecto a los bordes de la pantalla, y se descarta la cuarta parte izquierda del escritorio para evitar pulsar sobre los iconos que se sitúan habitualmente en esa región.

Los clics se generan mediante eventos de ratón físicos, a bajo nivel del sistema, y no mediante mensajes lógicos dirigidos a una ventana. La distinción es determinante, ya que en las comprobaciones más exigentes se instala un *hook* de ratón de bajo nivel con el que solo se captura la entrada que el sistema considera real, de modo que un mensaje lógico no llega a notificarse.

Por último, el mecanismo RTT más restrictivo consiste en mostrar un cuadro de diálogo y condicionar la ejecución de la carga útil a que un humano lo confirme. En un entorno de análisis en el que se ignore el diálogo, la muestra permanece bloqueada de forma indefinida.

En el módulo se aborda este mecanismo en tres pasos. En primer lugar, en cada iteración se recorren las ventanas del sistema en busca de botones cuyo texto corresponda a una confirmación, contemplando las formas habituales en varios idiomas. Cada diálogo se distingue por su identificador de ventana, con lo que se atienden los nuevos sin volver a pulsar uno ya confirmado. A continuación se trae la ventana del diálogo al primer plano, condición necesaria para que la pulsación surta efecto. En el tercer paso se confirma el diálogo mediante dos técnicas encadenadas. Primero se realiza un clic físico sobre el botón, situando el cursor sobre él. Después se envía un mensaje lógico de pulsación dirigido directamente al control.

3.4. Conjunto de reglas `capa` para detección estática

La detección estática se aborda mediante `capa` [17], herramienta con la que se identifican capacidades en un ejecutable evaluando reglas declarativas sobre las características extraídas del binario. Se optó por esta herramienta en lugar de por un programa a medida por dos motivos.

El primero es que la lógica de detección queda expresada de forma legible y verificable en ficheros independientes, lo que facilita su revisión y su reutilización. El segundo, y más relevante para este trabajo, es que una misma regla puede aplicarse tanto sobre el binario como sobre el informe de comportamiento generado por la *sandbox*, de modo que las vías estática y dinámica se abordan con un único conjunto de reglas.

Se desarrollaron siete reglas propias, agrupadas bajo el espacio de nombres `anti-analysis/anti-sandbox/reverse-turing-test`. Cinco de ellas se corresponden con los mecanismos de la clasificación presentada en la Sección 3.2, a las que se suman una regla exploratoria y una regla agregadora.

Cada regla se compone de dos bloques. En `meta` se recogen los metadatos, que no intervienen en la detección. En `features` reside la lógica, expresada como un árbol de condiciones cuyas hojas son características extraídas del binario y cuyos nodos son los operadores lógicos `or` y `and`.

En todas las reglas se declara el ámbito estático `function` y el ámbito dinámico `call`. Con la primera declaración se exige que las características se encuentren dentro de una misma función del binario. Con la segunda se habilita la aplicación de la misma regla sobre los informes de comportamiento de CAPEv2, sin modificación alguna.

Cuatro de las cinco reglas del núcleo siguen un patrón simple, en el que se enumeran bajo un operador `or` las funciones asociadas al mecanismo. La regla de estado de teclas agrupa `GetAsyncKeyState` y `GetKeyState`, y la de confirmación de diálogo enumera las funciones de las familias `MessageBox` y `DialogBox`.

La quinta regla presenta una estructura distinta y responde a un hallazgo obtenido durante la validación. Al probarse el conjunto inicial contra `Pafish`, solo casó la regla de consulta del cursor, pese a que en la herramienta se implementan siete comprobaciones de interacción. La inspección de su tabla de importaciones reveló la causa, ya que la actividad de clic no se detecta mediante sondeo del estado de los botones, sino instalando un *hook* de entrada de bajo nivel y esperando los eventos a través de un bucle de mensajes.

Se trata de un mecanismo documentado y empleado tanto en UpClicker [13] como en BaneChant [18], que no estaba cubierto por el conjunto inicial. Se desarrolló en

consecuencia la regla correspondiente, en la que se exige mediante un operador `and` tanto la llamada a `SetWindowsHookEx` como la presencia, en la misma función, de la constante que identifica el *hook* como de bajo nivel. Esa restricción es necesaria porque la llamada por sí sola aparece en abundante software legítimo, como gestores de portapapeles o utilidades de accesibilidad, y sin ella la regla resultaría inservible por exceso de falsos positivos.

La misma inspección de `Pafish` reveló el uso de `GetDoubleClickTime`, con la que se consulta el umbral temporal que el sistema considera un doble clic. Un programa en el que se compara la cadencia de dos clics consecutivos contra ese umbral no verifica que existan clics, sino que el ritmo entre ellos sea el propio de una persona.

Se desarrolló una regla para ese mecanismo, marcada de forma explícita como exploratoria y mantenida deliberadamente fuera del indicador agregado. El motivo es que esa función también aparece en software gráfico legítimo, en el que se emplea para calibrar la gestión de eventos, por lo que su capacidad para distinguir *malware* de software normal quedaba por determinar.

La séptima regla no se apoya en ninguna función propia. Su bloque `features` consiste en un operador `or` de cinco hojas `match`, con las que se referencia por su nombre a cada una de las reglas del núcleo. Con ella se obtiene el veredicto binario de si en una muestra se implementa algún mecanismo RTT, que es el indicador sobre el que se calcula la prevalencia agregada.

3.5. Extensión de la instrumentación dinámica de CAPEv2

La aplicación de las reglas sobre los informes de comportamiento solo resulta efectiva si las funciones buscadas llegan a registrarse. Por ese motivo se revisó qué funciones RTT se monitorizan realmente en CAPEv2.

El diagnóstico se realizó sobre ambas capas por separado, comprobando la presencia de las funciones de interceptación en el monitor y la de las entradas correspondientes en la tabla de traducción. Los cinco mecanismos se reparten en tres situaciones distintas.

En la consulta de la posición del cursor y en la instalación de *hooks* de entrada se dispone de interceptación y de traducción, de modo que ambas llegan al informe sin necesidad de intervención. En la consulta del tiempo de inactividad y en la del estado de teclas se dispone de interceptación en el monitor, pero faltaba la entrada en la tabla de traducción. El evento se capturaba y se perdía antes de llegar al informe. En las funciones de confirmación de diálogo no se dispone de interceptación en el monitor, por lo que la llamada no se registra en ningún momento.

Los dos casos en los que solo faltaba la traducción se resolvieron añadiendo las entradas correspondientes a la tabla, sin necesidad de recompilar el monitor. La corrección se verificó de extremo a extremo con la función de consulta del tiempo de inactividad. Tras reprocesar un análisis previo, la llamada aparece registrada en el informe y la regla asociada casa sobre él en modo dinámico, cuando antes no era posible.

El tercer caso no admite esa solución. Al no existir la interceptación en el monitor, la tabla de traducción no dispone de ningún evento que convertir, de modo que la corrección aplicada resulta condición necesaria pero no suficiente. Para cubrir las funciones de diálogo sería preciso implementar la interceptación en el propio monitor y volver a compilarlo, tarea que queda planteada como trabajo futuro en el capítulo 6.

3.6. Disponibilidad del código

Todo el material desarrollado se encuentra disponible en un repositorio público[1]. Dado que se trata de una bifurcación del proyecto CAPEv2, las aportaciones propias se localizan en rutas concretas dentro del árbol del proyecto, que se detallan en el Anexo B. En él se incluyen el módulo de simulación de interacción humana, las reglas capa desarrolladas, la corrección aplicada sobre la tabla de traducción, los scripts empleados en el estudio de prevalencia y el listado del corpus con su etiquetado.

[1] `https://github.com/Nicofaroo/CAPEv2-Trigger-Detection`

Capítulo 4

Experimentación y discusión de resultados

En este capítulo se presentan los resultados obtenidos. El módulo de simulación y las reglas de detección se evaluaron de forma distinta. El primero se validó sobre casos concretos, comprobando si las muestras llegan a ejecutar su carga útil, mientras que las segundas se aplicaron sobre un corpus amplio contrastado con un grupo de control. En primer lugar se describe el entorno de evaluación y el procedimiento seguido. Después, se analizan los resultados y se discuten las limitaciones de la detección estática.

4.1. Entorno de evaluación

El entorno de análisis se compone de una máquina anfitriona en la que se ejecuta el entorno CAPEv2 con el módulo desarrollado y una máquina virtual invitada en la que se detonan las muestras. Como anfitrión se empleó un equipo ASUS Zenbook S16 UM5606WA, con procesador AMD Ryzen AI 9 365 de 10 núcleos y 32 GB de memoria LPDDR5X, sobre Ubuntu 24.04 LTS. La virtualización se gestionó mediante KVM y QEMU a través de libvirt.

La máquina invitada es una instalación de Windows 10 Pro x64, en la que se ejecuta el agente de CAPEv2 sobre Python 3.11. Como se describió anteriormente, todos los análisis parten de un mismo estado limpio, con el agente activo y Windows Defender deshabilitado, de modo que ninguna ejecución hereda los efectos de la anterior. En la configuración de CAPEv2 la máquina se declara con arquitectura x86, parámetro que determina la variante del monitor que se inyecta durante el análisis y garantiza la compatibilidad con las muestras en formato PE32, mayoritarias en el corpus.

La conexión de red se establece mediante NAT a través de la interfaz virtual del anfitrión, con salida real a internet enrutada por su interfaz física. Esta configuración es necesaria porque el criterio de activación empleado se basa en la comunicación con la infraestructura de mando y control, que no podría observarse en un entorno de red

simulado.

Para el análisis estático se empleó capa 9.4.0 en su distribución autónoma [17]. Dado que en esta herramienta se sustituye el conjunto de reglas propio por el que se le indique, se construyó un árbol de reglas fusionado en el que se incorporaron las reglas propias al repositorio oficial. Con esta disposición se obtienen en una única pasada tanto las detecciones de mecanismos RTT como las proporcionadas por las reglas oficiales.

4.2. Procedimiento experimental

La validación del módulo se apoya en un diseño experimental en el que cada muestra se ejecuta un mínimo de cinco veces bajo cada una de las cinco condiciones, con el fin de descartar activaciones puntuales debidas al azar. Las diferencias observadas entre ejecuciones pueden atribuirse así a la condición de interacción.

Las condiciones cubren un espectro progresivo de realismo. La primera es la ausencia total de interacción, obtenida desactivando toda simulación, y actúa como control negativo. La segunda introduce un movimiento deliberadamente mecánico, generado por un módulo desarrollado para este fin en el que el cursor se desplaza en líneas rectas a velocidad constante entre puntos fijos. Su propósito no es superar las comprobaciones, sino confirmar que están operativas, ya que una muestra que rechaza este movimiento demuestra que verifica algo más que la mera existencia de desplazamiento. La tercera emplea el módulo de simulación que se incorpora por defecto en CAPEv2, con la que se establece la capacidad actual de la herramienta. La cuarta emplea el módulo desarrollado en este trabajo. La quinta es la interacción humana real, con el operador manejando directamente la máquina virtual, y actúa como control positivo.

Nótese que se requiere un criterio objetivo con el que determinar si la carga maliciosa se ha ejecutado. La señal más clara y difícil de falsear es la comunicación con la infraestructura de mando y control, es decir, los servidores a los que la muestra intenta conectarse una vez desplegada su carga. En una muestra inactiva no se produce esa comunicación, mientras que en una muestra activa se intenta resolver y conectar con sus dominios característicos.

Se atiende además a las llamadas al sistema registradas durante la ejecución y a las firmas de comportamiento generadas por la propia *sandbox*. Este criterio múltiple resulta necesario porque en algunas muestras se emplean técnicas de evasión de *hooks* o llamadas al sistema directas, con lo que no se registra ninguna API y solo queda disponible el indicador de red.

La medición de prevalencia sigue un procedimiento distinto, orientado al volumen. Se construye un corpus amplio de muestras y se le aplica de forma automatizada el

conjunto de reglas descrito en la Sección 3.4, contrastando los resultados con un grupo de control de software legítimo.

Como fuente de muestras se emplean los lotes diarios de MalwareBazaar [19], repositorio público en el que se publican muestras confirmadas con metadatos de familia. Se descartó recurrir a plataformas de búsqueda selectiva de muestras, ya que localizar muestras a partir de la propia característica que se desea medir produciría un resultado artificialmente alto y no una estimación de prevalencia.

4.3. Validación del módulo sobre casos de estudio

El módulo se validó contra cuatro casos, cada uno representativo de un mecanismo de naturaleza distinta.

4.3.1. `Gozi/Ursnif`

En esta muestra no se realiza una comprobación de tipo aprobado o rechazado, sino que los desplazamientos del cursor se emplean como entrada funcional de la rutina de descifrado. Sin movimiento no se obtiene una semilla válida y el descifrado no llega a completarse. Al emplearse técnicas de evasión de *hooks*, no se registran llamadas a la API, de modo que el indicador aplicable es la actividad de red.

En todas las condiciones se desempaqueta una primera capa común, independiente de la interacción. La diferencia aparece en la red. Sin interacción alguna, con el movimiento mecánico y con el módulo por defecto el comportamiento es idéntico, ya que en ninguno de los tres casos se llega a contactar con ninguna dirección de mando y control. Solo se observa tráfico de fondo del sistema operativo, y el análisis termina por agotamiento del tiempo asignado, fijado en 360 segundos. Con el módulo desarrollado se alcanza en cambio la fase de comunicación y se contacta con 5 direcciones IP de mando y control propias de la familia en el puerto 80, entre ellas 92.223.97.97 y 91.81.128.227. El mismo comportamiento se obtiene con interacción humana real.

4.3.2. `LummaC2 v4.0`

En esta variante no se comprueba únicamente que el ratón se mueva, sino que se analiza la geometría del movimiento. Los resultados obtenidos en cada condición se recogen en la Tabla 4.1.

Con el módulo desarrollado se alcanza el comportamiento completo de forma reproducible. Se resuelven los dominios de mando y control característicos de la familia, como `curtainjors.fun` o `gogobad.fun`, y se dispara la firma `mouse_movement_detect`,

Condición	APIs únicas	Actividad de red	Carga
Sin interacción	29	Ninguna	Inactiva
Movimiento mecánico	29	Ninguna	Inactiva
Módulo por defecto	29–72	Intermitente	Intermitente
Módulo desarrollado	74	Completa	Completa
Interacción humana real	74	Completa	Completa

Tabla 4.1: Comportamiento de `LummaC2 v4.0` según la condición de interacción.

con la que CAPEv2 indica de forma explícita que se ha dado por válido el movimiento del ratón.

La diferencia frente al módulo por defecto reside en la continuidad de rumbo. En el módulo por defecto se generan trazos con cambios de dirección arbitrarios que solo por casualidad caen bajo el umbral exigido por la muestra, lo que explica la activación intermitente recogida en la Tabla 4.1. En el motor de movimiento desarrollado cada giro se acota a un máximo de 18°, de modo que toda trayectoria generada cumple por construcción el requisito geométrico.

4.3.3. `LummaC2 2025`

En esta variante la ejecución no se condiciona al movimiento del ratón, sino a la confirmación de un cuadro de diálogo que aparece al iniciarse. En la traza registrada no aparece ninguna de las funciones asociadas a los mecanismos de interacción, por lo que la propia comprobación no resulta observable. El indicador aplicable es en este caso la resolución de dominios, que sí queda recogida. Los resultados obtenidos en cada condición se recogen en la Tabla 4.2.

Condición	Diálogo confirmado	Dominios resueltos
Sin interacción	No	Ninguno
Movimiento mecánico	No	Ninguno
Módulo por defecto	No	Ninguno
Módulo desarrollado	Sí	8
Interacción humana real	Sí	8

Tabla 4.2: Comportamiento de `LummaC2 2025` según la condición de interacción.

Con el módulo desarrollado se registra en el análisis la detección y confirmación del botón, tras lo cual se despliega el comportamiento completo. Se resuelven 8 dominios de mando y control, como `naturewsounds.help` o `nestlecompany.pro`, a los que se suma la conexión con `steamcommunity.com`, empleada por esta familia como vector de robo de credenciales. Se activan además 5 firmas de comportamiento del *sandbox*, entre ellas `network_cnc_https_generic` y `direct_syscall_evasion`.

4.3.4. `Pafish`

`Pafish` es una herramienta de referencia diseñada para poner a prueba la capacidad de una *sandbox* de eludir técnicas anti-análisis [11]. Su batería de comprobaciones de interacción resulta especialmente exigente, y a diferencia de las muestras anteriores se informa por pantalla del resultado de cada comprobación individual. El resultado comparado se recoge en la Tabla 4.3.

Comprobación	**Módulo por defecto**	**Módulo desarrollado**
Presencia del ratón	✓	✓
Movimiento del ratón	✓	✓
Velocidad del movimiento	✗	✓
Actividad de clic	✓	✓
Actividad de doble clic	✗	✓
Confirmación de diálogo	✗	✓
Confirmación de diálogo plausible	✗	✓

Tabla 4.3: Comprobaciones de interacción de `Pafish` superadas en cada condición.

Con el módulo desarrollado se superan las siete comprobaciones de forma simultánea, frente a las tres que se superan con el módulo por defecto. La última es la más estricta, ya que en ella no solo se exige pulsar el botón, sino que el cursor se encuentre físicamente sobre la ventana del diálogo en el instante de la confirmación.

4.4. Estudio de prevalencia de mecanismos RTT

4.4.1. Construcción del corpus

Se seleccionaron 6 lotes diarios de MalwareBazaar, agrupados en tres ventanas temporales separadas entre marzo de 2025 y julio de 2026, con el fin de evitar que los resultados quedasen condicionados por la actividad puntual de una campaña concreta. El material descargado se sometió a un filtrado en cuatro etapas, cuyo resultado se recoge en la Tabla 4.4.

Etapa	**Ficheros**
Extraídos de los seis lotes	11.374
Descartados por no ser PE válidos	6.866
Descartados por estar vacíos	2.522
Descartados por duplicación (SHA-256)	118
Descartados por tamaño	34
PE únicos resultantes	1.834

Tabla 4.4: Filtrado del material descargado.

En total, se obtuvieron 1.834 ficheros PE únicos. Esta cifra no constituye, sin embargo, un corpus válido, y conviene detallar el motivo. Los lotes descargados no contienen únicamente muestras, sino que algunas de ellas son contenedores de los que, al descomprimirlos, se extraen otros ficheros ejecutables en su interior. Cabe distinguir por tanto entre los ficheros de primer nivel, que son los que aparecen directamente en el lote y se corresponden con las muestras publicadas, y los ficheros anidados, obtenidos al descomprimir a los anteriores.

Al consultar en la plataforma el identificador de cada fichero se observó que esa distinción resulta determinante. Entre los ficheros de primer nivel el 78 % figura publicado como muestra, frente a menos del 1 % entre los anidados. La explicación es que un fichero extraído del interior de una muestra no constituye por sí mismo una muestra publicada, aunque proceda de un lote de la plataforma.

En su mayoría, esos ficheros anidados son componentes legítimos incorporados en instaladores, y no código malicioso. Se adoptó en consecuencia como criterio de inclusión que el fichero figure publicado como muestra de *malware* en la plataforma, con lo que el corpus experimental quedó constituido por 653 muestras, de las cuales 511 disponen de etiqueta de familia.

Ese criterio excluye también al 22 % de los ficheros de primer nivel que no figuraba publicado. Al no poder confirmarse su condición de muestra, se descartaron igualmente, en coherencia con el mismo criterio.

Como grupo de control se emplearon 700 binarios legítimos procedentes del directorio de sistema de la propia máquina virtual, seleccionados de forma aleatoria. Su tamaño se fijó deliberadamente próximo al del grupo experimental.

Esta elección presenta una limitación que conviene señalar. Al tratarse de componentes del sistema operativo, la mayoría están firmados por Microsoft y carecen de interfaz gráfica, por lo que su perfil de importaciones puede diferir del de una aplicación de usuario convencional.

4.4.2. Aplicabilidad del análisis por importaciones

Las reglas desarrolladas operan sobre la tabla de importaciones. Cuando en una muestra se resuelven las APIs en tiempo de ejecución, esa tabla queda prácticamente vacía y el nombre de la función nunca aparece donde las reglas lo buscan. En esos casos un resultado negativo no significa que el mecanismo no exista, sino que el método no resulta aplicable.

Resulta necesario, por tanto, distinguir unas muestras de otras antes de calcular cualquier proporción. La variable con la que mejor se captura esa condición es el número de importaciones, ya que una muestra que oculta las funciones que utiliza declara muy

pocas o ninguna.

Su distribución, recogida en la Figura 4.1, revela dos comportamientos muy distintos. En el grupo experimental los percentiles 10 y 25 se sitúan en una sola importación, y el 42 % de las muestras del corpus declara menos de 20. En el control esa proporción no llega al 5 % y el percentil 10 alcanza ya las 44 importaciones.

Se adoptó en consecuencia el umbral de 20 importaciones. Su justificación no reside en el valor concreto, sino en que se sitúa donde la distribución cambia de comportamiento, de modo que no divide una población continua sino que aísla el grupo de muestras que ocultan sus importaciones.

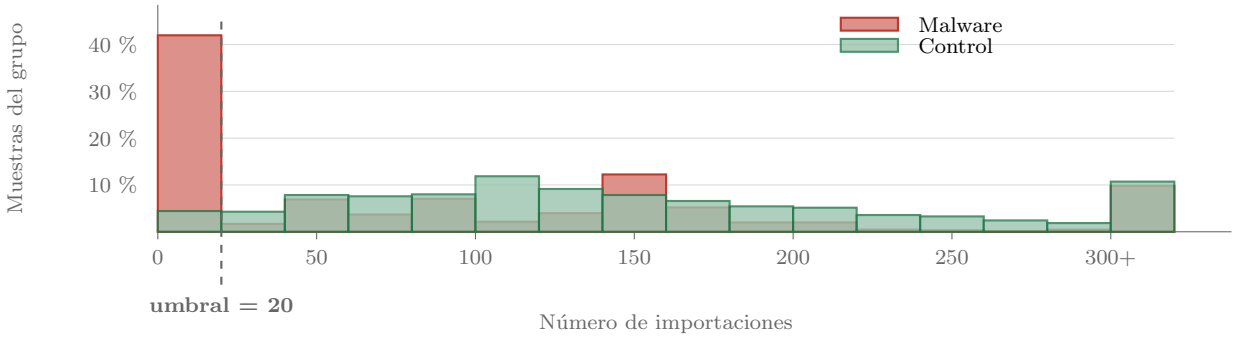

Figura 4.1: Distribución del número de importaciones en ambos grupos, expresada como porcentaje de muestras de cada grupo.

4.4.3. Resultados

Al separar las muestras según ese criterio se obtiene el resultado central del estudio, recogido en la Tabla 4.5. De las 653 muestras del corpus experimental y las 700 del grupo de control se analizaron finalmente 569 y 675 respectivamente. Las 84 y 25 restantes no pudieron procesarse por superar el límite de tiempo fijado en 600 segundos por muestra o por errores durante el análisis, y quedan excluidas de los cálculos posteriores.

Grupo	Estrato	Muestras	Con RTT	Prevalencia
Experimental	No aplicable	258	16	6,2 %
Experimental	Aplicable	311	167	53,7 %
Control	No aplicable	31	0	0 %
Control	Aplicable	644	65	10,1 %

Nota: En el estrato no aplicable la proporción no debe interpretarse como una medida de prevalencia.

Tabla 4.5: Prevalencia de RTT según la aplicabilidad del análisis por importaciones

Sobre las muestras en las que el análisis resulta aplicable, en el 53,7 % del *malware* se implementa algún mecanismo RTT, frente al 10,1 % del software legítimo comparable.

El mecanismo aparece por tanto unas cinco veces más en el grupo experimental que en el de control. Para comprobar que esa diferencia no responde al azar se aplicó la prueba exacta de Fisher, que arroja una razón de posibilidades de 10,3 y un valor p del orden de 10^{-47}, es decir, una probabilidad prácticamente nula de obtener un resultado así por casualidad.

Dentro del propio grupo experimental, el contraste entre estratos también resulta llamativo. En el estrato no aplicable el mecanismo solo se detecta en el 6,2 % de las muestras, muy por debajo del 53,7 % obtenido en el aplicable. Esa diferencia no refleja un comportamiento distinto del *malware* en cada estrato, sino la pérdida de capacidad de detección cuando la tabla de importaciones está vacía.

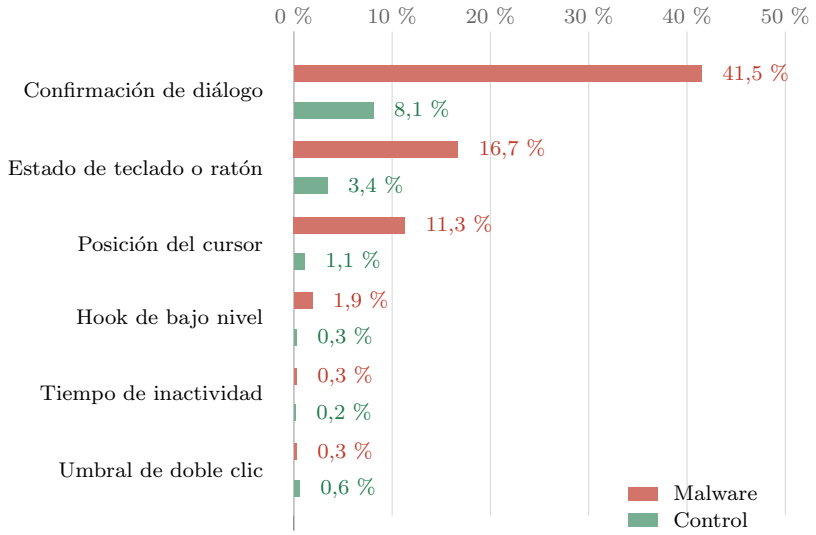

Figura 4.2: Prevalencia de cada mecanismo en el estrato aplicable.

Restringiendo el análisis al estrato aplicable se obtiene la comparación por mecanismo individual, recogida en la Figura 4.2. El reparto entre mecanismos dista mucho de ser uniforme. La confirmación mediante cuadros de diálogo alcanza el 41,5 % y se sitúa muy por encima del resto. Los dos siguientes, el estado de teclado o ratón y la posición del cursor, se quedan en el 16,7 y el 11,3 % respectivamente, mientras que los tres restantes no superan el 2 % en ningún caso.

Al comparar cada mecanismo con su presencia en el grupo de control se obtiene, sin embargo, una ordenación distinta. La mayor diferencia entre ambos grupos corresponde a la posición del cursor, que aparece unas diez veces más en *malware* que en software legítimo. La confirmación de diálogo, pese a ser el mecanismo más frecuente, presenta una diferencia menor, ya que las funciones que emplea también aparecen con cierta frecuencia en programas con interfaz gráfica.

Dos mecanismos se comportan de forma anómala respecto al resto. El tiempo de inactividad presenta prevalencias del 0,3 y el 0,2 % en cada grupo, valores demasiado

próximos como para distinguirlos. El umbral de doble clic llega incluso a detectarse con más frecuencia en el grupo de control que en el experimental.

Por último, el fenómeno tampoco se reparte de forma uniforme entre familias. Sobre aquellas con al menos diez muestras en el estrato aplicable, la prevalencia varía de forma notable, tal como se recoge en la Tabla 4.6.

Familia	**Muestras**	**Con RTT**	**Prevalencia**
Efimer	46	46	100 %
Formbook	13	12	92,3 %
AgentTesla	12	9	75,0 %
LummaStealer	16	6	37,5 %
Vidar	11	1	9,1 %
ValleyRAT	11	0	0 %

Tabla 4.6: Prevalencia por familia en el estrato aplicable, con un mínimo de diez muestras.

El contraste es claro. En Efimer el mecanismo aparece en las 46 muestras analizadas, mientras que en ValleyRAT no aparece en ninguna de las 11. Formbook y AgentTesla presentan también prevalencias altas, mientras que en Vidar solo se detecta en una muestra de 11.

4.5. Análisis de resultados y limitaciones

Conviene precisar el alcance del resultado principal. El 53,7 % corresponde al *malware* que puede analizarse de forma estática, no al *malware* en general. En el estrato no aplicable, la cifra obtenida solo refleja lo que las reglas han podido detectar, que es necesariamente menos de lo que hay, por lo que no debe interpretarse como una medida de prevalencia.

De la comparación por mecanismo se desprenden tres observaciones. La confirmación de diálogo es con diferencia el mecanismo más frecuente y explica 129 de los 167 casos positivos, de modo que el resultado agregado se apoya en buena medida en esa regla. La consulta de la posición del cursor es la que mejor discrimina, con un cociente de 10,4, lo que resulta coherente con su papel central en la literatura y con dos de los cuatro casos de estudio. Por último, en la regla exploratoria del umbral de doble clic se obtiene un cociente de 0,5, es decir, aparece más en software legítimo que en *malware*, con lo que se confirma su falta de poder discriminante y se justifica haberla mantenido fuera del indicador agregado.

Respecto a las limitaciones de la detección estática, la primera limitación a destacar es la ya señalada aplicabilidad. En el 45 % de las muestras analizadas la tabla de

importaciones está prácticamente vacía y las reglas no pueden concluir nada. Esa proporción se corresponde con el estrato no aplicable de la Tabla 4.5, y equivale al 42 % que se observa sobre el corpus completo en la Figura 4.1, calculado antes de descartar las muestras que no llegaron a procesarse.

La segunda limitación se identificó durante la validación contra `Pafish`. En esa herramienta no se emplea la familia de funciones de diálogo estándar para su comprobación de confirmación, sino que se construye una ventana propia mediante funciones gráficas de bajo nivel. Ese mecanismo no resulta detectable mediante reglas basadas en importaciones, dado que dichas funciones están presentes en cualquier programa con interfaz gráfica y su inclusión en la regla la haría inservible por exceso de falsos positivos. La prevalencia del mecanismo de confirmación queda por tanto subestimada en una magnitud desconocida.

La tercera limitación afecta a la cobertura de la propia regla de diálogo. Al revisar qué funciones importaban las muestras detectadas se observó el uso de algunas no contempladas en ella, como `MessageBoxIndirect` o `DialogBoxIndirectParam`. Se trata de funciones equivalentes a las incluidas, pero con una denominación distinta, por lo que no coinciden con ninguna de sus entradas.

Para determinar el alcance de ese hueco se revisaron las muestras del estrato aplicable que la regla no había detectado. En ninguna de ellas se empleaban esas funciones de forma exclusiva, ya que los binarios que las utilizan importan además alguna de las contempladas en la regla y acaban siendo detectados por esa vía. El hueco no ha producido, por tanto, falsos negativos en este corpus.

Las dos limitaciones principales comparten un mismo origen, que es la dependencia del análisis por importaciones. Ambas afectan a muestras que sí pueden ejecutarse, y es precisamente en la ejecución donde la información que el análisis estático no alcanza queda disponible.

Ahí es donde las dos líneas de este trabajo convergen. Las muestras sobre las que el instrumento estático no puede concluir son las que requieren análisis dinámico, y ese análisis solo produce resultados si la carga llega a ejecutarse. Como se ha mostrado, en las muestras con mecanismos RTT esa ejecución no se produce sin una simulación de interacción suficientemente realista. El módulo desarrollado es, por tanto, la condición que hace viable el análisis de ese segmento.

Capítulo 5

Trabajo relacionado

Los mecanismos estudiados en este trabajo no son una novedad reciente. Responden a más de una década de evolución en la que cada avance surge como reacción a la contramedida anterior, y esa trayectoria explica por qué las simulaciones sencillas dejaron de resultar efectivas. En este capítulo se sitúa el trabajo desarrollado respecto a las propuestas que ya abordan el mismo problema. Se recorre en primer lugar la evolución de los mecanismos RTT. A continuación se examinan los trabajos previos sobre simulación de interacción, y el capítulo se cierra delimitando la aportación propia.

5.1. Evolución de los mecanismos RTT

El primer caso ampliamente documentado de mecanismos RTT es `UpClicker` [13]. En esta familia se instala un *hook* de bajo nivel sobre el ratón y se espera el evento de soltado del botón izquierdo. Hasta que ese evento no se produce, el binario permanece por completo inerte, con independencia del tiempo que transcurra. La contramedida fue inmediata y consistió en simular un clic.

`BaneChant` [18] es la respuesta directa a esa contramedida. Se eleva el umbral y se exigen más de tres clics antes de la activación. El patrón que se repite a partir de aquí es que la comprobación no se sustituye, sino que se endurece.

En `Gozi/Ursnif` [20] la idea se lleva a un plano distinto y técnicamente más sólido. En lugar de emplearse la interacción como una condición que se comprueba, se utiliza como material criptográfico. Se calcula el desplazamiento del cursor entre muestras sucesivas y ese valor se emplea como semilla para derivar la clave con la que se descifra la propia carga. Si el ratón no se mueve, el desplazamiento es nulo y la clave no llega a obtenerse. La consecuencia es relevante, ya que no basta con satisfacer una comprobación, sino que la interacción resulta necesaria para que el código malicioso llegue siquiera a existir en memoria.

En `LummaC2 v4.0` [7] se presenta el avance más reciente en el mecanismo basado

en movimiento. No se verifica si el ratón se mueve, sino si el movimiento resulta geométricamente plausible. Se capturan posiciones consecutivas del cursor a intervalos regulares, se calculan los ángulos que forman los vectores entre puntos sucesivos y la ejecución solo continúa si esos ángulos se mantienen por debajo de un umbral. Con ello se descartan los desplazamientos bruscos que produce una simulación elemental.

Más allá del ratón, en otras familias se emplean variantes igualmente eficaces. En `Dridex` se recurre a una macro que se ejecuta al cerrar el documento en lugar de al abrirlo [21], con lo que un entorno de análisis que finalice sin cerrar el fichero no llega a ejecutar la carga maliciosa. En `Agent Tesla` [12] se consulta el tiempo transcurrido desde la última entrada del usuario y se suspende la actividad si supera un umbral específico.

5.2. Simulación de interacción

MORRIGU [6] es el trabajo más cercano en planteamiento. Es una herramienta de reconfiguración dinámica de la *sandbox* con la que se prueban de forma sistemática cinco métodos anti-evasión, entre ellos un test de Turing inverso con el que se simula interacción del usuario. Su simulación interpola entre posiciones para generar movimiento continuo, varía la velocidad del cursor, aleatoriza las pausas e incorpora apertura de aplicaciones y escritura de teclado.

Los resultados de ese trabajo resultan además pertinentes para el presente. Sobre un conjunto de 251 muestras se identificaron 40 con comportamiento evasivo, de las cuales 18 se activaron mediante el test de Turing inverso, que resultó ser el disparador individual más frecuente de los cinco evaluados [6]. Esa medida y la obtenida en el estudio de prevalencia de este trabajo responden a preguntas distintas, ya que la suya se refiere a la frecuencia del disparador dentro de un conjunto ya identificado como evasivo. Aun así, ambas apuntan en la misma dirección al situar este mecanismo entre los más relevantes. Sus autores señalan asimismo la inexistencia de un conjunto público de *malware* evasivo.

En UBER [22] se adopta una estrategia diferente y se generan artefactos de uso realistas en el sistema, como historial y ficheros recientes, para que la máquina no presente el aspecto de una instalación recién creada. La diferencia esencial es el momento de actuación, ya que en UBER se prepara el entorno antes del análisis mientras que en este trabajo se simula la interacción durante la ejecución. Ambos enfoques resultan complementarios, puesto que los mecanismos basados en actividad en tiempo real no se cubren preparando el entorno de antemano.

En el trabajo de Check Point Research [8] se aporta el resultado más incómodo

para cualquier simulación. Se demuestra que los movimientos de ratón generados por las simulaciones existentes resultan estadísticamente distinguibles de los humanos, de modo que pueden detectarse mediante el análisis de su distribución. Con ello se justifica que la simulación no se limite a desplazar el cursor, sino que se reproduzcan propiedades del movimiento humano real.

No ha sido posible, sin embargo, evaluar el módulo desarrollado frente a este tipo de detección, ya que el detector estadístico descrito no se publica. Sus autores justifican esa decisión de forma expresa, al señalar que no pretenden que el trabajo constituya una guía para evadir *sandboxes* con éxito, por lo que omiten deliberadamente los detalles concretos del método. La comprobación de si el movimiento generado resiste un análisis estadístico de esta naturaleza queda planteada como trabajo futuro.

5.3. Aportación del presente trabajo

La aportación de este trabajo se sitúa en tres planos. El primero es el de la simulación. El módulo desarrollado reproduce propiedades del movimiento humano que las comprobaciones más recientes verifican de forma explícita, como el perfil de velocidad de cada trazo y la continuidad de rumbo entre trazos consecutivos, y cubre además los mecanismos basados en confirmación mediante cuadros de diálogo, que resultaron ser los más frecuentes del estudio de prevalencia. Con ello se superan comprobaciones documentadas en la literatura como no resueltas por las simulaciones existentes, entre ellas la verificación geométrica de `LummaC2 v4.0`, aparecida con posterioridad a los trabajos revisados.

El segundo es el de la medición. Se aporta una estimación de la proporción de *malware* que incorpora estos mecanismos, obtenida sobre un corpus general de muestras recientes y contrastada frente a un grupo de control de software legítimo. Los trabajos previos abordan una pregunta distinta, ya que en ellos se mide la frecuencia de los disparadores dentro de conjuntos ya identificados como evasivos.

El tercero es el propio conjunto de muestras. Durante el trabajo se ha construido un corpus de *malware* con mecanismos RTT confirmados, obtenido a partir de lotes públicos y etiquetado según el mecanismo que implementa cada muestra.

Capítulo 6

Conclusiones y trabajo futuro

En este trabajo se han abordado los mecanismos de comprobación de presencia humana que el *malware* emplea para evadir el análisis dinámico. El problema se ha tratado desde dos vertientes complementarias. Por un lado, se ha caracterizado el fenómeno y se ha medido su prevalencia sobre un corpus de muestras recientes, con el resultado de que algo más de la mitad del *malware* analizable de forma estática incorpora alguno de estos mecanismos, frente a aproximadamente una décima parte del software legítimo comparable. Por otro, se ha desarrollado un módulo de simulación de interacción para CAPEv2 con el que se superan esas comprobaciones, validado sobre cuatro casos de estudio de naturaleza distinta.

El módulo desarrollado mejora al que se incorpora por defecto en CAPEv2 en cuatro aspectos. Se reproduce el movimiento humano modelando su perfil de velocidad y la duración de cada trazo, en lugar de generar desplazamientos aleatorios. Se acota el cambio de dirección entre trazos, con lo que toda trayectoria cumple por construcción el requisito geométrico y la activación deja de depender del azar. Los clics se generan como eventos físicos de entrada, condición necesaria para las comprobaciones que se apoyan en *hooks* de bajo nivel. Además, se amplía la cobertura a los mecanismos basados en confirmación mediante cuadros de diálogo, que en el módulo original no se contemplan.

Adicionalmente, en este trabajo se aporta un conjunto de siete reglas capa con las que se detectan estos mecanismos, aplicables tanto al análisis estático como al dinámico sin modificación alguna, y una medida de prevalencia contrastada frente a un grupo de control de software legítimo. Su aplicación por la vía dinámica reveló además un hueco de cobertura en la instrumentación de CAPEv2, del que se ha corregido la parte que no requería recompilar el monitor. Todo el material desarrollado se ha publicado en un repositorio de acceso público.

Ese hueco marca la continuación natural del trabajo. Las funciones de confirmación de diálogo carecen de interceptación en el monitor de CAPEv2, por lo que su cobertura

exige modificar el componente escrito en C y volver a compilarlo. La siguiente línea de trabajo futuro consiste en repetir el estudio de prevalencia por la vía dinámica, ejecutando las muestras con el módulo activado.

Bibliografía

[1] Amir Afianian, Salman Niksefat, Babak Sadeghiyan, and David Baptiste. Malware Dynamic Analysis Evasion Techniques: A Survey. *ACM Comput. Surv.*, 52(6):1–28, 2019. Preprint disponible en arXiv:1811.01190. URL: https://doi.org/10.1145/3365001.

[2] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A Survey on Automated Dynamic Malware-Analysis Techniques and Tools. *ACM Computing Surveys*, 44(2):6:1–6:42, 2012. URL: https://doi.org/10.1145/2089125.2089126.

[3] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of Static Analysis for Malware Detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)*, pages 421–430, Miami Beach, FL, USA, 2007. IEEE. URL: https://doi.org/10.1109/ACSAC.2007.21.

[4] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. Dynamic Malware Analysis in the Modern Era: A State of the Art Survey. *ACM Computing Surveys*, 52(5):1–48, 2019. URL: https://doi.org/10.1145/3329786.

[5] Alexei Bulazel and Bülent Yener. A Survey on Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web. In *Proceedings of the 1st Reversing and Offensive-Oriented Trends Symposium (ROOTS)*, Vienna, Austria, 2017. ACM. URL: https://doi.org/10.1145/3150376.3150378.

[6] Alan Mills and Phil Legg. Investigating Anti-Evasion Malware Triggers Using Automated Sandbox Reconfiguration Techniques. *Journal of Cybersecurity and Privacy*, 1(1):19–39, 2021. Herramienta MORRIGU. URL: https://www.mdpi.com/2624-800X/1/1/3.

[7] Outpost24. LummaC2 Anti-Sandbox Technique: Trigonometry for Human Detection, 2023. Accedido el 10 de junio de 2026. URL: https://outpost24.com/blog/lummac2-anti-sandbox-technique-trigonometry-human-detection/.

[8] Check Point Research. The Cat and Mouse Game: Exploiting Statistical Weaknesses in Human Interaction Anti-Evasions, 2025. Accedido el 10 de junio de 2026. URL: https://research.checkpoint.com/2025/the-cat-and-mouse-game-exploiting-statistical-weaknesses-in-human-interaction-anti-evasions/.

[9] Microsoft. PE Format Specification, 2024. Accedido el 8 de abril de 2026. URL: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

[10] Kevin O'Reilly. CAPE: Malware Configuration And Payload Extraction, 2025. Accedido el 15 de abril de 2026. URL: https://github.com/kevoreilly/CAPEv2.

[11] Alberto Ortega. Pafish: Paranoid Fish, 2019. Accedido el 2 de julio de 2026. URL: https://github.com/a0rtega/pafish.

[12] VMRay. Threat Bulletin: Exploring the Differences and Similarities of Agent Tesla v2 and v3, 2024. Accedido el 18 de junio de 2026. URL: https://www.vmray.com/threat-bulletin-exploring-the-differences-and-similarities-of-agent-tesla-v2-v3-2/.

[13] Volatility Labs. What Do UpClicker, Poison Ivy, Cuckoo and Volatility Have in Common?, 2012. Accedido el 14 de mayo de 2026. URL: https://volatility-labs.blogspot.com/2012/12/what-do-upclicker-poison-ivy-cuckoo-and.html.

[14] Andreas Plesner, Tobias Vontobel, and Roger Wattenhofer. Breaking reCAPTCHAv2, 2024. Accedido el 9 de julio de 2026. URL: https://arxiv.org/abs/2409.08831, arXiv:2409.08831.

[15] Tamar Flash and Neville Hogan. The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model. *The Journal of Neuroscience*, 5(7):1688–1703, 1985. URL: https://doi.org/10.1523/JNEUROSCI.05-07-01688.1985.

[16] Paul M. Fitts. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954. URL: https://doi.org/10.1037/h0055392.

[17] Mandiant. capa: Automatically Identify Malware Capabilities, 2025. Accedido el 2 de julio de 2026. URL: https://github.com/mandiant/capa.

[18] SecurityWeek. APT Malware Counts Mouse Clicks to Evade Researchers, 2013. Accedido el 14 de mayo de 2026. URL: https://www.securityweek.com/apt-malware-counts-mouse-clicks-evade-researchers.

[19] abuse.ch. MalwareBazaar, 2026. Accedido el 16 de julio de 2026. URL: https://bazaar.abuse.ch/.

[20] Joe Security. Analysis of Gozi/Ursnif Mouse Movement Evasion, 2017. Accedido el 21 de mayo de 2026. URL: https://www.joesecurity.org/blog/5852460122427342172.

[21] Proofpoint. Run-on-Close Macros Try to Shut the Door on Sandboxes, 2015. Accedido el 21 de mayo de 2026. URL: https://www.proofpoint.com/us/threat-insight/post/Run-on-Close-Macros-Try-to-Shut-the-Door-on-Sandboxes.

[22] Pengbin Feng, Jianhua Sun, Songsong Liu, and Kun Sun. UBER: Combating Sandbox Evasion via User Behavior Emulators. In *Information and Communications Security (ICICS 2019)*, pages 34–50. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-41579-2_3.

Anexos

Apéndice A

Dedicación y planificación temporal

En este anexo se recoge la dedicación empleada en el desarrollo del trabajo, así como su distribución a lo largo del tiempo.

La dedicación total asciende a 316 horas. De ellas, 266 corresponden al registro diario mantenido desde el 27 de mayo de 2026, y las 50 restantes al trabajo realizado durante el segundo semestre del curso académico. Ese trabajo previo comprende la instalación y configuración de CAPEv2 en el equipo de desarrollo, junto con los primeros módulos de prueba para la observación de llamadas a la API y árboles de ejecución.

El reparto por tipo de tarea se recoge en la Tabla A.1.

Tarea	**Horas**	**Porcentaje**
Desarrollo del módulo de simulación	82,0	25,9 %
Desarrollo de reglas y estudio de prevalencia	66,0	20,9 %
Análisis de malware y construcción del corpus	54,0	17,1 %
Trabajo previo del segundo semestre	50,0	15,8 %
Redacción de la memoria	42,0	13,3 %
Documentación continua	16,0	5,1 %
Preparación del entorno de análisis	6,0	1,9 %
Total	**316,0**	**100 %**

Tabla A.1: Dedicación por tipo de tarea.

La distribución temporal se recoge en la Figura A.1.

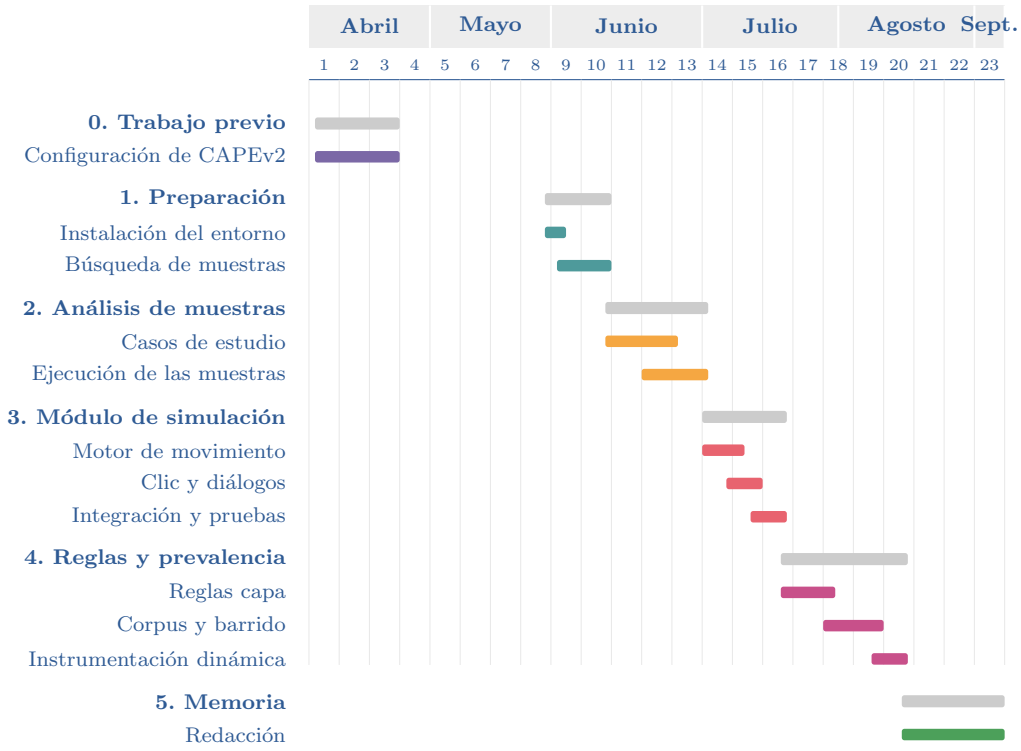

Figura A.1: Distribución temporal de las fases del trabajo y sus principales tareas.

Apéndice B

Contenido del repositorio

El material desarrollado se publica en `https://github.com/Nicofaroo/CAPEv2-Trigger-Detection`. Al tratarse de una bifurcación del proyecto CAPEv2, la mayor parte de los ficheros del repositorio pertenecen al proyecto original. Las aportaciones propias de este trabajo se localizan en las rutas que se indican a continuación.

Módulo de simulación `analyzer/windows/modules/auxiliary/human_improved.py`
: Módulo auxiliar de simulación de interacción humana descrito en la Sección 3.3. En el mismo directorio se incluye `human_robotic.py`, el módulo de movimiento mecánico empleado como condición de control en la validación.

Reglas capa `capa-rules/`
: Los siete ficheros de reglas descritos en la Sección 3.4, en formato YAML.

Instrumentación dinámica `lib/cuckoo/common/logtbl.py`
: Tabla de traducción de CAPEv2, con las entradas añadidas para las funciones descritas en la Sección 3.5.

Estudio de prevalencia `capa-rules/estudio-prevalencia/`
: Scripts que implementan el procedimiento completo, desde la construcción del corpus hasta la generación de los resultados.

Corpus `dataset/`
: Listado de identificadores SHA-256 de las muestras empleadas, junto con su familia.