

**Trabajo Final de Grado  
Ingeniería en Informática**

# **Verificación de transparencia de entornos de análisis dinámico de malware**

**Autor:** Braulio Leonel Pablos Di Marco

**Director:** Felipe Evans



**2020**





RINFI se desarrolla en forma conjunta entre el INTEMA y la Biblioteca de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata.

Tiene como objetivo recopilar, organizar, gestionar, difundir y preservar documentos digitales en Ingeniería, Ciencia y Tecnología de Materiales y Ciencias Afines.

A través del Acceso Abierto, se pretende aumentar la visibilidad y el impacto de los resultados de la investigación, asumiendo las políticas y cumpliendo con los protocolos y estándares internacionales para la interoperabilidad entre repositorios



Esta obra está bajo una [Licencia Creative Commons Atribución-  
NoComercial-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-nc-sa/4.0/).

## **Agradecimientos**

*A mi familia, por su apoyo incondicional.*

*A mi director (a quien espero poder decirle colega pronto), siempre bien predispuesto e irradiando pasión. Es inmensurable lo mucho que he aprendido de la disciplina y de la profesión por su causa.*

*A los amigos que he hecho en la facultad, compañeros de aventuras dentro y fuera de las aulas quienes se han convertido en una parte irremplazable de mi vida. Ha sido un placer compartir esta odisea con ellos.*

*A los amigos con los que vengo compartiendo vida desde antes. Me alegro de poder disfrutar el cierre de otra etapa a su lado y espero que sea una más de muchas por venir.*

*A los profesores que tuve a lo largo de la carrera, por haberme brindado las herramientas necesarias para llegar hasta aquí.*

*A la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata y a la educación pública en general, por haberme permitido seguir esta carrera que tanto amo.*

## Índice

Capítulo 1: Introducción.....	5
1.1 Objetivos.....	6
Capítulo 2: Marco teórico.....	7
2.1 Análisis estático y análisis dinámico.....	7
2.2 Malware que cambia su comportamiento en base al contexto.....	8
2.3 Definición de transparencia.....	9
2.4 Clasificación de técnicas de evasión de análisis dinámico mediante reconocimiento de entornos.....	11
2.4.1 Clasificación de técnicas según el sistema operativo destino.....	12
2.4.2 Clasificación de técnicas según su meta.....	14
2.4.3 Clasificación de técnicas según la metodología aplicada.....	20
2.4.3.1 Técnicas que recopilan información brindada por el entorno.....	22
2.4.3.2 Técnicas que evalúan el comportamiento del entorno.....	22
2.4.3.3 Técnicas basadas en la relación de un proceso con el entorno.....	24
2.5 Recopilación de técnicas de evasión de análisis dinámico de malware basadas en el reconocimiento de entornos.....	26
2.6 Conclusiones.....	26
Capítulo 3: Requerimientos del mercado.....	28
3.1 Segmentos objetivo.....	28
3.2 Requerimientos del producto.....	29
3.3 Productos existentes.....	29
3.4 Visión del producto.....	34
Capítulo 4: Arquitectura seleccionada.....	36
4.1 Esquema de seguridad.....	40
4.1.1 SECCHIWARE-HMAC-256.....	40
4.2 Atributos de calidad.....	41
Capítulo 5: Implementación y distribución.....	43
5.1 Lenguaje de programación.....	43
5.2 Back-end web.....	48
5.3 Base de datos.....	49
5.4 Implementación, ejecución y almacenamiento de tests.....	50
5.5 Caché.....	52
5.6 Control de concurrencia.....	54
5.7 Cliente por línea de comandos.....	60
5.8 Cliente gráfico.....	60
5.8.1 Pantallas.....	61
5.8.1.1 Bienvenida.....	61
5.8.1.2 Gestión del servidor de comando y control.....	62
5.8.1.3 Entornos.....	63
5.8.1.4 Búsqueda y consulta de sesiones.....	65
5.9 Selección e implementación de pruebas.....	67
5.10 Distribución.....	69
Capítulo 6: Verificación de transparencia de entornos selectos.....	71
Capítulo 7: Gestión del proyecto.....	74
7.1 Proceso de desarrollo.....	74
7.2 Comparativa entre la planificación original y la ejecución concretada del proyecto.....	78

---

7.3 Lecciones aprendidas.....	80
Capítulo 8: Conclusiones.....	82
8.1 Trabajos futuros.....	83
8.1.1 Mejoras a incorporar.....	83
8.1.2 Expansión comercial.....	85
Bibliografía.....	86
Apéndice A - Compendio de técnicas de anti-análisis basadas en el reconocimiento del entorno.....	93
A.1. Bibliografía.....	120
Apéndice B - APIs.....	122
B.1. Esquemas JSON.....	122
B.2. Endpoints de un nodo.....	126
B.2.1. DELETE /.....	126
B.2.2. GET /reports.....	126
B.2.3. GET /test_sets.....	127
B.2.4. PATCH /test_sets.....	127
B.2.5. DELETE /test_sets/{package}.....	128
B.3. Endpoints de un servidor de comando y control.....	129
B.3.1. GET /environments.....	129
B.3.2. POST /environments.....	129
B.3.3. DELETE /environments/{ip}/{port}.....	130
B.3.4. GET /environments/{ip}/{port}/info.....	131
B.3.5. GET /environments/{ip}/{port}/installed.....	132
B.3.6. PATCH /environments/{ip}/{port}/installed.....	132
B.3.7. DELETE /environments/{ip}/{port}/installed/{package}.....	133
B.3.8. GET /environments/{ip}/{port}/reports.....	134
B.3.9. GET /executions.....	135
B.3.10. DELETE /executions/{execution_id}.....	137
B.3.11. GET /sessions.....	137
B.3.12. GET /sessions/{session_id}.....	139
B.3.13. DELETE /sessions/{session_id}.....	139
B.3.14. GET /test_sets.....	140
B.3.15. PATCH /test_sets.....	140
B.3.16. DELETE /test_sets/{package}.....	141
Apéndice C - Comandos del cliente CLI.....	143
C.1. Argumentos comunes.....	143
C.2. Opciones comunes.....	143
C.3. Comandos.....	143
Apéndice D - Verificación de transparencia de entornos selectos.....	148
D.1. Entornos.....	148
D.2. Resultados de las pruebas.....	149
Apéndice E - Diagramas de evolución temporal del proyecto.....	152
E.1. Gantt de la planificación teórica.....	152
E.2. Gantt de la evolución real del proyecto.....	153

# Capítulo 1: Introducción

El análisis de malware es un proceso por el cual se estudian muestras de software malintencionado para entender su funcionalidad y propósito. Su fin último es el de obtener información suficiente para elaborar mecanismos de detección de estos agentes, mitigar su accionar, remover su presencia de sistemas infectados y reparar los daños que hayan podido efectuar.

En principio, los investigadores en seguridad informática comenzaron a utilizar técnicas de análisis estático para observar la composición de archivos ejecutables en busca de indicios de malware y para entender su implementación. Sin embargo, los maleantes encontraron formas de dificultar la lectura del código binario de sus programas, mermando así la efectividad de esta vía [3].

Como respuesta a esta situación, directamente se empezó a ejecutar el software maligno y supervisar su accionar, lo que se conoce como análisis dinámico. A través de este, el comportamiento de una muestra se describe mediante la observación de su interacción con el sistema y los cambios que en él produce.

Dado que ejecutar malware en un equipo computacional desprotegido supone un riesgo muy elevado, para poder llevar adelante análisis dinámico con tranquilidad se difundió el uso de sandboxes. Un sandbox es un entorno controlado que sirve para aislar en su interior a un proceso y de esta forma evitar poner en riesgo al resto de un sistema computacional [52]. Ya desde su construcción estas herramientas presentan diferencias de mayor o menor magnitud con los ambientes informáticos “reales” con los cuales se trabaja en el día a día.

Desafortunadamente los atacantes también idearon estrategias para hacerle frente a esta metodología, de entre las cuales se encuentra la aplicación de técnicas de reconocimiento del entorno. Mediante aquellas el malware es capaz de encontrar las diferencias previamente mencionadas, asumir que está siendo estudiado y en consecuencia puede decidir finalizar su ejecución abruptamente o mostrar un comportamiento benévolo. Por citar algunos ejemplos, hay procesos malignos que determinan que están siendo analizados si encuentran artefactos de virtualización en el sistema sobre el cual se están ejecutando, otros buscan anomalías en el comportamiento del ambiente y hay algunos cuyo criterio diferenciador es la existencia de interacción humana.

Esto no solo supone un impedimento en el correcto estudio de procesos malintencionados, sino que además es un fuerte golpe contra la efectividad de herramientas que también aplican análisis dinámico con fines defensivos, como los antivirus o los honeypots. Como medida para contrarrestar esta situación, dentro del ámbito de la seguridad informática se ha propuesto la elaboración de ambientes más transparentes [10].

Esta tarea no es fácil: los sistemas computacionales presentan un conjunto virtualmente ilimitado de variantes tanto en su composición como en su comportamiento que los hacen muy difíciles de replicar fielmente, por no decir que es una tarea imposible. Las discrepancias existentes entre entornos son la base detrás de sus técnicas de identificación y por ello son tan numerosas como aquellas.

Si bien ya hay disponibles algunos instrumentos en el mercado que buscan dar respuesta a esta problemática mediante la evaluación de la transparencia de entornos, el alcance de esta se encuentra sumamente limitado, tanto por las características que efectivamente son indagadas como por las plataformas en las que pueden operar. Particularmente los sistemas operativos basados en Linux, como Android y los presentes en la mayoría de los servidores del mundo, han sido ignorados por los desarrolladores de estas herramientas. Ya sea por su popularidad o por la cantidad de información que gestionan, estos sistemas presentan un riesgo lo suficientemente importante que justifica destinar esfuerzos para mejorar también el análisis de malware que los tiene como objetivo.

## 1.1 Objetivos

El objetivo principal de este proyecto es el diseño y construcción de un prototipo de verificador de transparencia multiplataforma cuyo fin es el de asistir a la comunidad de la ciberseguridad en la preparación de mejores entornos destinados al análisis dinámico de malware.

Para alcanzar este propósito se establecieron los siguientes objetivos específicos:

- Revisar la literatura existente referida a la elusión de análisis dinámico por parte de malware para comprender el dominio de este problema.
- Elaborar una clasificación de técnicas para identificar ambientes de análisis dinámico de malware para comprender el alcance de estas y los conceptos que las sustentan.
- Elaborar un compendio de técnicas de reconocimiento de entornos debidamente categorizadas en base a la clasificación previa.
- Analizar el estado del mercado de los verificadores de transparencia de entornos, identificar sus requerimientos e investigar productos semejantes disponibles.
- Diseñar e implementar un prototipo de verificador de transparencia de entornos de análisis dinámico de malware que satisfaga los requerimientos observados.
- Seleccionar e implementar un conjunto representativo de técnicas del compendio elaborado como pruebas de transparencia para verificar el producto desarrollado y validar los conceptos teóricos tras la bibliografía consultada.
- Distribuir el sistema implementado como software libre y gratuito.

## Capítulo 2: Marco teórico

Malware es una contracción del inglés *malicious software*; se define como software cuya intención es sacar provecho de un sistema computacional o ejercer un daño sobre él [36]. Su existencia ha supuesto un grave problema a la ciberseguridad y ha sido el impulsor de muchas decisiones de diseño aplicadas a todo tipo de sistemas a nivel mundial.

La diversidad y volumen de estas entidades malignas ha ido de la mano con la creciente expansión tecnológica que ha experimentado el planeta desde que surgieron el siglo pasado. Su expansión ha alcanzado un punto tal que la mayoría del malware reconocido en el año 2019 son amenazas de día cero [45] (esto quiere decir que explotan vulnerabilidades que aún no fueron percatadas por el público y para las cuales todavía no existen mecanismos de defensa). Aquello indica una tendencia en la cual cada nueva muestra de software malicioso encontrada tiene una alta probabilidad de ser una variante hasta el momento desconocida.

Con el paso de los años surgieron muchos tipos de herramientas para identificar y estudiar programas malintencionados con el propósito último de mitigarlos, tales como los tan conocidos antivirus. Resulta de especial interés explorar las dos grandes metodologías que existen para llevar esto a cabo: el análisis estático y el análisis dinámico.

### 2.1 Análisis estático y análisis dinámico

Afianian et al. [3] explican que, en primera instancia, los investigadores en seguridad informática comenzaron a aplicar lo que se conoce como análisis estático. Este hace uso de herramientas tales como desensambladores para estudiar la composición de los archivos ejecutables portadores de malware.

De lo que en principio era una tarea manual se derivó una forma automatizada de aplicar análisis estático que se popularizó entre los antivirus: la identificación de firmas (patrones comunes característicos de una familia de software maligno) en un binario. Esta estrategia se respalda en el mantenimiento de una base de datos de firmas contra la cual se compara el archivo bajo estudio.

He aquí la gran falencia de este sistema: depende de que se haya reconocido con anterioridad la variedad de malware bajo tratamiento para poder reconocer muestras subsecuentes. Si se tiene en consideración el panorama actual en el cual la mayoría de las muestras analizadas corresponden a nuevas amenazas, claramente la efectividad de la revisión estática se ha visto severamente mermada. A su vez, los maleantes comenzaron a aplicar tácticas de ofuscación de código y otro tipo de medidas que dificultan la comprensión de los programas elaborados, lo que ha perjudicado aún más al análisis estático.



Como respuesta a estas limitaciones surgió el análisis dinámico. En este cambia el foco del estudio: lo que trata de caracterizar es el comportamiento real del ejemplar observado. Para ello el programa es efectivamente iniciado y su ejecución es seguida gracias a instrumentos como depuradores, rastreadores o monitores de red.

Debido al riesgo que supone dejar que un malware opere libremente en un sistema computacional, se empezaron a construir entornos que confinan y aíslan a un proceso en su interior. Estos ambientes cerrados se conocen popularmente como sandboxes, los cuales pueden ser contruidos con diversas tecnologías, de las cuales la más popular ha sido la virtualización. Su uso se ha difundido en gran medida para la realización de análisis dinámico automatizado, el cual ha ayudado a estudiar grandes volúmenes de ejemplares en cortos periodos de tiempo.

Sin embargo, los atacantes también encontraron formas de hacerle frente a la revisión dinámica, entre las cuales se encuentra la aplicación de técnicas de reconocimiento de entornos.

## 2.2 Malware que cambia su comportamiento en base al contexto

Tal como mencionan Kedrowitsch et al. [44] aproximadamente un 40% del malware en circulación se ejecuta distinto bajo la presencia de virtualización y herramientas de análisis. También influye en esta diferencia de comportamiento la presencia de huellas en el sistema que denotan su uso por parte de un ser humano. Esta variante de software suele presentar actividad benévola en caso de determinar que está siendo activamente analizado e, incluso, puede detener tempranamente su ejecución de ser ese el caso.

Lo que este tipo de programas realiza es una discriminación de entornos que separa aquellos que forman parte de su objetivo, los cuales desea atacar, y aquellos que no, a los cuales ignora o de los que se esconde activamente por temor a tratarse de ambientes de revisión. Esta distinción puede realizarse de dos maneras. Se va a llamar  $U$  al universo de todos los entornos posibles,  $O$  al conjunto de entornos que representan el objetivo de un programa dañino particular y  $R$  al grupo de entornos rechazadas por aquel. Estos conjuntos cumplen las siguientes propiedades:

- $U = O \cup R$
- $\emptyset = O \cap R$

En caso de que un malware aplique técnicas de reconocimiento buscando explícitamente características presentes en su entorno objetivo, se puede decir entonces que usa un criterio de aceptación. Aquí  $O$  está bien definido y el conjunto de rechazo se define como:

$$R = U - O$$

en donde  $R$  incluye todo ambiente que no presente las propiedades esperadas. En términos simples, el programa tiene en claro lo que está tratando de atacar y esquivo todo lo demás.

En cambio, si se trata de identificar particularidades no deseadas en un entorno, se está utilizando un criterio de rechazo. Por ende,  $R$  está especificado debidamente y  $O$  surge de:

$$O = U - R$$

Entonces aquí lo que se conoce bien es lo que se está evitando y lo que se intenta dañar es todo lo que no se pueda identificar como parte del primer grupo.

Sea cual sea la política aplicada para diferenciar entornos, esto supone un drástico impedimento para la correcta realización de análisis dinámico. Aquellos sandboxes que no cumplan con las condiciones necesarias para formar parte del objetivo de un proceso malicioso dan lugar a falsos negativos o, en caso de que se pueda identificar de alguna forma que se está en presencia de malware, su entendimiento se ve severamente acotado. A día de hoy no se ha encontrado una solución completamente efectiva para esta situación.

He aquí entonces la problemática que da lugar a este trabajo: ¿de qué forma se pueden mejorar los entornos destinados a análisis dinámico de sistemas malignos con el fin de que esta tarea pueda ser realizada correctamente? Bulazel et al. [10] agrupan en su trabajo las metodologías que han surgido para tratar comportamientos evasivos en dos categorías: las activas, que involucran su detección y consecuente mitigación; y las pasivas, que se basan en fortalecer la transparencia de los entornos de análisis. A su vez destacan la prevalencia de estas últimas sobre las activas, lo cual atribuyen a que los métodos pasivos escalan mejor y no son inherentemente vulnerables a las técnicas de ofuscación de rutas.

Queda claro entonces que el camino a seguir más económico y robusto es la elaboración de ambientes más transparentes. En particular, la solución que se busca ofrecer en este proyecto es un verificador de transparencia de entornos para análisis dinámico de malware. Es por ello que se debe definir adecuadamente qué es en sí la transparencia en este ámbito.

## 2.3 Definición de transparencia

La popularidad que han tomado los sandboxes implementados mediante mecanismos de virtualización ha llevado a entender colectivamente a la transparencia como el grado de similitud de un ambiente virtualizado frente a uno físico.

Esta concepción presenta algunos inconvenientes. El primero de ellos es que las herramientas de virtualización basadas en hipervisores (tales como VMWare o VirtualBox) sacrifican la transparencia en sus diseños en pos de la compatibilidad con diversos sistemas operativos y arquitecturas y de la velocidad de ejecución [31]. A su vez, aquellas

que aplican emulación completa sacrifican rendimiento, lo cual también termina marcando una diferencia con los entornos de ejecución físicos. Por otra parte, el creciente uso de máquinas virtuales en servidores ha provocado que se conviertan en objetivo para los atacantes, quienes además de poder recuperar información de ellas podrían aprovechar alguna vulnerabilidad para escapar y comprometer al host [73].

También se han desarrollado sandboxes en bare-metal, en donde los sistemas a analizar son ejecutados bajo ningún tipo de infraestructura de virtualización [46, 47]. Esta modalidad es aún vulnerable a otras técnicas para reconocer un entorno de análisis que no se basan en identificar virtualización y que tratan cuestiones tales como la existencia de interacción humana, la presencia de archivos y servicios que caracterizan a un servidor particular o la ejecución de herramientas de monitoreo.

Resulta imprescindible entonces hallar una definición de transparencia más acorde a la problemática presentada. Bulazel et al. [10] la describen como “la propiedad de los sistemas de análisis que los hacen indiscernibles de sistemas no destinados a dicho fin, incluso para un adversario especializado”.

Tomando esta descripción como base, en este trabajo se propone que la transparencia es la propiedad de un entorno de análisis dinámico de software que lo hace indistinguible para un malware particular de aquellos entornos que forman parte del objetivo de este último.

A partir de esto se puede concluir lo siguiente:

- Un entorno de análisis dinámico de malware es transparente para un software malicioso determinado si y solo si el mismo no se resiste a llevar adelante su acción dañina dentro de dicho ambiente.
- A un entorno de análisis le basta solo con responder adecuadamente a las técnicas de reconocimiento de un software maligno en especial para ser transparente frente a él.
- La transparencia de un entorno es relativa al malware bajo estudio. La diversidad de metodologías conocidas y por conocer para detectar ambientes de análisis, como a su vez las innumerables características individuales que se presentan en los sistemas objetivos de ataques sugieren que cualquier interpretación absoluta de transparencia no es de valor práctico en la búsqueda por conformar mejores sandboxes.

Tras definir lo que es transparencia, es de interés destacar aquellas propiedades de un entorno que influyen en la misma y por ende en la construcción de ambientes de análisis robustos. A continuación se presentan los atributos de un entorno sobre las cuales se sustenta la transparencia, determinadas a partir de la investigación realizada:

- **Atributos informados por el mismo entorno:** este pilar engloba aspectos del ambiente o de los elementos que allí residen que un proceso puede consultar, tales

como el modelo y cantidad de núcleos del procesador, la cantidad de memoria volátil y persistente disponible, los procesos en ejecución, los archivos presentes, los dispositivos conectados, etc.

- **El comportamiento del entorno:** la información aquí abarcada no es preguntada directamente al sistema como en el ítem anterior, sino que es deducida a partir de la observación activa del entorno. Involucra cuestiones tales como el tiempo de procesamiento o los resultados obtenidos tras la ejecución de ciertas instrucciones.
- **La relación entre el entorno y el proceso:** alcanza a las características de un entorno requeridas por un software determinado para poder desenvolverse adecuadamente en el mismo; también incluye a las propiedades que el sistema le asigna a un proceso al momento de su creación y la manera en que es gestionado mientras dure su ejecución. Esto, además de influir en cómo el proceso interactúa con el ambiente, también influye en cómo se relaciona con otros procesos y sistemas externos. Reúne aspectos tales como los privilegios asignados, las librerías de software presentes o la conectividad en red habilitada.

## 2.4 Clasificación de técnicas de evasión de análisis dinámico mediante reconocimiento de entornos

Para facilitar la comprensión de los conceptos generales detrás de la operatoria de las técnicas aplicables por programas maliciosos para evitar ser estudiados, es imprescindible clasificarlas bajo algún criterio que destaque claramente sus características comunes. Esto no es una tarea fácil: los ambientes reales (o no destinados al análisis) presentan un sin fin de variables en su comportamiento, configuración y constitución que conllevan a que la tarea de confeccionar un entorno de observación que los replique fielmente sea prácticamente imposible, más aún si se considera que dicha fidelidad puede ser contraria a su capacidad de análisis y/o confinamiento [31]. Consecuentemente, las técnicas para identificar sandboxes derivan en un ejercicio de creatividad por el cual se encuentran diferencias entre los sistemas reales y los destinados a la investigación.

Aún así, existen diversos artículos en los cuales ya han quedado plasmados esfuerzos por categorizar estas técnicas [3, 5, 10, 36, 60]. Los criterios utilizados para elaborar las clasificaciones suelen mezclar la meta de la técnica en cuestión (por ejemplo, detectar si se está en presencia de un sistema virtualizado) con la metodología general aplicada (identificar dispositivos conectados, capturar eventos del sistema, etc), lo que conlleva a que los límites de las categorías propuestas resulten difusos.

Esta situación derivó en la elaboración de clasificaciones propias en un intento de organizar más adecuadamente los conceptos detrás de las técnicas de reconocimiento, lo cual a su vez es de utilidad a la hora de establecer el rango de capacidades que debería presentar un verificador de transparencia de entornos. Se han establecido entonces los siguientes criterios de clasificación:

- Según el sistema operativo destino
- Según la meta de la técnica
- Según la metodología aplicada

### **2.4.1 Clasificación de técnicas según el sistema operativo destino**

Este primer agrupamiento tiene como objetivo discriminar los métodos para evadir análisis dinámico tomando en consideración las diferencias presentes en la pluralidad de sistemas operativos donde son aplicados. La disparidad que puede apreciarse en el diseño, configuración y comportamiento de estos productos deriva en que aunque dos posibles técnicas apliquen la misma metodología general en dos sistemas distintos, la manera de proceder en cada uno de ellos puede ser drásticamente distinta; a su vez, existen estrategias que solo pueden ser aplicadas en algún sistema particular.

El alcance de este trabajo abarca tres sistemas operativos popularmente conocidos, los cuales se detallan a continuación. Esta selección se fundamenta en su adopción general, los estudios disponibles referidos a la materia y en la magnitud del daño que una infección podría ocasionar.

#### **Windows**

Windows es una familia de sistemas operativos propietarios desarrollados y comercializados por la empresa Microsoft. Su versión original fue lanzada en 1985 con el fin de brindar una interfaz gráfica a MS-DOS, aunque posteriormente se elaboró el kernel Windows NT, el cual ha evolucionado con el pasar del tiempo y es la base de todas las versiones actuales. La distribución en la que este proyecto se centra es en la de escritorio, ya que históricamente ha sido el sistema operativo más ampliamente utilizado en este ámbito, lo que ha concentrado los esfuerzos de actores malignos sobre él y consecuentemente la atención de investigadores en seguridad.

Una de sus características distintivas es el Registro de Windows [68], una base de datos jerárquica que almacena los parámetros de configuración de bajo nivel del sistema operativo en forma binaria. Resguarda aspectos tales como los atributos de los programas instalados, incluyendo la manera de inicializarlos, la lista de recursos disponibles, los permisos del usuario actual, la lista de drivers instalados, etc.

#### **Linux**

Se trata de un kernel monolítico, libre y de código abierto originalmente creado por Linus Torvalds en 1991 [51]. Está basado en Unix, por lo cual sigue el Estándar de Jerarquía de Archivos [53] (FHS, del inglés Filesystem Hierarchy Standard), el cual define su estructura de directorios.

Siguiendo los lineamientos de dicho sistema operativo, muchos aspectos de Linux son representados como archivos, por lo que elementos tales como documentos, carpetas o dispositivos de entrada/salida son tratados como flujos de bytes expuestos a través de un

espacio de nombres. Usa archivos en texto plano para resguardar la configuración tanto del sistema como de las aplicaciones que en él se ejecutan y trabaja con algunos sistemas de archivos virtuales montados en su jerarquía de archivos única para proporcionar información sobre procesos y muchos otros aspectos.

Tradicionalmente Linux no ha tenido popularidad en ambientes de escritorio, donde lo común es encontrarlo en sistemas operativos GNU/Linux que combinan el kernel con el conjunto de software ofrecido por el proyecto GNU, como por ejemplo Debian [20]. En cambio, su difusión ha sido notable en servidores; actualmente es el sistema operativo más desplegado en este ámbito [51]. Además, presenta una gran difusión en otros aparatos, entre los cuales se encuentran routers, impresoras, cámaras de seguridad, televisores smart, controladores industriales y dispositivos embebidos destinados a IoT (*Internet of Things*, en español Internet de las Cosas) [51].

Debido a su baja adopción en el pasado, este sistema operativo ha sido mayoritariamente ignorado como objeto de ataques informáticos, aunque esta tendencia ha empezado a cambiar dado que los atacantes han tomado conciencia de la criticidad de la plataformas que se sustentan sobre él, tal como menciona Foltyn en su artículo [88]. También han surgido investigaciones referidas a la situación actual del malware en Linux y a las posibilidades para llevar adelante su análisis con tecnologías bajo este entorno [17, 35, 44].

Se concluye entonces que la criticidad de los dispositivos que corren alguna versión de este sistema supone un riesgo lo suficientemente elevado como para requerir estudiar y buscar contrarrestar las potenciales vulnerabilidades que el kernel pueda presentar.

## **Android**

Es un sistema operativo para dispositivos móviles desarrollado por el consorcio Open Handset Alliance. Su principal contribuidor y agente de comercialización es Google. Es de código abierto, aunque cada distribuidor puede incluir componentes propietarias.

Android utiliza un kernel de Linux modificado, aunque posee una API distinguida. Además, aunque los lenguajes de desarrollo principales para este sistema sean Kotlin y Java, Android no opera con el bytecode clásico de Java, utilizando en cambio un formato propio entendido por la componente del sistema conocida como Android Runtime (ART) [34].

Actualmente este sistema domina el mercado de los smartphones, lo que por supuesto ha llevado a que se desarrollen una variedad de programas malignos para el mismo. De entre los estudios realizados sobre el tema, se destaca el realizado por Gajrani et al. [30] sobre la detección de sistemas de análisis dinámico.

## **Multisistema**

Esta categoría adicional aborda técnicas que son aplicables a más de un sistema operativo. Esto puede deberse a que se basan en la observación de propiedades de un

nivel más bajo de abstracción, tal como la arquitectura del procesador; porque realizan un análisis genérico del comportamiento del ambiente, como la gestión de la suspensión de un hilo; o porque abordan conceptos generales de sistemas operativos sin profundizar en las características particulares de un producto determinado. Para ejemplificar esto último podemos mencionar la enumeración de los procesos o archivos presentes en el sistema; la concreción de estas actividades en distintos sistemas operativos no difiere más que en las APIs disponibles para dicho fin.

## **2.4.2 Clasificación de técnicas según su meta**

Consultar por la presencia de un archivo y por su contenido es una práctica común llevada a cabo por el malware que trata de evadir a los entornos de análisis. Sin embargo, la información obtenida de dicho fichero y que un atacante espera es ampliamente variable: quizás es un archivo de configuración de una máquina virtual, revelando así su presencia; quizás es un indicio del uso del sistema por parte de un ser humano o también podría ser el binario ejecutable de alguna herramienta de monitoreo.

Debido a que una misma práctica puede ser realizada con distintos propósitos, en la siguiente clasificación se recopilan aquellos que han podido ser indagados tras la investigación efectuada. Se puede apreciar en la Figura 1 un diagrama resumen de la categorización efectuada.

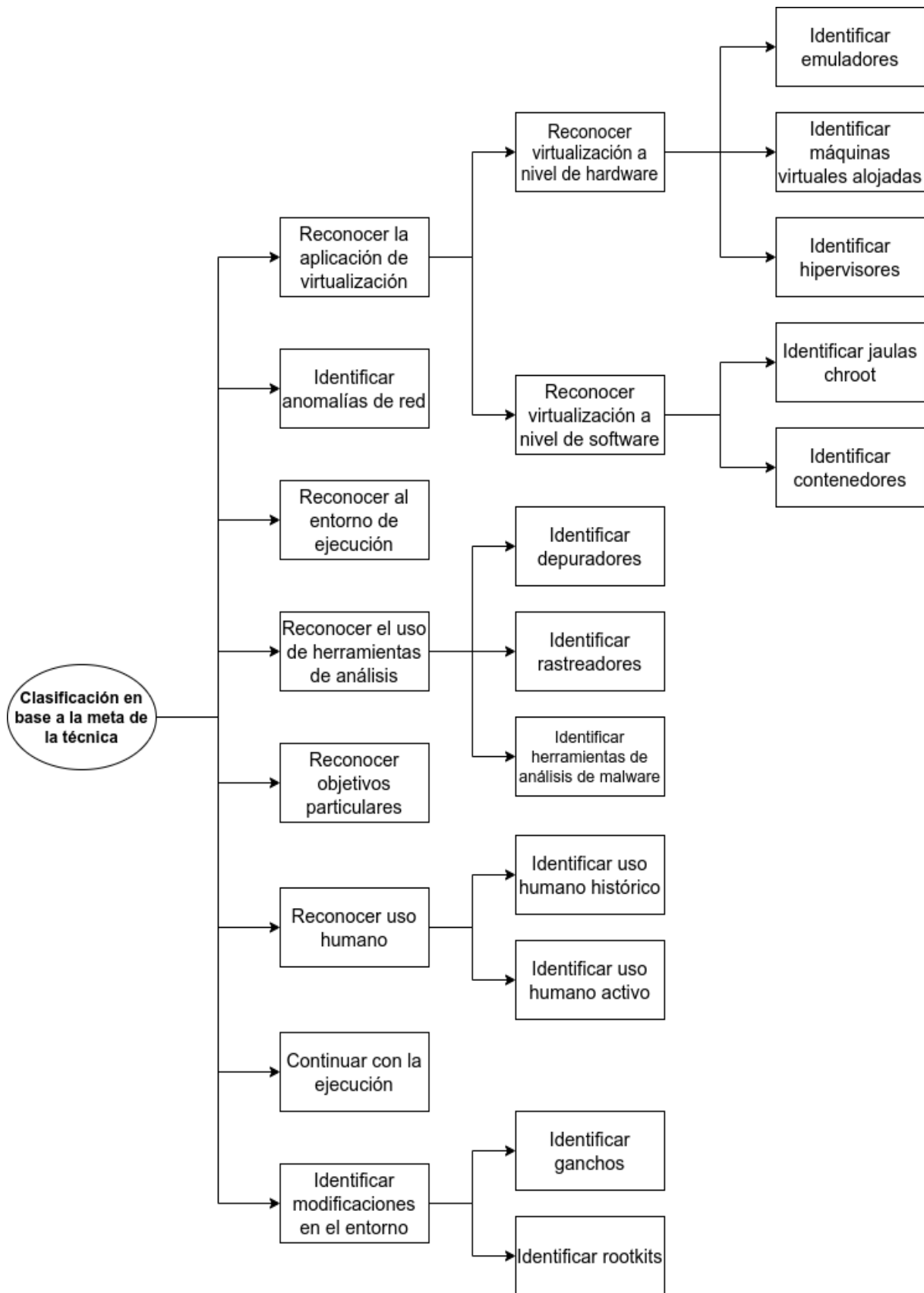


Figura 1: Clasificación de técnicas en base a su intención



## Reconocer la aplicación de virtualización

Debido a la trayectoria del mercado en la construcción de sandboxes sustentados por infraestructura virtualizada, esta ha sido la intención más ampliamente observada en las muestras de malware evasivo. La virtualización se refiere a la provisión mediante software de versiones virtuales de recursos tecnológicos.

En el siguiente listado se presentan las formas de virtualización abordadas en este trabajo:

- **Virtualización a nivel de hardware:** en primera instancia interesa la confección de ambientes lógicos a través de la virtualización de equipos de computadoras, también conocida como virtualización a nivel de hardware. Bulazel et al. [10] explican que se habla de “invitado” (*guest*) para referirse al sistema que está siendo virtualizado, mientras que se usa el término “anfitrión” (*host*) para aludir al sistema que efectúa la virtualización. Los mismos autores describen que esta puede ser implementada a través de alguna de las siguientes formas:

- **Emuladores:** todas las instrucciones del invitado son ejecutadas por software, lo que permite que el invitado y el anfitrión tengan arquitecturas de procesador distintas. QEMU [63] es uno de los emuladores más populares disponibles hoy en día.

Pueden controlar, monitorear y modificar el estado del sistema cómodamente, lo que facilita el estudio de programas maliciosos. Sin embargo, debido a que virtualizan completamente un equipo computacional presentan un notorio déficit en su rendimiento. Por otro lado, los errores en la implementación de las instrucciones de máquina del procesador simulado conllevan a diferencias entre los resultados obtenidos en aquel y en su equivalente real que pueden ser utilizados como mecanismos de detección. Paleari et al. [59] presentaron un procedimiento automatizado para generar pruebas con el fin previamente explicado.

- **Máquinas virtuales alojadas:** se tratan de máquinas virtuales que se desenvuelven en sistemas operativos anfitriones. Debido a esto, su desempeño suele ser mejor que el de un emulador, aunque no poseen sus facilidades para estudiar el comportamiento de software. En la literatura se las puede encontrar también con el nombre de hipervisores de tipo 2. Su uso en la industria del análisis de malware está ampliamente extendido, en donde VMWare [82] y VirtualBox [80] destacan como las plataformas sobre las cuales se han desarrollado diversos sandboxes.

Requieren que la arquitectura del procesador del invitado sea la misma que la del anfitrión ya que la mayor parte de las instrucciones del invitado se ejecutan directamente sobre hardware. Esto último produce que una instrucción ejecutada con o sin virtualización mediante arroje los mismos resultados,

evitando así los errores presentes en emulación. Por otro lado, gracias a las facilidades para virtualizar provistas por los procesadores modernos (las tecnologías VT-x [2] de Intel y AMD-V [1] de AMD) no se requiere la manipulación manual de instrucciones no virtualizables que podrían revelar la presencia de una máquina virtual.

- **Hipervisores:** estos sistemas son ejecutados directamente sobre hardware y virtualizan código de la misma arquitectura que el anfitrión, lo cual produce una sobrecarga menor que otros tipos de virtualización, contribuyendo así a la transparencia de estos sistemas. Son también conocidos como máquinas virtuales nativas, máquinas virtuales sobre hardware o hipervisores de tipo 1. El hipervisor Xen [89] es notable en esta categoría.
- **Virtualización a nivel de software:** como contraparte a la virtualización a nivel de hardware se encuentra este tipo de virtualización. En ella el aislamiento se consigue a través de múltiples instancias de espacios de usuario que comparten el mismo kernel subyacente.
  - **Jaulas chroot:** *chroot* es una operación presente en los sistemas operativos derivados de Unix que modifica el directorio raíz tanto para el proceso que lo invoca como para los hijos de este último. Con esto se impide el acceso al resto del sistema de archivos que haya quedado fuera de la nueva raíz; a su vez, dentro de esta pueden colocarse solo las librerías, archivos y carpetas generales indispensables para que un programa se ejecute correctamente. En resumen, las jaulas *chroot* encapsulan a un proceso dentro de un sistema de archivos acotado.
  - **Contenedores:** son la forma más difundida hoy en día de virtualización a nivel de software. No son máquinas virtuales, sino que pueden ser entendidos como entornos virtuales con sus propios procesos, sistema de archivos y pila de red. Debido a esto presentan un rendimiento similar al de una máquina desnuda. Los instrumentos para proveer contenedores más reconocidos hasta el momento son LXC [85] y Docker [23]; ambos aprovechan herramientas provistas por el kernel de Linux para brindar la separación entre espacios virtuales, entre ellas la ya explicada operación *chroot*.

Se ha puesto en duda la efectividad de los contenedores en la elaboración de sandboxes debido a que sus mecanismos de aislamiento no son tan rigurosos como los presentes en la virtualización a nivel de hardware. Sin embargo, las particularidades de esta tecnología la hacen naturalmente inmune a muchos métodos de detección conocidos; a su vez, su escasa sobrecarga es un atractivo para equipos de bajos recursos, como puede ser un dispositivo IoT, en los cuales no es viable la aplicación de los esquemas de virtualización explicados previamente. Hellinger et al. [35] y Kedrowitsch et al. [44] han

presentado investigaciones sobre las características de los contenedores como entornos de análisis dinámico de malware.

## Reconocer el uso de herramientas de análisis

Mediante las técnicas aquí englobadas un software maligno trata de encontrar indicios de instrumentación mediante la cual puede inferir que está siendo estudiado. Los métodos de detección identificados incluyen la búsqueda de archivos ejecutables y procesos correspondientes a herramientas conocidas, la consulta de banderas provistas por el sistema operativo que informan que un programa en ejecución está siendo depurado, la comprobación del rendimiento actual del entorno (dado que la monitorización de software suele incurrir en una sobrecarga notable para el sistema), el hallazgo de puntos de ruptura (*breakpoints*), entre otros.

A continuación se presentan algunos de los instrumentos más comúnmente buscados:

- **Depuradores (*debuggers*):** después de la recolección de pistas para identificar a un equipo virtualizado este caso es el más común. La depuración es un proceso aplicado comúnmente para identificar y solucionar errores de programación a través de la ejecución controlada de un programa. A través del mismo se monitorean las operaciones llevadas a cabo por el proceso bajo estudio y los cambios en los recursos computacionales provocados por ellas. Los depuradores permiten detener la ejecución de un programa a través de puntos de ruptura y muestran el estado de la memoria, los registros de la CPU y otros recursos involucrados. Un depurador popular usado en muchos sistemas derivados de Unix es GDB [32] (GNU Debugger), el cual soporta diversos lenguajes de entre los cuales se encuentran C, C++ y Fortran.

Existen dos tipos: el primero de ellos comprende a los depuradores a nivel de fuente o simbólicos, que permiten trabajar con el código fuente de una aplicación; su uso en ambientes de desarrollo está universalmente difundido. El segundo tipo es el de los depuradores a bajo nivel o de lenguaje de máquina, cuyo funcionamiento se basa en emplear un desensamblador sobre el archivo binario del programa que se busca estudiar; este es el tipo más común en el análisis de malware ya que en este campo no se suele contar con el código fuente del software a monitorear.

- **Rastreadores (*tracers*):** son instrumentos de bajo nivel que registran información sobre un proceso. Son aplicados para medir su desempeño, sus llamadas al sistema operativo, su flujo de paquetes TCP/IP, entre muchos otros aspectos más. En Linux la herramienta *strace* [78] es particularmente conocida. La misma es capaz de monitorear y manipular interacciones entre un proceso y el kernel.
- **Monitores de red:** constan de software capaz de observar el flujo de las comunicaciones a través de la red. Proporcionan información acerca de la dirección IP tanto de origen como de destino de un paquete, las direcciones MAC

correspondientes, los protocolos utilizados, la carga de la red, etc. Entre los programas de este estilo más difundido se encuentra Wireshark [87], que se especializa en el análisis de paquetes.

- **Herramientas de análisis de malware:** en este apartado se incluyen todas las herramientas disponibles desarrolladas específicamente para la investigación de software maligno. Aquí se encuentran los sandboxes para dicho fin, los cuales suelen estar implementados mediante alguna infraestructura virtualizada combinada con las herramientas de análisis previamente explicadas. Cuckoo Sandbox [19] goza de buena popularidad en este ámbito.

## Identificar anomalías de red

Un malware puede decidir no avanzar con su payload si detecta condiciones extrañas en la conectividad. Algunos elementos mencionables de este conjunto de anomalías son una velocidad de internet extremadamente alta, la correcta resolución de consultas DNS por recursos que no existen o la falta total de conectividad.

## Reconocer al entorno de ejecución

En el caso de programas malignos cuya ejecución depende de un intérprete (como los codificados en los lenguajes Python, PHP, Java o Javascript) o que se sustentan en un navegador web, las características del entorno de ejecución pueden resultar ser un factor decisivo. Algunos ejemplos de ellas son la versión del entorno que el mismo informa o su comportamiento (que puede variar entre versiones y entre productos alternativos). Que un entorno de ejecución tenga características contradictorias puede ser tomado como una pista de que se trata de un impostor cuyo fin es la de estudiar los programas que son corridos sobre él.

## Identificar modificaciones en el entorno

En algunos entornos se introducen cambios en el sistema con el propósito de esconder, manipular o fabricar la información consultada por un proceso; otro fin es el de ampliar programas y librerías para que además de llevar a cabo sus funciones originales también realicen operaciones de análisis de forma transparente. A continuación se presentan dos formas comunes de implementar estas modificaciones:

- **Ganchos:** se denomina *hooking* a la técnica por la cual son instrumentados y modificados el flujo y comportamiento de llamadas a una API. En el contexto del estudio de programas se usa para introducir una función adicional (a la que se le denomina gancho) cada vez que una llamada es realizada, la cual es responsable de realizar las tareas de análisis correspondientes. Su aplicación más común en este contexto involucra la interceptación de llamadas al sistema.
- **Identificar rootkits:** un rootkit está conformado por un conjunto de software que permite acceder de forma privilegiada a un equipo computacional manteniendo su presencia oculta. Aunque son usados comúnmente por malware para ocultar su actividad maliciosa, también pueden ser implementados en entornos de análisis

para esconder procesos, archivos y otros recursos contrarios a su transparencia. Los rootkits pueden existir en el espacio de usuario, a nivel de kernel, a nivel del hipervisor o en el firmware de un dispositivo. Se recomienda la lectura del artículo elaborado por Levine et al. [49] sobre rootkits que sobrescriben la tabla de llamadas al sistema de Linux.

## **Reconocer uso humano**

Aquí se engloban todas las técnicas que buscan hallar pistas sobre la utilización del sistema por parte de una persona. Se pueden discernir dos tipos, siendo el primero de ellos el de las técnicas que escanean el sistema en busca de artefactos que son resultado de la actividad humana. Ejemplos de estos últimos son los archivos multimedia y los documentos personales generalmente ubicados en directorios tradicionales para el sistema operativo en consideración, los archivos abiertos recientemente, el historial de navegación web, los nombres de usuarios del sistema y el registro de llamadas en un dispositivo móvil.

El segundo tipo involucra a aquellos medios por los cuales se busca identificar que un humano está utilizando el sistema en un preciso instante. Tal es el caso de la monitorización de las teclas presionadas y de los movimientos del mouse, como a su vez los valores arrojados por el giroscopio en un smartphone.

## **Reconocer objetivos particulares**

Se trata de aquellas técnicas aplicadas por agentes malintencionados que buscan infectar a un equipo o equipos específicos, evitando todo aquel que no presente ciertas características esperadas. Como consecuencia de esto, son capaces de esquivar ambientes de análisis que no reproduzcan las propiedades buscadas. Un caso ejemplar es esperar que un supuesto servidor tenga ciertas aplicaciones instaladas y en ejecución activa, además de tener ciertos puertos abiertos.

## **Continuar con la ejecución**

Un programa dañino puede que simplemente necesite de ciertas condiciones del entorno para poder cumplir su propósito y que ante el incumplimiento de estas no sea capaz de ejecutarse debidamente. Se puede ejemplificar esta situación con un software que depende de ciertas bibliotecas compartidas que no se encuentran presentes en el sistema.

### **2.4.3 Clasificación de técnicas según la metodología aplicada**

En esta clasificación se asocian técnicas cuyos procedimientos son semejantes; esto es, presentan algoritmos similares y dependen de los mismos recursos físicos y lógicos (o sus equivalentes entre sistemas computacionales intrínsecamente distintos).

Remitiendo a las propiedades de los entornos de análisis referidas a la transparencia explicados previamente, en primera instancia se distribuyen las metodologías aquí

descriptas según la propiedad involucrada. En la Figura 2 se exhibe el diagrama asociado a esta clasificación.

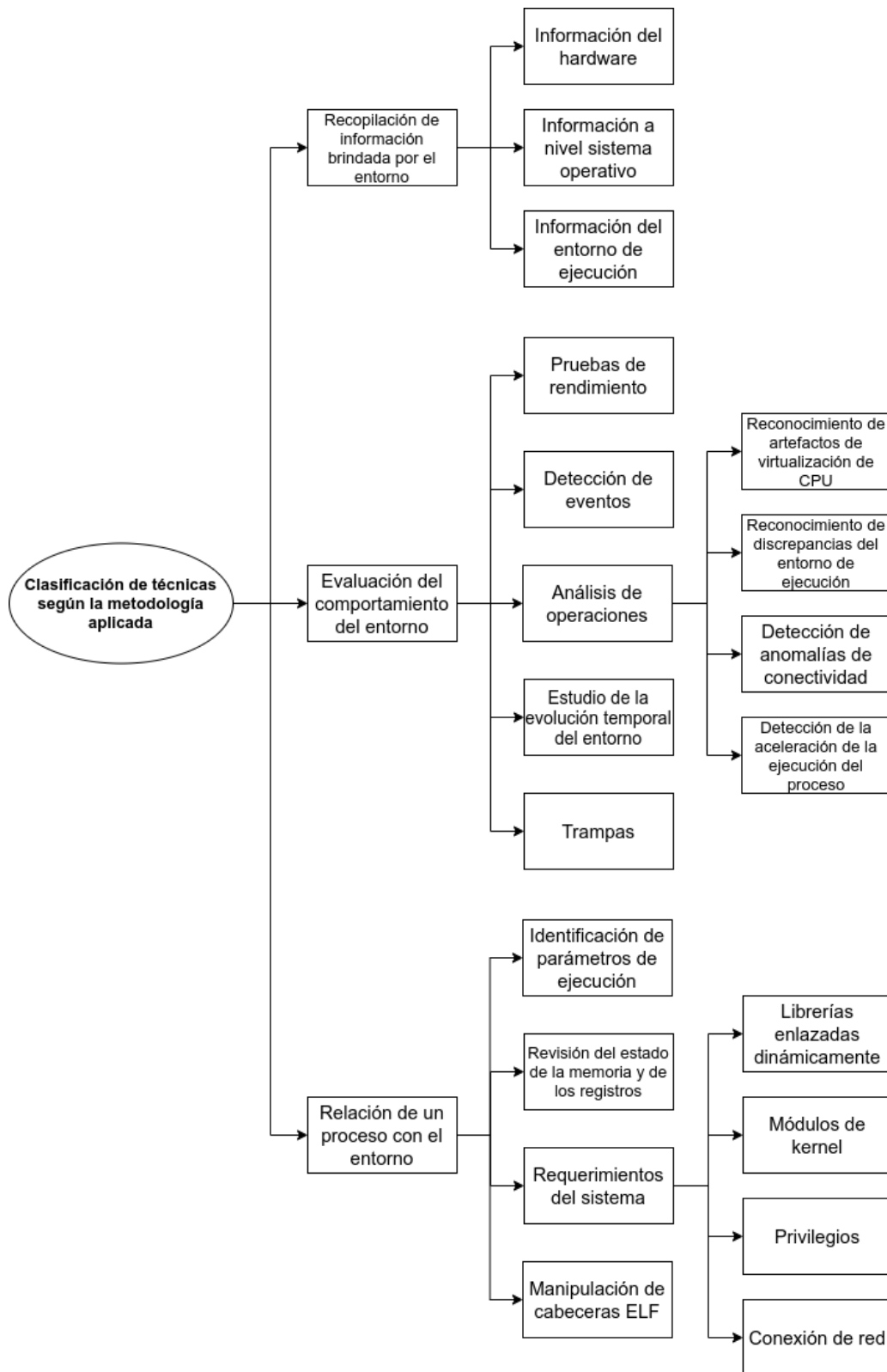


Figura 2: Clasificación de técnicas de reconocimiento de entornos según la metodología aplicada

### **2.4.3.1 Técnicas que recopilan información brindada por el entorno**

Se tratan de prácticas que buscan datos provistos por el sistema mismo. La información que puede encontrarse puede referirse a los siguientes aspectos:

- **Información del hardware:** engloba las características del supuesto dispositivo físico sobre el cual un proceso está operando. Involucra datos tales como:
  - El fabricante, número de serie o el modelo de los recursos físicos instalados.
  - La velocidad del procesador, sus instrucciones soportadas o la cantidad de núcleos que posee.
  - La velocidad de la memoria, su tipo o su capacidad.
  - Capacidad del disco instalado o la forma en la que está particionado.
  - Tipo de los medios extraíbles conectados.
  - Características de los sensores presentes, tales como cámaras, giroscopios, acelerómetros, etc.
- **Información a nivel de sistema operativo:** en este apartado se engloban datos derivados de elementos lógicos del sistema, tales como:
  - Información propia del sistema operativo, como el desarrollador o la versión del kernel.
  - La configuración del sistema, como los parámetros del kernel o los servicios cuya inicialización está prevista en el arranque.
  - Usuarios, grupos y permisos asociados.
  - Los procesos en ejecución del sistema, sus identificadores correspondientes o la presencia de banderas que indiquen su depuración activa.
  - Las ventanas abiertas y la que tiene el foco en un preciso instante.
  - Características de los sistemas de archivos montados, los documentos presentes y su contenido.
  - La configuración de las interfaces de red.
- **Información del entorno de ejecución:** se aplica a programas que dependen de algún intérprete, navegador o similar para operar. Involucra características tales como la versión anunciada por el entorno o sus módulos instalados.

### **2.4.3.2 Técnicas que evalúan el comportamiento del entorno**

Se refieren a métodos a través de los cuales se analiza al sistema para determinar como reacciona ante distintas cargas de trabajo, cuales son los resultados generados por instrucciones particulares, etc.

- **Pruebas de rendimiento:** estas prácticas verifican el desempeño del sistema al ejecutar ciertas tareas, la velocidad de internet disponible, entre otros. Entre las posibilidades de implementación disponibles destacan la comparación de los resultados obtenidos contra bases de datos de valores esperados, según el supuesto modelo de la componente del ambiente analizada, y el estudio de la varianza obtenida tras ejecutar repetidamente un experimento.
- **Detección de eventos:** implican técnicas a través de las cuales se identifican y analizan eventos generados en el sistema, tales como los derivados del movimiento de un cursor o la pulsación de teclas. Esta táctica se usa en general con el fin de reconocer el uso activo del sistema por parte de un humano.
- **Estudio de la evolución temporal del entorno:** un malware puede estudiar pasivamente los cambios que experimenta el ambiente durante un periodo de tiempo y actuar de una u otra manera dependiendo de lo que haya podido apreciar. Un ejemplo de estas técnicas es la observación del ritmo de creación de procesos en el sistema.
- **Análisis de operaciones:** las pruebas que aquí se agrupan se caracterizan por llevar a cabo determinadas tareas para luego analizar los resultados obtenidos y el estado del entorno consecuente.
  - **Reconocimiento de artefactos de virtualización de CPU:** también conocidas como *red-pills*, las técnicas aquí abarcadas fueron mencionadas brevemente cuando se describieron a los emuladores y las maneras de identificarlos. Se basan en la ejecución de una o más instrucciones de máquina que se comportan de distinta forma cuando son ejecutadas en un ambiente real que cuando son ejecutadas en uno emulado. Nuevamente se recomienda leer el trabajo de Paleari et al. [59] en el que se explica como se desarrolló una metodología automatizada para generar este tipo de pruebas.
  - **Reconocimiento de discrepancias del entorno de ejecución:** con el propósito de detectar ambientes impostados o que presentan anomalías, los procesos malintencionados pueden utilizar este tipo de técnicas. La más común de ellas es observar la ejecución de ciertas instrucciones que arrojen resultados distintos en ciertos motores de ejecución debido a diferencias en la implementación de distintos fabricantes y sus versiones correspondientes; esto se ve comúnmente en malware codificado en lenguajes interpretados o compilados justo a tiempo, destacándose entre estos JavaScript por su difusión en navegadores.

Estos últimos exhiben vulnerabilidades particulares, siendo algunas de ellas:

- Diferencias del parseador de HTML y CSS entre distintos navegadores.
- *Document Object Model* (DOM) incompleto (comúnmente visto en emuladores de navegadores).



- “Ruido” que se produce tras el renderizado del *canvas* (lienzo) de HTML5 que permite distinguir al sistema subyacente [11].
- **Detección de anomalías de conectividad:** algunos programas malintencionados están programados para identificar situaciones irregulares a la hora de hacer peticiones en línea. Un ejemplo de esto son las peticiones DNS por recursos que no existen; si las mismas son respondidas con éxito se puede deducir que el sistema está simulando la conexión a internet y por ende debe tratarse de un entorno de análisis.
- **Detección de la aceleración de la ejecución del proceso:** los sandboxes para análisis dinámico de malware suelen tener definidos tiempos acotados para el estudio de cada muestra. Para reducir la probabilidad de que se agote dicho tiempo antes de que un proceso llegue a exhibir su comportamiento maligno, estos entornos aplican estrategias para acelerar la ejecución o saltar ciertos periodos de inactividad (como los que se producen mientras un proceso duerme). Estas medidas pueden ser identificadas mediante la comprobación del tiempo real transcurrido tras llevar a cabo ciertas acciones.
- **Trampas:** los programas malignos pueden contener código que al ser indagado por un depurador u otros instrumentos, la producción de información específica (o su ausencia) revela la presencia del mismo. Una táctica popular en esta categoría es usar ciertas instrucciones y observar cómo son manejadas. Un ejemplo es el del malware Max++, el cual aprovecha una instrucción que en ausencia de un depurador genera una excepción que puede ser atrapada por el software malicioso; en cambio, si tiene un depurador asociado la excepción es transferida al mismo y nunca llega al malware, el cual puede inferir a partir de esto que está siendo estudiado [38].

### **2.4.3.3 Técnicas basadas en la relación de un proceso con el entorno**

Se fundamentan en componentes, configuraciones y parámetros del sistema que afectan directamente a la operatoria de un proceso, incluso imposibilitando su correcta ejecución.

- **Requerimientos del sistema:** aquí se presentan algunos de los elementos que deberían estar presentes en un entorno para que un programa pueda trabajar fructuosamente:
  - **Librerías enlazadas dinámicamente:** el entorno en donde se desenvuelve un proceso debe contener todas las bibliotecas compartidas que este necesita; con que una sola falte puede ser suficiente para que la ejecución de aquel no sea fructuosa. Estas librerías aparecen comúnmente como binarios DLL en los sistemas operativos Windows y como archivos ELF en sistemas derivados de Unix como Linux.

- **Módulos de kernel:** se tratan de archivos que contienen código objeto, a través de los cuales se extiende el núcleo en ejecución, dando así soporte a dispositivos, formatos de sistema de archivos u otros elementos. Un software también puede depender de estas componentes para funcionar correctamente. En Windows se conocen más comúnmente como controladores (*drivers*).
- **Privilegios:** las operaciones llevadas a cabo por un proceso pueden requerir de ciertos permisos que el sistema operativo no le otorgue, previniendo así su correcta ejecución.

Un ejemplo pintoresco de este tipo de técnicas aplicable en Linux es intentar ejecutar ciertos comandos desde un contenedor: un proceso adentro del mismo puede estar asociado a un usuario y grupo privilegiados, los cuales se encuentran mapeados a un usuario no privilegiado en el sistema anfitrión. Dado que ciertos comandos que trabajan directamente con el núcleo necesitan sí o sí permisos elevados, el intento por ejecutarlos en este contexto se ve frustrado, lo que demuestra una discrepancia en los privilegios supuestamente asignados al proceso. Finalmente, de esta anomalía se puede deducir la presencia de un contenedor [35].

- **Conexión a la red:** un malware puede requerir que el sistema subyacente se encuentre conectado a una red local o a internet.
- **Identificación de parámetros de ejecución:** ciertas configuraciones establecidas sobre la ejecución de un proceso son un indicio de que está siendo estudiado. La forma más común de encontrar estos parámetros es como variables de entorno.

Resulta curioso mencionar que en sistemas Unix se pueden enlazar símbolos (funciones, estructuras, variables, etc) provistos por ciertas bibliotecas compartidas antes que otras a través de la variable de entorno LD\_PRELOAD [33]. Esto se puede aplicar para incorporar funcionalidades de análisis a la hora de arrancar un proceso, aunque se debe tomar la precaución de esconder la variable de forma tal que no pueda ser fácilmente consultada.

- **Revisión del estado de la memoria y de los registros:** se basa en comprobar los elementos fundamentales del estado de un proceso, los registros y la memoria asignada, en busca de alteraciones o agentes extraños introducidos por manipulaciones ejercidas por el entorno. Un software potencialmente podría identificar estos cambios, ya que los valores presentes son inesperados o resultan absurdos para el flujo de ejecución actual.

En investigación de malware estas perturbaciones se realizan típicamente para modificar los datos de operación de una muestra o sus instrucciones con la intención de forzar la ejecución de su carga dañina. Por otra parte, los depuradores comúnmente ejercen transformaciones al introducir puntos de ruptura en el código del programa cargado en memoria bajo estudio.

- **Manipulación de cabeceras ELF:** una clase de técnicas particular se presenta en sistemas tipo Unix. ELF (*Executable and Linkable Format*, en español Formato Ejecutable y Enlazable) es el formato usado en dichos sistemas para archivos ejecutables, código objeto, bibliotecas compartidas y volcados de memoria [7].

Un método capaz de prevenir el análisis de un binario en este formato es la alteración de su cabecera alterando valores por otros incorrectos o inesperados, lo que conlleva a que una herramienta que deba analizarla sintácticamente (como un depurador) falle o, peor aún, se paralice o cierre inesperadamente. Esto se fundamenta en que son pocos los campos específicos de la cabecera de un archivo ELF que son leídos realmente por el núcleo cuando se carga el programa en memoria; sin embargo, este análisis no es llevado a cabo igual por el kernel que por otras herramientas particulares, lo que conduce a fallos en su ejecución al leer los susodichos valores [43].

## 2.5 Recopilación de técnicas de evasión de análisis dinámico de malware basadas en el reconocimiento de entornos

En paralelo a la confección de las clasificaciones expuestas, se aprovechó la misma bibliografía utilizada para elaborar un compendio con las técnicas de anti-análisis que allí pudieron encontrarse. No todas las halladas fueron agregadas; se determinaron que ciertas condiciones debían cumplirse para que cada técnica recuperada fuera de utilidad para este trabajo:

- La técnica debía estar directamente relacionada con la transparencia del ambiente en el que se llevara a cabo para poder considerarla dentro del alcance del verificador planteado.
- La técnica debía estar lo suficientemente bien detallada en su fuente para entender el concepto que la respalda. Aquellas que presentaban un alto grado de ambigüedad fueron descartadas para evitar recurrir a más bibliografía y así mantener razonablemente delimitada la misma.

A partir del material revisado fueron reconocidas 132 formas de diferenciar entornos; a todas ellas se les asignó una categoría por cada una de las clasificaciones propuestas en el apartado anterior. Debido a la extensión de esta recopilación, la misma ha sido incorporada mediante el Apéndice A. Aquella es un importante insumo para la fabricación de un verificador de transparencia, ya que da lugar a la conversión de sus técnicas en pruebas útiles para efectuar esta evaluación.

## 2.6 Conclusiones

En este capítulo se ha presentado la problemática del malware que logra esquivar los esfuerzos de análisis dinámico sobre sí mismo mediante la aplicación de técnicas de reconocimiento de entornos y cómo esto tiene un impacto negativo en el ámbito de la

seguridad informática. A su vez, se ha determinado que la mejor estrategia para mitigar esta cuestión es la elaboración de entornos más transparentes, para lo cual se ha planteado el desarrollo de un verificador de transparencia de entornos.

Ante la falta de valor práctico que tiene la forma en que se ha tratado este tópico en la bibliografía revisada, se confeccionaron tanto una definición de transparencia como clasificaciones de técnicas de identificación de entornos propios. Esto, en conjunto con la elaboración de un compendio de técnicas, describe el alcance y potencial de un producto como el mencionado, lo que aporta una entrada significativa para su correcto diseño y construcción.

## Capítulo 3: Requerimientos del mercado

El producto objeto de este trabajo, un verificador de transparencia de entornos de análisis dinámico de malware, desde un principio se planteó como un software enlatado dirigido a la comunidad especializada en la seguridad informática y la administración de sistemas. Para comprender las características del mercado donde se incrustaría el producto, se realizó un análisis inspirado en el proceso de elaboración de un Documento de Requerimientos del Mercado [75] (conocido comúnmente como MRD, por sus siglas en inglés).

### 3.1 Segmentos objetivo

La bibliografía recuperada para confeccionar tanto la clasificación de técnicas de evasión de análisis como el compendio de aquellas que sirven para identificar entornos resultó tener un tercer propósito: servir de material para llevar adelante lo que en análisis de mercado se conoce como una investigación secundaria del público potencialmente interesado en un verificador de transparencia.

Así, tres segmentos de mercado particulares pudieron identificarse:

- El de los investigadores de malware, quienes utilizan todo tipo de herramientas con el fin de poder comprender la composición y accionar de programas maliciosos. No solo está compuesto por académicos, sino que también hay empresas como IBM que cuentan con equipos de investigación que incluso han aportado contenido acerca de la evasión de ambientes de análisis [36].
- El de los administradores de sistemas, quienes utilizan antivirus, honeypots y otros mecanismos de defensa que aplican análisis dinámico para proteger las infraestructuras a sus cargos.
- Los fabricantes de sandboxes para análisis de malware, quienes están al tanto de la existencia de técnicas de elusión y buscan mejorar la calidad de sus productos. Algunos de ellos incluso han elaborado material respecto a este tema, tal como McAfee [70] y VMRay [5],

El software maligno que evita ser analizado presenta un problema fundamental para todos estos actores, ya que atenta directamente contra sus objetivos: los investigadores se ven incapaces de llevar adelante estudios, los administradores no pueden asegurar sus sistemas y los desarrolladores de sandboxes ven mermada la efectividad de sus soluciones. Consecuentemente, todos ellos se verían beneficiados por el uso de un instrumento que permita identificar las técnicas de reconocimiento contra las cuales un ambiente de análisis es o no es resistente.

## 3.2 Requerimientos del producto

De la investigación secundaria se pudieron recopilar también los requisitos fundamentales de un verificador de transparencia para poder satisfacer las necesidades subyacentes de los segmentos identificados:

- La diversidad de técnicas de anti-análisis, producto de las condiciones tan variadas que se pueden presentar en un sistema computacional, y la constante evolución de estas hacen indispensable que un verificador de transparencia tenga la capacidad de incorporar nuevas pruebas definidas por sus usuarios de manera sencilla y sin requerir la modificación del código fuente del programa.
- De la necesidad anterior se desprende el requisito de gestionar un repositorio de de tests.
- El sistema debe ser capaz de resguardar los resultados históricos de los tests realizados y brindar mecanismos para buscar y consultar esta información.
- La solución ofrecida debe ser multiplataforma, hablando en términos tanto de arquitectura de computadoras como de sistema operativo. Lo primero se debe a la multitud de técnicas basadas en red-pills que conlleva a que el sistema deba contemplar la ejecución de código pensado para explotar las particularidades de una arquitectura destino. Lo segundo es consecuencia del alcance del malware y de las características particulares de cada sistema operativo, tópico que ya fue abarcado en el apartado 2.4.1.
- Se ha podido observar que es recurrente en el nicho del análisis dinámico de software maligno el uso de múltiples entornos de ejecución, ya sea que estén conformados con distintas tecnologías o que presenten configuraciones personalizadas. Esto puede deberse tanto por motivos de estudio y comparativa de las distintas herramientas disponibles para la elaboración de sandboxes (en término tanto de hardware como de software) o por el uso complementario de más de un entorno con el fin de mejorar los resultados obtenidos al analizar una muestra [50]. Es por ello que resulta de interés que el producto pueda asistir en la gestión y ejecución de pruebas en múltiples ambientes en funcionamiento.

## 3.3 Productos existentes

Aunque la problemática que presenta el reconocimiento de sandboxes para análisis de malware está en la consideración de empresas e investigadores de la disciplina de la seguridad informática, no existen proyectos de software populares que busquen asistir en la elaboración de mejores entornos. Particularmente son solo siete las herramientas que se han podido encontrar públicamente en la web al momento de la confección de este trabajo; las cuales no presentan difusión en el mercado: Pafish [58], Sandbox\_tester [71], InviZzible [76], sems [74], VMDE [81], AI-Khaser [55] y SandGrox [88].

En el afán de descubrir posibles causas por los cuales estos productos no han tenido éxito, se han analizado las fortalezas y debilidades de cada uno, para lo cual se ha tomado como principal criterio diferenciador el grado de cumplimiento de los requisitos enunciados previamente. Esta tarea ha sido posible gracias a que cinco de los seis instrumentos tienen su código fuente publicado; debido a que SandGrox no está disponible para su uso de ninguna forma, sus características se han recopilado solamente en base a la información provista por su autor.

A continuación se presentan las virtudes y limitaciones de las herramientas encontradas:

- **Pafish**

- **Fortalezas:**

- De licencia libre.
- Incluye por defecto un conjunto variado de tests.

- **Debilidades:**

- Su desarrollo se detuvo en el 2016.
- Es solo una prueba de concepto.
- Dispone de un conjunto estático de pruebas que no puede ser ampliados sin modificar directamente el código fuente de la herramienta.
- No permite especificar pruebas particulares a ejecutar.
- Orientado a Windows únicamente.
- No presenta persistencia de los reportes generados.
- Solo se encarga de ejecutar pruebas en un único entorno.
- No dispone de interfaz gráfica.

- **Sandbox\_tester**

- **Fortalezas:**

- Implementa una arquitectura mediante la cual un servidor HTTP y otro de DNS se utilizan para recuperar externamente los reportes generados por la ejecución de pruebas en un sandbox.
- Utiliza una base de datos integrada para persistir los resultados obtenidos.

- **Debilidades:**

- No especifica una licencia.
- Su desarrollo se detuvo en 2017.

- Utiliza Python 2, versión a la que ya se le ha dejado de dar mantenimiento.
- Orientado a Windows únicamente.
- Dispone de un conjunto estático de pruebas que además están acotadas en su alcance. No se puede ampliar sin modificar el código fuente del programa.
- No permite especificar pruebas particulares a ejecutar.
- Desde la aplicación no se brindan utilidades para consultar la información almacenada en la base de datos, por lo que se debe recurrir a una herramienta externa para ello.
- No dispone de interfaz gráfica.
- **InviZzible**
  - **Fortalezas:**
    - De licencia libre.
    - Se actualizó por última vez en diciembre de 2019, lo cual es un indicio de que su desarrollo aún se encuentra activo.
    - Incluye por defecto un conjunto variado de tests.
    - Permite seleccionar ciertos conjuntos de pruebas a ejecutar.
    - Permite incorporar nuevos tests mediante su descripción en formato JSON.
    - Vuelca los reportes generados en archivos para su persistencia.
    - Permite visualizar los reportes en formato HTML.
  - **Debilidades:**
    - La manera en que se incorporan nuevas prueba no está debidamente documentada.
    - El formato JSON para describir nuevas pruebas es rígido, por lo que agregar técnicas de anti-análisis más avanzadas requiere modificar el código fuente.
    - Solo se encarga de ejecutar pruebas en un único entorno.
    - Orientado a Windows únicamente.
    - No otorga mecanismos para buscar reportes almacenados.
    - No dispone de interfaz gráfica.
- **sems**
  - **Fortalezas:**



- Persiste los resultados obtenidos en archivos organizados según el perfil de la herramienta de virtualización o análisis relacionada.
- **Debilidades:**
  - No especifica una licencia.
  - Su desarrollo se encuentra paralizado desde el 2016.
  - Es solo una prueba de concepto.
  - Dispone de un conjunto estático de pruebas que además están acotadas en su alcance.
  - Orientado a Windows únicamente.
  - Solo se encarga de ejecutar pruebas en un único entorno.
  - No permite especificar tests particulares a ejecutar.
  - No presenta mecanismos para buscar reportes bajo algún criterio dado.
  - No dispone de interfaz gráfica.
- **VMDE**
  - **Fortalezas:**
    - De licencia permisiva.
    - Es capaz de producir archivos que contienen los resultados obtenidos tras la ejecución de pruebas.
  - **Debilidades:**
    - La última vez que se actualizó fue en el 2017 y su repositorio se encuentra archivo, lo que indica que el proyecto ha sido descontinuado.
    - Dispone de un conjunto estático de pruebas que además solo busca identificar virtualización.
    - Orientado a Windows únicamente.
    - Solo se encarga de ejecutar tests en un único entorno.
    - La selección de tests a ejecutar se realiza mediante banderas lógicas incrustadas en el código fuente, lo que requiere recompilar el programa cada vez que se desea especificar otro conjunto de pruebas.
    - No presenta mecanismos para buscar reportes bajo algún criterio dado.
    - No dispone de interfaz gráfica.
- **Al-Khaser**
  - **Fortalezas:**

- Desarrollo activo.
- Las pruebas implementadas están documentadas.
- Incluye un conjunto variado de tests.
- **Debilidades:**
  - Orientado a Windows únicamente.
  - Incorporar nuevos tests requiere modificar el código fuente.
  - Solo se encarga de ejecutar pruebas en un único entorno.
  - La selección de tests a ejecutar se realiza mediante banderas lógicas incrustadas en el código fuente, lo que requiere recompilar el programa cada vez que se quiere especificar otro conjunto de pruebas.
  - No persiste los resultados obtenidos tras la realización de tests.
  - No dispone de interfaz gráfica.
- **SandGrox**
  - **Fortalezas:** No se ha podido determinar ninguna en base a la poca información disponible acerca del producto.
  - **Debilidades:**
    - Es solo una prueba de concepto.
    - No es una herramienta disponible públicamente.
    - El creador considera que los tests deben ser independientes de cualquier producto particular, lo cual reduce su utilidad práctica.
    - Las pruebas que su autor dice haber implementado abarcan un acotado espectro de las técnicas de anti-análisis conocidas, ya que solo se centra en aquellas cuya intención es identificar plataformas de virtualización de forma genérica.

A partir del estudio realizado, se han podido extraer estas conclusiones:

- Los desarrolladores de estos instrumentos solo han centrado sus esfuerzos en Windows; de esta forma, ignoran otros mercados como el de los servidores que ejecutan Linux, el cual es de alto riesgo debido a ser el centro de la infraestructura informática de diversas empresas. El de dispositivos móviles también es otro mercado importante, ya que en dichos aparatos suele concentrarse una gran cantidad de información personal.
- En ninguno de los productos analizados se han implementado pruebas para la detección de virtualización mediante contenedores.

- Solo InviZzible y Al-Khaser aún tienen algún grado de mantenimiento activo y presentan intenciones de crecer dentro de la comunidad destinataria.
- InviZzible es la única herramienta que permite agregar nuevas pruebas sin necesidad de modificar el código fuente.
- La mayoría de las alternativas son simplemente pruebas de concepto.
- Ningún producto ofrece interfaz gráfica y, en los casos que presentan algún tipo de persistencia de resultados, tampoco se brindan maneras de buscar esta información cómodamente bajo algún criterio. Ambas características son un importante valor agregado a la hora de trabajar seriamente con una herramienta de este estilo.
- Solo Sandbox\_tester presenta una arquitectura que habilita de alguna forma a trabajar de manera sencilla con múltiples ambientes de prueba.

Se puede apreciar entonces que el ámbito de los verificadores de transparencia de entornos destinados al análisis dinámico de software malicioso está básicamente desaprovechado debido a la falta de madurez, ambición y alcance de los productos existentes, lo cual explica por qué estos son relativamente desconocidos.

### **3.4 Visión del producto**

Las características que pudieron ser observadas en el mercado facilitaron la caracterización del sistema que finalmente fue desarrollado. El producto que es expuesto en este trabajo es Secchiware, un prototipo de verificador de transparencia de entornos de análisis dinámico de malware mediante el cual se busca ofrecer una herramienta realmente pensada para su uso profesional en el nicho del estudio de software maligno.

Secchiware plantea un instrumento mediante el cual mitigar la problemática del malware que utiliza técnicas de reconocimiento del entorno para eludir ser estudiado mediante la aplicación de estas mismas técnicas como pruebas de la transparencia de un ambiente. De esta forma, toma como misión asistir a la comunidad de investigadores en seguridad informática, administradores de sistemas y desarrolladores de sandboxes para análisis dinámico en la preparación, elaboración e implementación de mejores entornos destinados al estudio de procesos maliciosos.

El nombre se deriva del utensilio conocido como disco Secchi, el cual permite medir la turbidez de un cuerpo de agua [21]. La analogía de la función de este instrumento con la del sistema propuesto y su denominación tan llamativa se percibieron como ideales para proporcionar un título al verificador tan pertinente como atractivo.

El sistema atiende los requerimientos esperados por el mercado, pero se establecieron límites sobre algunos de ellos para simplificar su alcance en el marco de este trabajo:

- El verificador debe poder gestionar y ejecutar tests en al menos tres sistemas operativos seleccionados: Windows, Linux y Android. Dado que este conjunto abarca a nivel mundial a la mayoría de equipos computacionales activos, con asegurar el funcionamiento de Secchiware en estos sistemas es suficiente para cubrir a gran parte del mercado.
- El software contemplado debe ser capaz de ejecutar pruebas de píldoras rojas específicamente pensadas para las arquitecturas x86 y x86\_64, en las cuales se centra la bibliografía encontrada respecto al tema.

Por otro lado, se definieron las metas particulares a alcanzar con Secchiware:

- Ofrecer una interfaz gráfica para facilitar el uso prolongado de la solución propuesta y así aprovechar que ninguna solución existente en el mercado presenta esta característica.
- Brindar un sistema libre y gratuito que cualquier interesado pueda utilizar y, si así lo desease, adaptar.
- Poner el producto a disposición de cualquier interesado a través de un canal en línea.
- En sintonía con la meta anterior, también se pretende documentar debidamente el producto para facilitar su comprensión y uso.
- Escribir y distribuir el código del aplicativo, sus interfaces de usuario y su documentación en inglés para abarcar en forma general a la mayor cantidad de público posible.
- Seleccionar e implementar un conjunto representativo de técnicas de anti-análisis a partir del compendio elaborado (ver Apéndice A) con el fin de demostrar las capacidades del sistema.

## Capítulo 4: Arquitectura seleccionada

Como ya fue mencionado en el capítulo anterior, se pudo observar que es recurrente entre los investigadores de seguridad llevar a cabo pruebas para determinar las propiedades de plataformas sobre las cuales implementar entornos de análisis dinámico de malware. Típicamente la finalidad de estas investigaciones es confeccionar comparativas que destaquen las ventajas y desventajas de los productos estudiados. Incluso se ha propuesto que para incrementar la probabilidad de estudiar correctamente una muestra, la misma debe ser ejecutada en ambientes diferentes con el fin de reconocer si su comportamiento está ligado al contexto [50].

Ante esta situación, se determinó que el sistema a elaborar debería proporcionar los medios necesarios para gestionar múltiples entornos al mismo tiempo. Con esto se buscó facilitarle al usuario la propagación de tests entre diferentes ambientes, como a su vez la recuperación de los reportes generados tras la ejecución de los mismos. Este aspecto del sistema resultó ser el más crítico en cuanto a arquitectura se refiere y por ende fue el que condicionó mayormente a la misma.

La arquitectura finalmente planteada se inspira en la variedad de malware que al ser inicializado informa a un servidor central, denominado servidor de comando y control [15] (abreviado como servidor C2 o servidor C&C), que se encuentra activo y a la espera de sus órdenes. En la Figura 3 se presenta el diagrama de componentes asociado y en la Figura 4 un diagrama de un posible despliegue genérico. Son distinguibles tres subsistemas principales, cuyas comunicaciones son llevadas a cabo mediante el protocolo HTTP:

- **Nodo:** se trata del software que se ejecuta dentro del entorno que se desea verificar. Se distinguen cuatro subcomponentes:
  - **Inicializador:** Al arrancar un nodo, el mismo trata de registrarse al servidor de comando y control; en caso de éxito pone en marcha un servidor HTTP propio. Si la conexión no fue exitosa, acciona todos los tests que se encuentren instalados en su ambiente y devuelve los resultados a través de la salida estándar; tras esto, termina su ejecución.
  - **Servidor HTTP:** recibe las directrices del servidor de comando y control, para lo cual publica una API REST.
  - **Repositorio de tests:** contiene módulos de tests preinstalados al iniciar el nodo o enviados por el servidor de comando y control. Implementa las operaciones típicas de alta, modificación, baja y lectura.
  - **Ejecutor de tests:** se encarga de cargar y ejecutar las pruebas que le sean indicadas, como a su vez generar el reporte individual de cada una de ellas.

- **Servidor de comando y control:** esta es la componente que se comunica con los distintos entornos que se registren a la misma. Se pueden identificar las subcomponentes descriptas a continuación:
  - **Servidor HTTP:** expone una API REST que habilita a que los nodos se inscriban como también se den de baja del servidor; por otro lado, recibe las peticiones de componentes cliente y toma las acciones necesarias para cumplirlas, ya sea usando la información a su alcance o mandando las solicitudes correspondientes al nodo involucrado.
  - **Repositorio de tests:** es un almacén de módulos de prueba desde el cual es posible difundirlos entre los nodos asociados. Acepta operaciones básicas de alta, baja, modificación y lectura.
  - **Caché de tests:** para evitar procesar la información acerca de la composición del repositorio de tests cada vez que la misma es solicitada, se utiliza esta componente para preservarla en memoria. También resguarda la información proveniente de los almacenes de los nodos conectados, con lo cual se busca reducir la cantidad de peticiones enviadas a los entornos.
  - **Historial de sesiones:** registra los datos de los entornos brindados por los nodos durante su tiempo de actividad, además de los reportes generados tras la ejecución de pruebas en dicho periodo.
- **Cliente:** se encarga de la interacción con el usuario y de trasladar sus peticiones al servidor de comando y control. Este último se encuentra totalmente desacoplado del cliente debido al uso de un protocolo de comunicación estándar (HTTP) y a que su API REST ha sido especificada detalladamente. Esta situación permite que el cliente pueda tener diversas implementaciones paralelas; en este trabajo se han explorado dos de ellas: una basada en línea de comandos y otra construida como una interfaz gráfica destinada a usuarios de escritorio. Ambas serán abordadas en el Capítulo 5.

Los detalles de las APIs REST, tanto para un nodo como para un servidor de comando y control, han sido documentadas en formato YAML siguiendo la especificación OpenAPI [79] en su versión 3.0.2. La información allí contenida ha sido adaptada para su facilidad de lectura en este trabajo; debido a la extensión de esta documentación, la misma ha quedado plasmada en el Apéndice B.

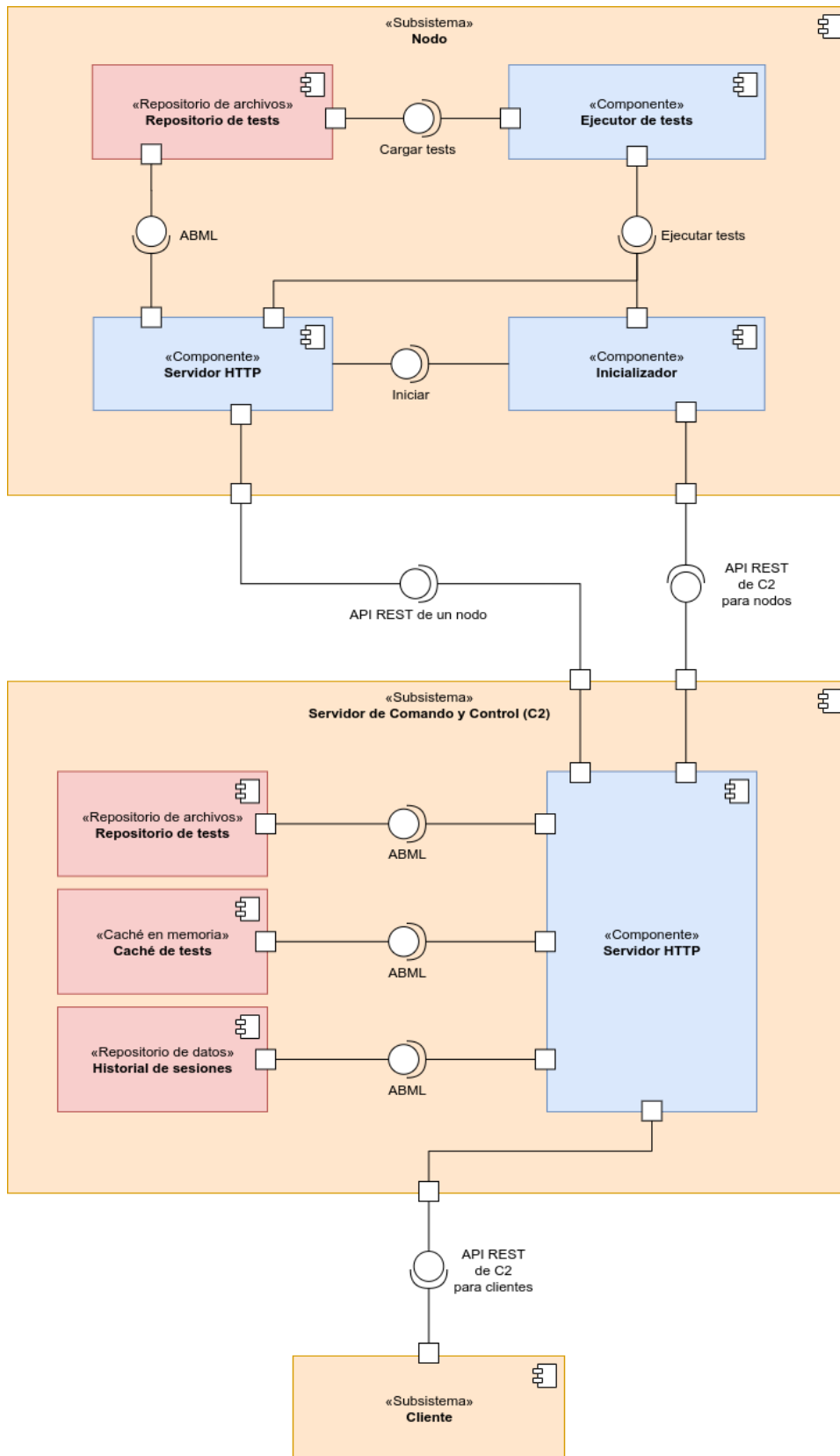


Figura 3: Diagrama de componentes

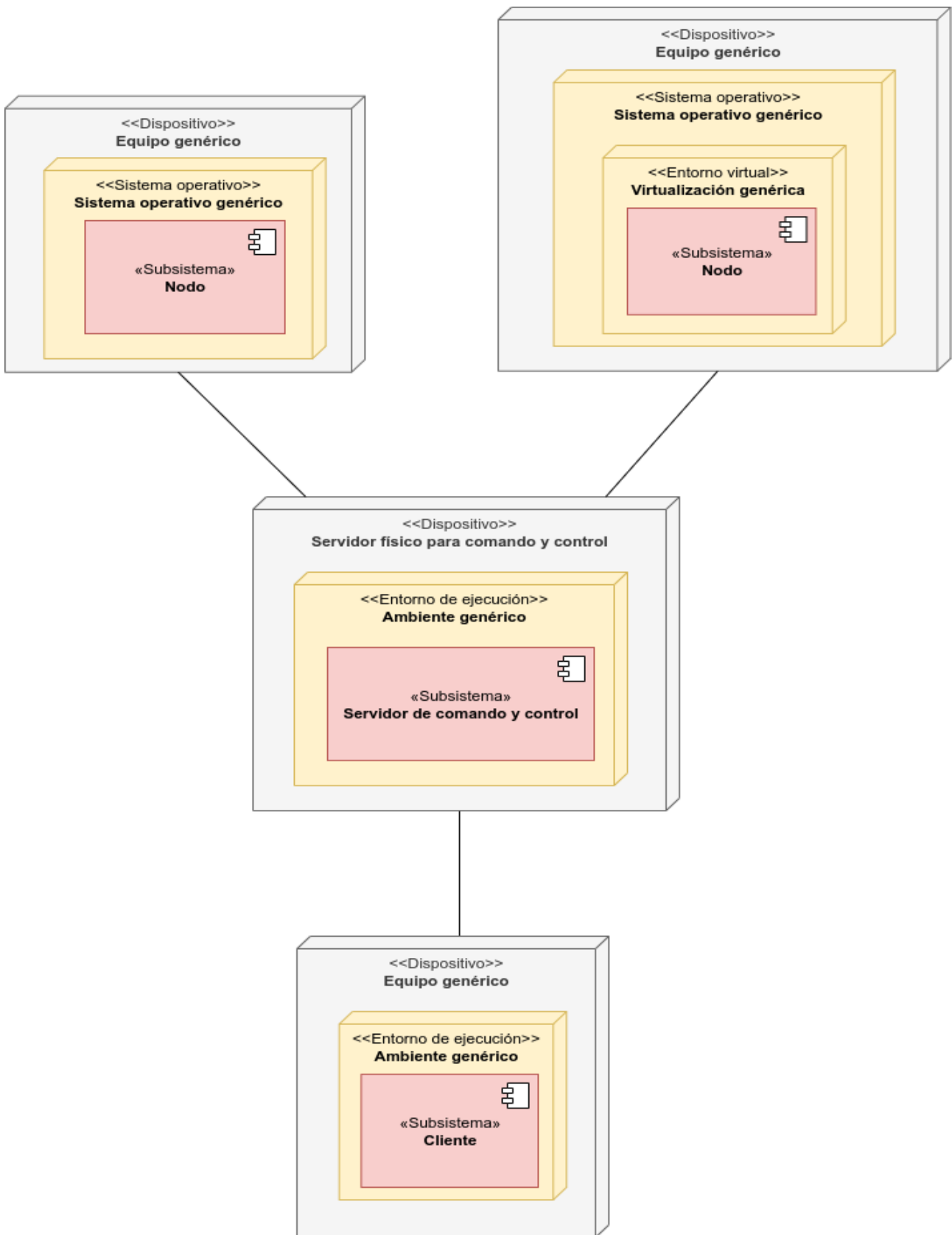


Figura 4: Diagrama de despliegue



## 4.1 Esquema de seguridad

La transmisión de código en forma de tests que finalmente podrán ser ejecutados por un nodo es una de las funcionalidades principales del software. Esto vuelve indispensable la incorporación de mecanismos de autenticación para garantizar que los mensajes críticos (aquellos que pueden cambiar el estado de una componente del sistema) provengan de una entidad autorizada.

Secchiware tiene como destinatarios pequeños grupos de trabajo operando bajo una red local, por lo que por simplicidad de uso de la herramienta se ha tenido como objetivo que la misma pueda funcionar bajo el protocolo HTTP plano. Debido a que se buscó que el software pueda autenticar de manera segura aún sin que las comunicaciones se encuentren encriptadas, se descartaron mecanismos que requirieran el envío de credenciales. Finalmente se utilizó una modalidad basada en la firma de mensajes, la cual se describe en la siguiente subsección.

### 4.1.1 SECCHWARE-HMAC-256

El esquema de firmas propuesto para este trabajo surgió tras analizar el planteado por Cavage y Sporny en la versión 12 de su borrador [12] y “Signature Version 4”, utilizado por la empresa Amazon [62]. Ambos fueron considerados demasiado complejos para la reducida escala del proyecto; sin embargo, se tomaron ciertas características de ellos para elaborar un mecanismo más simple denominado “SECCHWARE-HMAC-256”.

En el mismo se impone el uso de la cabecera “Authorization” en todo mensaje HTTP que deba ser autenticado, cuyo valor presenta el siguiente formato:

```
SECCHWARE-HMAC-256 keyId={id_clave},[headers={cabeceras}],signature={firma}
```

Consecuentemente se describen los parámetros de dicho valor:

- **keyId:** es una cadena en texto plano que ayuda a quien reciba el mensaje a identificar la clave necesaria para verificarlo.
- **headers:** es una lista de los nombres en minúscula de todas aquellas cabeceras involucradas en la firma, en el orden en el cual participan en la misma y separadas por punto y coma (;) (más adelante se describe el procedimiento de firmado).
- **signature:** la firma del mensaje codificada en base 64.

Se detalla entonces el algoritmo para generar la firma:

1. Crear una cadena de texto vacía.
2. Anexar a la cadena el método de la petición HTTP en minúsculas seguido por un carácter de fin de línea (“\n”).
3. Anexar a la cadena la URI canónica sin modificaciones de la petición HTTP (ruta a partir del host, sin incluir una posible cadena de consulta) seguida por un carácter de fin de línea (“\n”).

4. De haber una cadena de consulta, anexarla a la cadena siguiendo la codificación para URLs (con el carácter de espacio representado como '%20') seguida por un carácter de fin de línea ('\n').
5. Por cada cabecera especificada, se debe anexar a la cadena la expresión "{nombre\_de\_cabecera}: {valor\_de\_cabecera}", donde {nombre\_de\_cabecera} es la clave de la cabecera en minúscula y {valor\_de\_cabecera} es su respectivo valor sin modificaciones. A cada una de las expresiones resultantes se le debe anexar un fin de línea ('\n').
6. La última línea de la cadena generada debe omitir el carácter '\n' al final de la misma.
7. Calcular el resumen de la cadena obtenida mediante HMAC-SHA256 y codificarla en base 64 para conseguir la firma.

Todo aquel servidor que implemente esta modalidad debe rechazar una petición no autorizada mediante el código de error HTTP 401 (*Unauthorized*), incluyendo en la respuesta la cabecera "WWW-Authenticate" para indicar los requisitos para acceder al recurso pertinente.

## 4.2 Atributos de calidad

La arquitectura propuesta fue pensada con la intención de satisfacer ciertas características definidas en la norma ISO/IEC 25010 [39], las cuales fueron determinadas como críticas en el presente proyecto. A continuación se enuncian los atributos definidos como de mayor importancia y se comenta brevemente cual es su influencia en el diseño presentado:

- **Eficiencia de desempeño:** es necesario que la componente del nodo cuide la **utilización de recursos** para poder funcionar correctamente en entornos con limitaciones.
- **Compatibilidad:** al tratarse de un sistema distribuido, la **interoperabilidad** de sus componentes es un factor crucial. Es por ello que se decidió aplicar un protocolo de comunicación altamente difundido y probado (HTTP) y se especificaron en detalle los endpoints y formatos para intercambio de información.
- **Usabilidad:** el cliente por línea de comandos es un medio simple para interactuar con el sistema, aunque no es adecuado para largas sesiones de trabajo. Con la incorporación de un cliente con interfaz gráfica se busca facilitar su **aprendizaje** y su **uso**.
- **Seguridad:** la confidencialidad no es un aspecto relevante ya que en el sistema planteado la información más crítica que se transmite son módulos contenedores de tests y los resultados de su consecuente ejecución. Sin embargo, la incorporación dinámica de nuevas pruebas implica que los nodos son capaces de

ejecutar código arbitrario. Esto convierte en prioritaria la implementación de un mecanismo de **autenticación** y verificación de **integridad** de mensajes que solo permita la ejecución de acciones críticas (aquellas que modifican el estado del sistema) solicitadas por entidades autorizadas y que defiendan de ataques por parte de intermediarios. Con este fin se diseñó el esquema de verificación de mensajes “SECCHIWARE-HMAC-256” explicado previamente.

- **Mantenibilidad:** la separación del producto en componentes bien definidas le brinda una gran **modularidad**, dado que estas solo están acopladas en base a las APIs que exponen.
- **Portabilidad:** es imprescindible que el nodo se **adapte** efectivamente a distintos sistemas operativos y arquitecturas de computadoras para poder realizar pruebas en entornos diversos. Para el alcance particular de este proyecto debe ser compatible mínimamente con los sistemas Linux, Android y Windows y con las arquitecturas x86\_64 y ARM, al menos a alto nivel.

A su vez, se busca que cualquier interesado en el producto pueda modificar y elaborar su propia versión de las partes que lo componen, de acuerdo a sus preferencias y requisitos particulares. Como ejemplo de esta **capacidad de reemplazo** se proponen dos implementaciones de clientes alternativas. Esta característica también es parte de los incentivos por los cuales se detallan exhaustivamente las APIs de un nodo y de un servidor de comando y control.

## Capítulo 5: Implementación y distribución

Se exponen aquí las decisiones que fueron tomadas a lo largo del desarrollo del producto, se describen las características y procesos detrás de las partes que lo componen y finalmente se informa como se ha puesto a disponibilidad del público.

### 5.1 Lenguaje de programación

Una de las decisiones más tempranas y críticas a tomar al principio del desarrollo fue la elección del lenguaje de programación mediante el cual se implementaría la componente del nodo que se alojaría en los distintos entornos a probar. Se consideraron tres lenguajes candidatos, los cuales son presentados a continuación junto a una breve justificación de por qué fueron tomados en cuenta en primera instancia:

- **C**, por ser el lenguaje compilado por excelencia, lo que evita depender de un entorno de ejecución.
- **Python**, por su popularidad y su carácter interpretado, lo que ayuda a prototipar y probar rápidamente las soluciones implementadas.
- **Java**, dado que es uno de los lenguajes de programación principales para Android.

Para seleccionar la alternativa adecuada, se determinó que los candidatos debían ser evaluados bajo los siguientes criterios:

- **Popularidad:** uno de los objetivos del proyecto era proporcionar un mecanismo para facilitar el desarrollo de nuevas pruebas. Elegir un lenguaje que goce de buena popularidad influiría positivamente en el atractivo del producto y su alcance en la comunidad destinataria del producto.
- **Facilidad de uso:** esta característica fue tomada en cuenta también con miras a favorecer la elaboración de tests propios por parte de los usuarios. A su vez, desde la perspectiva del desarrollo de sistemas informáticos es un aspecto que tiende a reducir los errores a la hora de programar.
- **Capacidades de bajo nivel:** existen diversas técnicas de reconocimiento de entornos que aprovechan peculiaridades de la arquitectura del procesador, el manejo de memoria, la implementación del sistema operativo, entre otros (algunas fueron descritas en la clasificación presentada en este trabajo). Es por esto que resultaba importante que el lenguaje escogido brinde herramientas para poder operar a bajo nivel.
- **Librerías disponibles:** debido a que el proyecto aquí presentado iba a ser encarado por una sola persona, un aspecto deseable del lenguaje a optar era que contara con un rico ecosistema de librerías maduras y probadas que aliviaran la tarea de desarrollar el producto propuesto.

- **Compatibilidad con Android:** de entre los tres sistemas escogidos para encarar el desarrollo del prototipo (Windows, GNU/Linux y Android), el sistema operativo para smartphones resulta ser el más crítico en este apartado. Los tres lenguajes bajo evaluación se usan sin inconvenientes en el ámbito de escritorio; esto no sucede en Android, dado que sus principales herramientas de desarrollo (y que permiten operar fácilmente con el sistema) están destinadas para Java y Kotlin.

Finalmente, a cada lenguaje se lo puntuó en base a cada una de las características anteriores. La escala aplicada va del número 1 al 5, en la cual 1 representa un extremadamente pobre desempeño en el aspecto correspondiente, mientras que un 5 indica que el lenguaje se destaca ampliamente en el mismo. La evaluación realizada sobre cada una de las alternativas ha sido detallada en las Tablas 1, 2 y 3.

Como puede apreciarse, Python fue el lenguaje seleccionado principalmente por su difusión y buen recibimiento por parte de la comunidad de programadores, características que pueden repercutir de manera favorable a la popularidad que tome el verificador de transparencia. A su vez, el desarrollo de este producto se vería agilizado gracias a su carácter interpretado, algo que congenia adecuadamente con el prototipado de soluciones, a su biblioteca estándar y a su variada colección de paquetes de terceros fácilmente descargables. Particularmente, se escogió dar soporte a las versiones del lenguaje a partir de la 3.6 debido a que en la misma se incorporaron los “f-strings” [40], una manera simple y de alto rendimiento para generar cadenas formateadas.

Por otro lado, C se tomó como lenguaje secundario con el fin de asistir en la elaboración de pruebas a bajo nivel, aprovechando así la facilidad de Python para integrarse con él mediante el módulo *ctypes*.

**Tabla 1: Evaluación del lenguaje C**

<b>Criterio</b>	<b>Detalle</b>	<b>Puntaje</b>
Popularidad	Aunque se trata de uno de los lenguajes más difundidos históricamente, hoy en día se aplica principalmente a desarrollos a nivel hardware o sistema operativo.	3
Facilidad de uso	Presenta la ventaja de ser un lenguaje estáticamente tipado y por ende gran parte de los errores derivados de este concepto son atrapados en tiempo de compilación; pero, su sistema de conversión de tipos resulta muy permisivo, por lo que un uso incorrecto del mismo puede llevar a fallos confusos durante la ejecución. A su vez, no incluye implementaciones de estructuras de datos de alto nivel de abstracción y la gestión de memoria debe ser aplicada manualmente por el programador; esto último es una fuente de errores comúnmente cometidos por la comunidad.	2
Capacidades de bajo nivel	Por su bajo nivel de abstracción, su manejo manual de memoria y por permitir insertar código ensamblador C se ha destacado históricamente en este aspecto.	5
Librerías disponibles	La librería estándar de C es bastante limitada, más en comparación con otros lenguajes. Tampoco existe un gestor de paquetes oficial mediante el cual importar módulos desarrollados por otros programadores. Sin embargo, es posible encontrar varios proyectos de código abierto en repositorios como GitHub.	2
Compatibilidad con Android	El NDK de Android [56] es un conjunto de herramientas que facilitan la compilación de módulos de C y C++ para la plataforma. Es importante destacar que el sistema operativo utiliza su propia versión de la biblioteca estándar de ambos lenguajes, lo cual hay que tener en consideración al enlazarla a la hora de compilar programas. Sin embargo, no se provee una API nativa en estos lenguajes para acceder directamente a funcionalidades particulares del sistema operativo, tal como el envío de SMS, lo cual debería realizarse con llamados a la API de Java mediante JNI [42].	3
<b>Puntaje total</b>		<b>15</b>

Tabla 2: Evaluación del lenguaje Python

Criterio	Detalle	Puntaje
Popularidad	Python ha sido en los últimos años el lenguaje predominante en el área de la inteligencia artificial. Más allá de eso, su sencillez y su rico ecosistema de librerías le han dado difusión en otros entornos y lo han impulsado a ser uno de los lenguajes más populares de hoy en día.	5
Facilidad de uso	Aunque es un lenguaje dinámicamente tipado, realiza muy pocas conversiones implícitas y al momento de tratar de realizar una operación en donde se encuentra involucrado un dato incompatible genera excepciones esclarecedoras de este hecho. Esto sumado a una sintaxis sencilla con características que permiten resolver ciertos problemas en pocas líneas de código (al menos en comparación con otros lenguajes), una biblioteca estándar muy rica, gestores de paquetes simples que permiten incorporar fácilmente programas de otros desarrollares y su sencillez para prototipar y probar debido a su carácter interpretado lo han llevado a ser uno de los lenguajes más amenos para la comunidad.	5
Capacidades de bajo nivel	Los lenguajes interpretados, por su naturaleza, no suelen brindar facilidades para operar a nivel hardware; Python no es la excepción. A pesar de ello, el módulo de la biblioteca estándar <i>ctypes</i> [18], el cual se encuentra en varias implementaciones del intérprete, permite la carga de librerías escritas en C e invocar a las funciones contenidas en ellas como si hubieran sido escritas en Python. Esto potencialmente puede ser aprovechado para desarrollar algunas pruebas puntuales.	3
Librerías disponibles	Este es uno de los aspectos más reconocidos de Python. Como ya fue mencionado previamente, no solo su biblioteca estándar es extensa, sino que además cuenta con una gran colección de paquetes de terceros fácilmente disponibles a través los gestores existentes.	5
Compatibilidad con Android	Existen varias formas de tener un intérprete funcional en Android. De entre todas ellas se destaca QPython [64], una IDE adaptada al sistema operativo móvil que cuenta con un intérprete y que incluye la librería SL4A [72], la cual habilita interactuar transparentemente con la API de Android directamente desde ciertos lenguajes interpretados.	4
<b>Puntaje total</b>		<b>22</b>

**Tabla 3: Evaluación del lenguaje Java**

<b>Criterio</b>	<b>Detalle</b>	<b>Puntaje</b>
Popularidad	Java ha sido ampliamente adoptado desde sus inicios, lo que lo llevó a incursionar en una gran variedad de plataformas. Hoy en día sigue siendo extremadamente popular en los ámbitos empresarial y educativo; sin embargo, la aparición de alternativas que resultan más agradables para la comunidad ha disminuido su utilización. Sus detractores opinan que es un lenguaje verboso y dogmático, con una estructura de proyectos excesivamente compleja.	4
Facilidad de uso	Es un lenguaje estáticamente y fuertemente tipado, por lo que muchos de sus controles se realizan en tiempo de compilación y suele dar información útil para identificar errores en tiempo de ejecución. A su vez su mecanismo de conversión de tipos es más estricto que el de C por la construcción misma del lenguaje, aunque su abuso aún puede derivar en errores de ejecución. A su vez, su fuerte adhesión a ciertos principios de diseño y de programación orientada a objetos impone hasta cierto punto un estilo de programación ordenado; sin embargo, esto en ocasiones puede llevar a necesitar implementar soluciones más complejas de lo que deberían ser para solventar ciertas limitaciones.	4
Capacidades de bajo nivel	La Máquina Virtual de Java (JVM) fue diseñada originalmente para brindar una abstracción del hardware a los programadores que posibilitara escribir código multiplataforma. En consecuencia, el lenguaje no brinda medios para operar a bajo nivel. Sin embargo, para invocar código nativo se podría utilizar JNI [42] o, como una alternativa más simple, JNA [41], una librería comunitaria de código abierto análoga al módulo <i>ctypes</i> de Python.	3
Librerías disponibles	El lenguaje cuenta con una rica biblioteca estándar. Sin embargo, su gestor de paquetes (entre otras funciones) más difundido, Maven, es una herramienta moderadamente compleja de usar y aunque cuenta con un repositorio principal, en la práctica existen muchos otros paralelos que ofrecen sus propias librerías.	3
Compatibilidad con Android	Java es uno de los lenguajes para el cual se desarrolla y mantiene activamente una API para interactuar con el sistema operativo.	5
<b>Puntaje total</b>		<b>19</b>



## 5.2 Back-end web

Tras haber elegido un lenguaje, el siguiente aspecto a establecer de manera prioritaria fue el framework para desarrollo del back-end, dado que tanto las componentes del servidor de comando y control como del nodo se diseñaron como servidores HTTP. En el entorno de Python los ejemplares más conocidos de estas tecnologías son Django [22] y Flask [28].

El primero de estos frameworks es una solución que incluye una gran cantidad de características orientadas al desarrollo de un sitio web en sus totalidad, las cuales van desde la interacción con la base de datos hasta la conformación y entrega de documentos HTML. Tomando en consideración que las dos componentes ya mencionadas simplemente debían implementar una API REST en la cual el formato de las respuestas fuera JSON, la variedad de funcionalidad que no iba a ser aprovechada y la complejidad de Django hicieron que rápidamente fuera descartado como candidato.

Por otro lado, Flask es mucho más acotado (en su documentación oficial lo catalogan de “microframework”), aunque puede ser expandido a través de extensiones según lo requiera el interesado. Esto lo convierten en una alternativa liviana, flexible (a diferencia de Django que es dogmático en su diseño) y fácil de usar y aprender, propiedades sumamente apetecibles en el contexto de este proyecto y por las cuales fue escogido finalmente. La versión de este framework aplicada al prototipo fue la 1.1.1.

Una particularidad de ambas alternativas es que implementan el estándar WSGI [24] (*Web Server Gateway Interface*). El mismo describe una interfaz mediante la cual un servidor web y una aplicación web pueden comunicarse a pesar de sus características particulares y trabajar en conjunto para procesar solicitudes. La principal fortaleza de esto es que permite abstraerse del servidor de producción en cual se desplegará finalmente una aplicación al momento de escribirla. Gracias a ello, la componente del servidor de comando y control programada no impone una solución de despliegue particular, sino que se le da la libertad de elegirla a sus implementadores. Este no es el caso de los nodos, los cuales solo deberían responder consultas del servidor central en la práctica y por ende operan con el servidor HTTP interno de Flask, que aunque su uso esté orientado solo a pruebas resulta más que suficiente en este contexto.

Por otro lado, se tomó en consideración que los navegadores web de hoy en día, por defecto, aplican una política del mismo origen [61] restrictiva. Es por ello que fue necesario que en el servidor de comando y control se habilite el Intercambio de Recursos de Origen Cruzado (*CORS* [16], por sus siglas en inglés); sin esto, un cliente implementado con tecnologías web bloquearía las comunicaciones con el susodicho servidor. Para incorporar CORS se utilizó la extensión para Flask llamada Flask-Cors [27] en su versión 3.0.8.

### 5.3 Base de datos

La decisión acerca del sistema de gestión de bases de datos a utilizar en el servidor de comando y control fue tomada rápidamente. La simpleza del modelo de entidad-relación necesario para persistir la información de los entornos y sus reportes (solo tres entidades fueron contempladas) y la reducida escala del sistema (el cual fue pensado para ser usado por un pequeño grupo de personas dentro de una red local) hicieron de SQLite [84] una solución más que apropiada. Además, Python brinda soporte para este gestor desde su librería estándar mediante el módulo *sqlite3* [77], con lo cual se evitó incorporar una dependencia externa adicional al producto. En la Figura 5 se puede observar el diagrama de entidad-relación que refleja la implementación final de la base de datos.

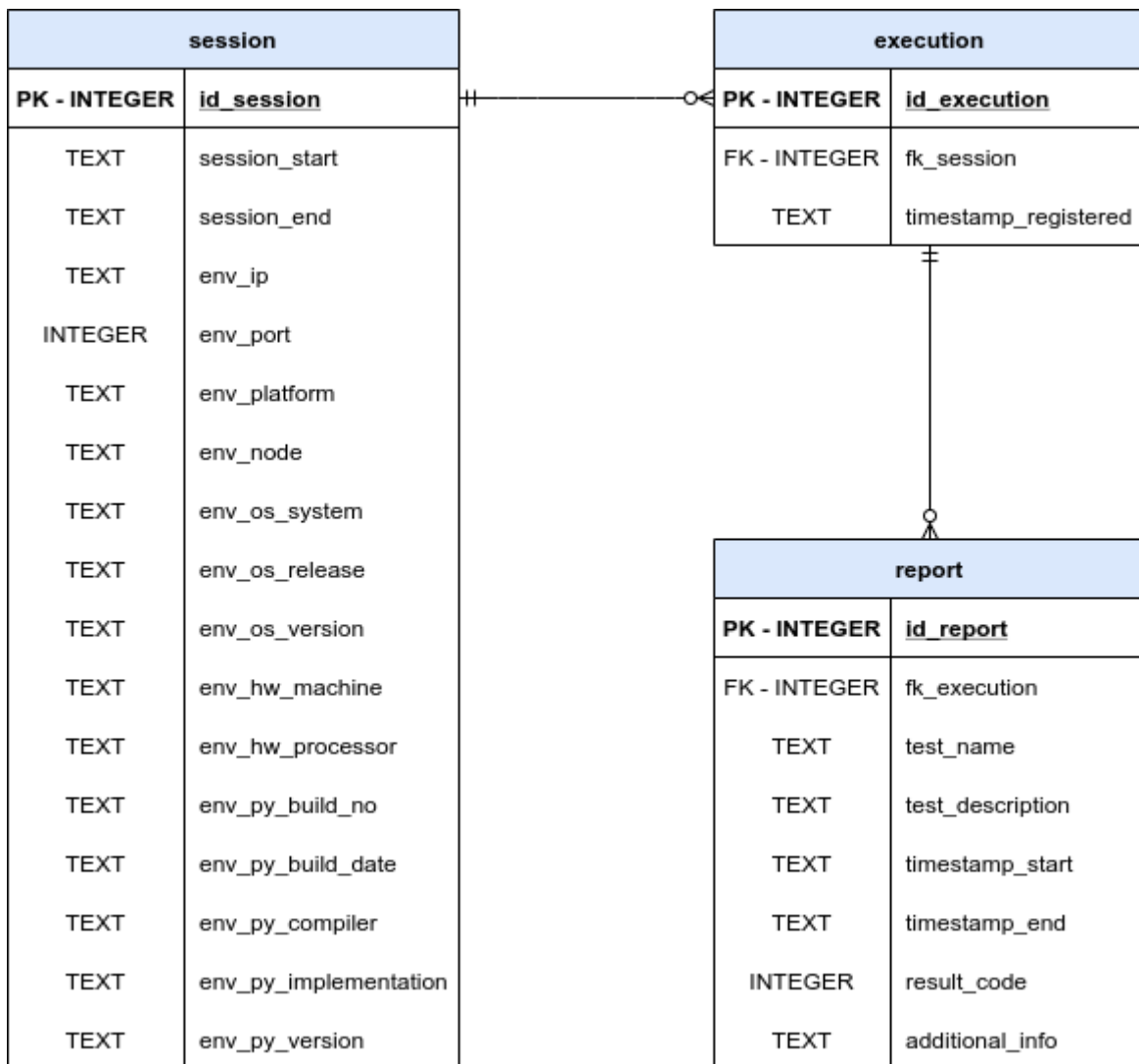


Figura 5: Diagrama de entidad-relación

La entidad “session” representa el periodo de funcionamiento de un nodo en un entorno, desde su arranque (“session\_start”) hasta su suspensión (“session\_end”); este último campo se mantiene en nulo mientras el nodo se mantenga activo. La IP y el puerto

mediante los cuales el nodo se mantuvo escuchando por la llegada de peticiones del servidor de comando y control está representados por los campos “env\_ip” y “env\_port”. “env\_node” es el nombre de la máquina (virtual o física) desde la que se ejecutó la componente. Los atributos que comienzan con “env\_os” alojan información acerca del sistema operativo del entorno, los que empiezan con “env\_hw” respecto al modelo y arquitectura de su procesador y los que inician con “env\_py” resguardan datos sobre la versión de Python y del intérprete particular usados para ejecutar al nodo.

“execution” es el grupo de reportes que fueron solicitados por una llamada particular al endpoint encargado de la ejecución de pruebas. “timestamp\_registered” es el momento en que se volcó la respuesta devuelta en la base. Tal como indica la relación correspondiente, para una sesión dada pueden existir cero o más ejecuciones adjudicadas (asociadas mediante el atributo de clave foránea “fk\_session”).

Los reportes están descritos por “report”. “timestamp\_start” y “timestamp\_end” son el instante de comienzo y de fin del test; “result\_code” es un valor numérico que refleja el éxito (número positivo), fracaso (número negativo) o indeterminación (cero) de la prueba; y el campo “additional\_info” contiene una cadena JSON con la información arbitraria que el desarrollador de la prueba haya querido incluir junto al código de resultado. Por supuesto, el nombre y la descripción del test se almacenan en los atributos “test\_name” y “test\_description”. Los reportes pertenecientes a una ejecución se identifican a través del atributo “fk\_execution”, el cual toma la función de clave foránea.

Cabe mencionar que todas las fechas se almacenan en UTC como cadenas en formato RFC 3339 [57]. Se plantearon otros esquemas que ocupasen menos espacio de almacenamiento, pero debido a que se buscaba soportar que los tiempos de inicio y fin de una prueba pudieran estar descritos con cualquier precisión de fracción de segundo tolerada por el ambiente de ejecución en cuestión, finalmente se decidió utilizar el formato ya descrito. El mismo también se aplicó a otros campos que no requieren de dicho nivel de detalle para mantener homogéneo el manejo de fechas.

Se agregó un índice filtrado a la tabla “session” sobre los campos “env\_ip” y “env\_port” y condicionado a que los registros asociados tengan un valor nulo en el campo “session\_end”. El fin de esta incorporación es la de agilizar las consultas respecto a nodos activos, de las cuales dependen una gran cantidad de rutas del servidor de comando y control.

Para concluir, se menciona que la política de eliminación tomada es la de propagación en cascada, ya sea que se parta desde una sesión o desde una ejecución.

## 5.4 Implementación, ejecución y almacenamiento de tests

Como ya fue mencionado previamente, la capacidad de poder incorporar nuevas pruebas desarrolladas por sus usuarios se definió como una de las características principales del prototipo de verificador de transparencia. Para dicho fin, se creó un módulo

de Python denominado “test\_utils”; el mismo brinda facilidades para la creación de tests como a su vez la gestión e inspección del repositorio correspondiente.

El mecanismo aplicado es semejante a la implementación típica de extensiones. Se diseñó una clase abstracta “TestSet” en la que se definieron métodos esenciales para la ejecución de pruebas y, particularmente, un decorador para especificar que un método es un test (los decoradores de Python son un mecanismo mediante el cual se envuelve una función dentro de otra con la intención de extender su funcionalidad).

El decorador programado solo acepta métodos que devuelvan o un entero o una tupla compuesta de un entero y de un diccionario (que corresponden al código de resultado de la prueba y a la información adicional opcional que el desarrollador quiera incluir tal como indica la estructura “TestReport”, ver Apéndice B). También recibe de forma obligatoria el nombre y la descripción de la prueba y se encarga de consultar la fecha y hora antes y después de la ejecución del test de forma transparente. Con la información mencionada el decorador se encarga de generar el diccionario que representa el reporte correspondiente a la concreción de la prueba y el método original se ve modificado devolviendo ahora dicho diccionario.

El programador puede definir sus propias clases derivadas de “TestSet” agrupando pruebas según le parezca y desarrollando los métodos correspondientes a las mismas. De esta manera, solo debe concentrarse en los algoritmos a implementar y en las características definitorias del test en sí. Estas clases pueden almacenarse en módulos de Python y estos a su vez en paquetes que pueden contener otros subpaquetes. En la Figura 6 se presenta uno de los tests desarrollados para demostrar el potencial del verificador, el cual ilustra los aspectos descriptos previamente.

Para cargar y ejecutar pruebas se proporcionó la clase “TestSetCollection” que, como su nombre indica, alberga un grupo de clases derivadas de “TestSet”. La misma posibilita especificar las pruebas a ejecutar a cuatro niveles de granularidad: todos los tests de un paquete, los de un módulo, los de una clase o simplemente algunas pruebas específicas. Cabe decir que las clases solo se instancian al momento de la ejecución y se realiza un control durante su importación para evitar duplicados. Finalmente, “TestSetCollection” contiene un método que permite realizar todas las pruebas que hayan sido seleccionadas y que devuelve los reportes correspondientes.

El módulo “test\_utils” contiene otras funciones auxiliares que permiten inspeccionar paquetes de Python contenedores de pruebas para generar la representación de su contenido, la cual está especificada por la estructura “TestPackageInfo” (ver Apéndice B). También incluye funciones para comprimir y descomprimir un conjunto de estos paquetes en formato tar.gz, lo que brinda asistencia durante la transmisión de estos a un servidor de comando y control o a un nodo.

Estas dos últimas componentes tienen destinada una carpeta que se especifica como un paquete de Python en la cual se almacena el código referido a los tests. De esta manera, dicho directorio funciona como el paquete raíz desde el cual pueden ser

importadas las pruebas, ya sea para inspeccionarlas como para instanciarlas y ejecutarlas. Esta importación es dinámica; la librería “test\_utils” hace uso del módulo estándar *importlib* [37] para tal fin.

```
1  from test_utils import TestResult, TestSet
2
3
4  class X86VirtualizationSet(TestSet):
5
6      @TestSet.test(
7          name="Does the product name correspond to VirtualBox or VMWare?",
8          description=
9              "Checks if the content of '/sys/class/dmi/id/product_name' "
10             "matches against 'VirtualBox' or 'VMWare'.")
11     def product_name(self) -> TestResult:
12         with open("/sys/class/dmi/id/product_name", "r") as f:
13             product_name = f.read().rstrip()
14             additional_info = {
15                 'found_product_name': product_name
16             }
17             if product_name in {'VirtualBox', 'VMWare'}:
18                 result = TestSet.TEST_FAILED
19             else:
20                 result = TestSet.TEST_PASSED
21             return result, additional_info
```

Figura 6: Ejemplo de implementación de pruebas

## 5.5 Caché

El proceso de inspección de las pruebas instaladas explicado previamente presenta un costo computacional que no debe ser despreciado. El mismo requiere importar todos los módulos de Python de un paquete indicado (y sus subpaquetes), recuperar todas las clases derivadas de “TestSet” que en ellos se encuentra y por cada una de estas encontrar los métodos marcados como tests. Esto derivó en la necesidad de implementar algún tipo de caché en el servidor de comando y control con la intención de preservar esta información una vez conformada y así agilizar su consulta.

Cabe decir que no se requiere la aplicación de un caché en los entornos sobre los que se ejecutan las pruebas debido a que el servidor de comando y control hace de intermediario entre clientes y nodos, lo que le permite respaldar la información en juego de todas las componentes asociadas a él. Con esto no solo evita incurrir en un consumo de recursos mayor en los nodos, sino que además se reducen las comunicaciones necesarias entre el servidor y estos ya que las consultas de los clientes por los tests instalados pueden ser resueltas sin ser propagadas.

Las siguientes características fueron establecidas como deseables en una implementación particular del caché en cuestión:

- Residir en RAM, con lo cual evitar usar espacio de almacenamiento en disco para almacenar información que no interesa mantener entre sesiones de actividad del servidor de comando y control.
- Tener la granularidad de un paquete de pruebas raíz definido por un usuario; de tal manera se agiliza la inserción, actualización o eliminación del caché asociado a un paquete si el mismo es alterado en el repositorio.
- Ser compartido entre procesos. Esto surge como consecuencia de que las aplicaciones desarrolladas con Flask pueden ser desplegadas en cualquier servidor que implemente la interfaz WSGI. Los servidores de esta índole que podemos encontrar y sus formas de responder peticiones concurrentes son diversas; algunas de estas son la aplicación de una arquitectura orientada a eventos en la que un hilo es capaz de gestionar múltiples consultas de forma asíncrona, la utilización de múltiples hilos en donde cada uno se encarga de atender una solicitud a la vez o su equivalente usando procesos. Esta última es la más crítica; una solución de caché compatible con servidores multiproceso es compatible a su vez con los demás por su misma naturaleza.
- Ser funcional tanto en Linux como en Windows.

A priori se consideró implementar una solución embebida (y que no requiriera introducir nuevas dependencias) en el servidor de comando y control mediante SQLite, el cual tiene la posibilidad de crear y gestionar bases de datos temporales en memoria. Sin embargo, las mismas solo pueden ser accedidas desde el proceso que las creo ya que residen en su memoria asignada.

Se decidió entonces buscar otras tecnologías disponibles que cumplieran con los requisitos listados. Acto seguido se encontraron las dos herramientas más comúnmente utilizadas como sistemas de caché en RAM: Memcached [54] y Redis [66]. Ambas son soluciones de código abierto que siguen una arquitectura cliente-servidor, multiplataforma y que alojan contenido en estructuras de datos clave-valor, aspectos que satisfacen todas las características esperadas.

La selección de la alternativa a utilizar fue rápida y concisa. Mientras que Memcached solo puede organizar su información bajo el esquema clave-valor, Redis soporta otros tipos de datos estructurados que resultan útiles para diversos casos de uso, tal como la confección de un índice sobre las claves presentes. Es por esto que se eligió a Redis como repositorio en memoria para implementar el caché buscado.

Para trabajar con Redis desde Python se utilizó la librería externa *redis* [67] en su versión 3.5.2. Al iniciar el servidor de comando y control el caché se inicializa con los paquetes que ya se encontraban en el repositorio. Los paquetes instalados en un nodo particular se vuelcan recién tras la primera petición que los consulte por parte de un

cliente; este es el único momento en que el servidor debe comunicarse con el nodo para poder entregar esta información, en solicitudes consecuentes le basta solo con recuperarla del caché.

Los cambios posteriores en los repositorios se reflejan en el caché impactando exclusivamente contra las claves de los paquetes involucrados. En el caso de la instalación de nuevas pruebas en los entornos provenientes del repositorio de un servidor de comando y control, no es necesario que se inspeccionen en los nodos correspondientes, dado que la información de su estructura ya fue generada y cacheada en dicho servidor.

En la Tabla 4 se ha plasmado el modelo del almacén en Redis que utiliza Secchiware como caché.

Tabla 4: Detalle de las claves del caché en Redis

Clave	Tipo	Descripción
<b>repository:{paquete}</b>	String	Contiene la representación en JSON de un paquete raíz del repositorio.
<b>repository_index</b>	Sorted Set	Índice con los nombres de los paquetes en el repositorio, ordenado alfabéticamente.
<b>environments:{ip}:{puerto}</b>	Hash	Contiene información acerca del entorno en la dirección provista por <i>ip</i> y <i>puerto</i> . Sus campos son: <ul style="list-style-type: none"> <li>• <b>installed_cached</b>: guarda “0” (falso) o “1” (verdadero) dependiendo de si los paquetes instalados en el nodo ya han sido incorporados al caché.</li> <li>• <b>installed:{paquete}</b>: resguarda la representación en JSON de un paquete raíz del repositorio.</li> </ul>
<b>environments:{ip}:{puerto}:installed_index</b>	Sorted Set	Índice con los nombres de los paquetes instalados en el entorno en la dirección provista por <i>ip</i> y <i>puerto</i> , ordenado alfabéticamente.

## 5.6 Control de concurrencia

Una vez implementada toda la funcionalidad del servidor de comando y control, se analizó el flujo de información que potencialmente podría darse si dos peticiones se atendieran simultáneamente. Esto conllevó a encontrar ciertas condiciones de carrera descritas a continuación:

- Si dos o más clientes consultan al mismo tiempo las pruebas instaladas en un nodo cuyo caché aún no ha sido inicializado, se producen peticiones redundantes que deben ser enviadas al entorno y se repiten escrituras con la misma información en el repositorio de Redis.

- Si se instalan paquetes en un nodo provenientes del almacén del servidor de comando y control mientras este está siendo actualizado, los paquetes efectivamente enviados pueden estar malformados debido a cambios a medias. Esto conlleva también a que el caché correspondiente al entorno, que se actualiza con la información sobre los módulos de prueba que ya están presentes en el servidor, quede inconsistente.

Esta condición de carrera también puede generar malestar entre los usuarios del sistema debido a que al tratar de actualizar el repositorio de un nodo indicando ciertos paquetes con un determinado contenido en un momento dado, este no sea el que quede instalado finalmente.

- Si dos o más solicitudes que buscan introducir cambios en el repositorio de pruebas de un nodo son atendidas concurrentemente, no hay garantía del orden en que se envían los comandos de actualización de los datos a Redis.
- Una condición análoga a la anterior se produce en el servidor de comando y control respecto a su almacén de tests. Sin embargo, en este caso no solo es el caché lo que puede quedar en un estado indeseado, sino que además los paquetes en el repositorio presentan el riesgo de terminar malformados.

Si se toma en cuenta que el producto está dirigido a equipos de trabajo pequeños bajo una misma red local y por ende la posibilidad de que se den las anomalías descritas es baja, podría parecer innecesaria la aplicación de mecanismos de control de concurrencia. Pero, de suceder, la correctitud funcional del sistema se vería notoriamente perjudicada y, en consecuencia, la experiencia de los usuarios. Es por esto que el riesgo subyacente no debía ser ignorado y por ello se decidió incorporar bloqueos para corregir la situación.

Como ya fue explicado, el servidor de comando y control de Secchiware está pensado para ser desplegado en cualquier servidor listo para producción compatible con la interfaz WSGI. Para control de concurrencia nuevamente el caso crítico de despliegue es el de múltiples procesos atendiendo solicitudes en simultáneo y nuevamente fue Redis la herramienta a utilizar en la solución.

El repositorio en memoria, además de ser naturalmente compatible con un modelo multiproceso por su arquitectura cliente-servidor, brinda entre su funcionalidad básica la capacidad de establecer tiempos de vida a las claves que en él se registren. Si se considera el riesgo de que un proceso que adquirió un bloqueo finalice inesperadamente antes de poder liberarlo y en consecuencia el recurso asociado se torne inaccesible, poder indicar un tiempo máximo de vigencia al bloqueo resulta sumamente idóneo.

El módulo de Python *redis* utilizado para operar con la herramienta homónima tiene definidas de por sí ciertas componentes de software para establecer bloqueos de exclusión mutua. Este tipo se consideró suficiente para proteger los repositorios de los distintos nodos, ya que se concluyó que la lectura de aquellos no era crítica y no



necesitaba de traba alguna. De esta manera, solo las regiones de código que modifican de alguna manera el contenido en los entornos fueron aseguradas.

Resulta llamativo destacar la secuencia de operaciones en el caso particular de la consulta por la información en el almacén de un nodo, recordando que el caché relacionado se crea tras la primera petición pertinente. El algoritmo codificado indica que al atender una solicitud primero debe verificarse si el caché ya fue inicializado, en cuyo caso el contenido allí presente se usa para conformar la respuesta. De no ser así, se intenta conseguir el mutex asociado; si esta operación tiene éxito se envía la petición al entorno correspondiente, se actualiza el almacén en memoria en base a la respuesta recibida, se deja indicado que se inició el caché, se libera el bloqueo y se retransmite la respuesta al cliente. En caso de no haber podido obtener el mutex, la solicitud es suspendida por un tiempo antes de volver a verificar si la bandera de inicialización ya ha sido puesta en verdadero y en su defecto tratar nuevamente de conseguir el bloqueo. La razón de ser del ciclo descrito es que puede darse la situación de que la petición que primero obtuvo el mutex tenga algún inconveniente que la haga finalizar tempranamente sin llegar a introducir cambios en Redis; es por ello que las peticiones consecuentes no pueden asumir que el caché estará inicializado una vez que el bloque sea liberado. En la Figura 7 se ha representado el diagrama de flujo simplificado de la secuencia explicada.

Por otra parte, se concluyó que era necesario bloquear en los casos en los que se alteren los paquetes existentes en el repositorio del servidor central, como a su vez cuando es requerida la lectura de este almacén al enviar módulos de tests desde aquel a un nodo (ver endpoint GET /environments/{ip}/{port}/installed del Apéndice B). Esta última situación tiende a ser común al trabajar con entornos nuevos y por ende presenta una probabilidad alta de concurrencia. Ante esto, un mutex aplicado también a la lectura sería una solución poco elegante; así fue como se decidió aplicar un boqueo lector-escritor, más apropiado al caso.

La librería *redis* no incluye mecanismos de este tipo, por lo cual fueron codificadas algunas clases de Python que subyacentemente utilizan la librería mencionada para construir el modelo de bloqueo buscado. En el caso de un lector, este primero obtiene un identificador otorgado por Redis (el mismo se trata de un valor entero que se incrementa con cada solicitud). Luego, debe conseguir el mutex correspondiente al recurso asociado (de no hacerlo, espera un tiempo determinado y vuelve a intentarlo). Después se anota en un conjunto ordenado (Sorted Set) de lectores del susodicho recurso presente en el almacén en memoria, en el cual se vuelca tanto el identificador obtenido como el instante en que este registro expirará (valor por el cual se organiza la estructura de datos), calculado a partir de la fecha y hora actual y un tiempo de vida preestablecido. Tras esto, el lector libera el mutex, ejecuta la región crítica de código a la que buscaba acceder y finalmente elimina su entrada del conjunto ordenado de lectores. La secuencia descrita ha sido ilustrada en la Figura 8.

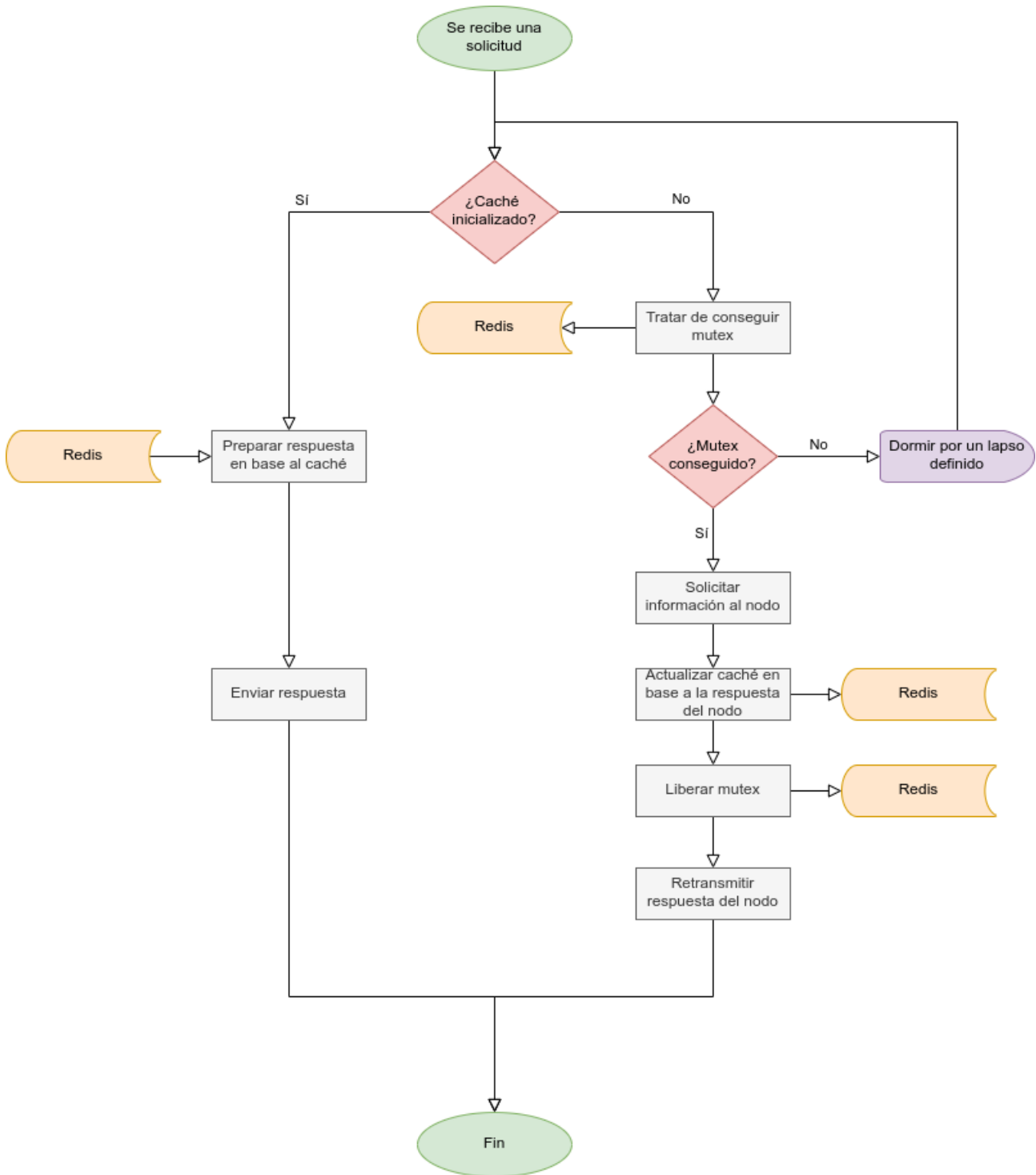


Figura 7: Secuencia simplificada de consulta del repositorio de pruebas de un nodo

Como contraparte, lo primero de lo que se encarga un escritor es de eliminar de dicha estructura de datos todo registro que ya haya expirado. Seguidamente comprueba que no hayan quedado lectores activos en el conjunto y trata conseguir el mutex asociado al recurso solicitado. En caso de obtenerlo, ejecuta las operaciones críticas correspondientes y libera el bloqueo. En el caso de que aún persistan algunos lectores o que el mutex ya esté tomado, el escritor duerme un tiempo definido, nuevamente remueve

todas las entradas expiradas del conjunto de lectores y reitera la verificación y el intento de obtención ya descritos, ciclo que perdura mientras que no se pueda conseguir el bloqueo. Este flujo ha quedado plasmado en la Figura 9.

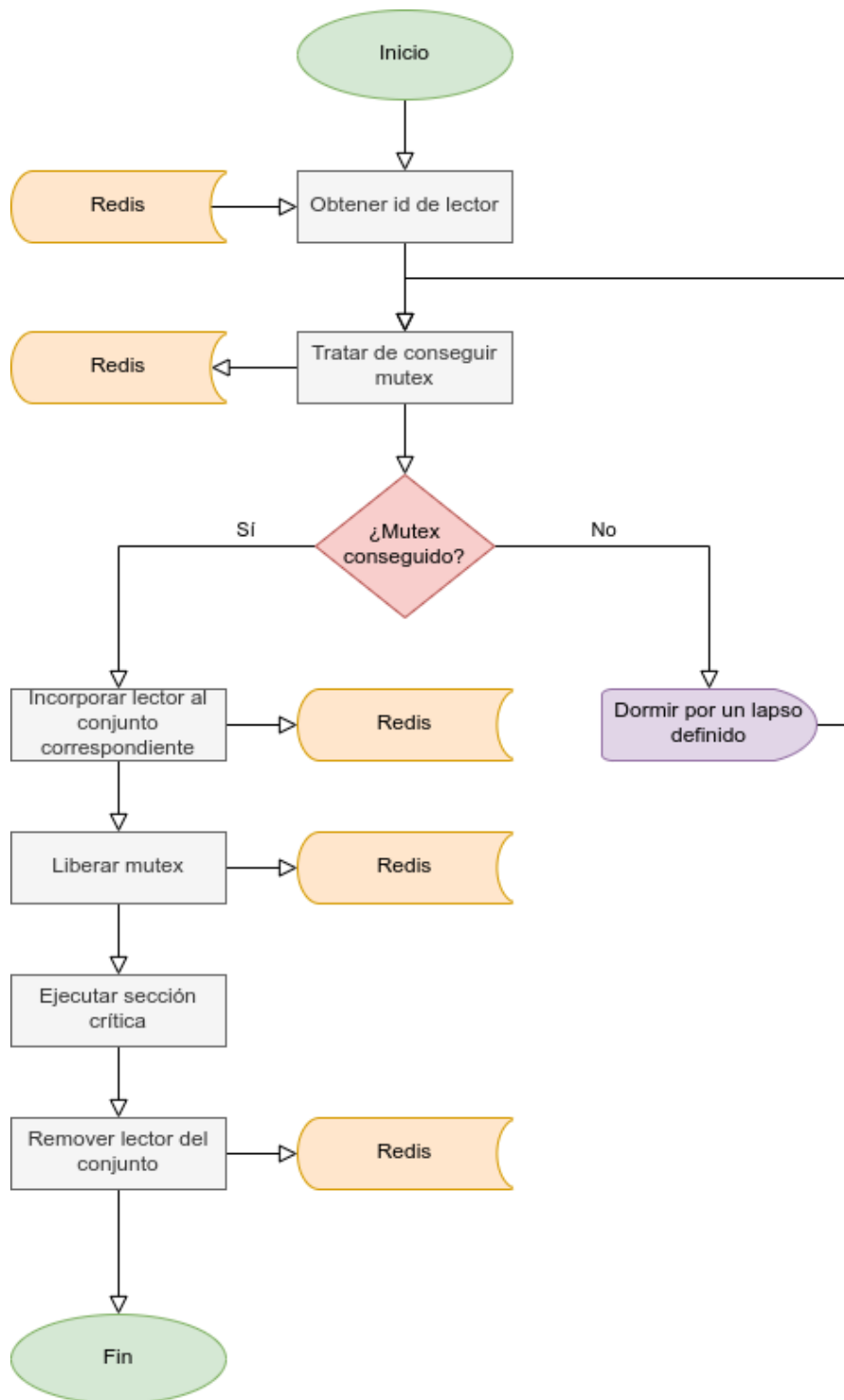


Figura 8: Secuencia de operaciones de un lector

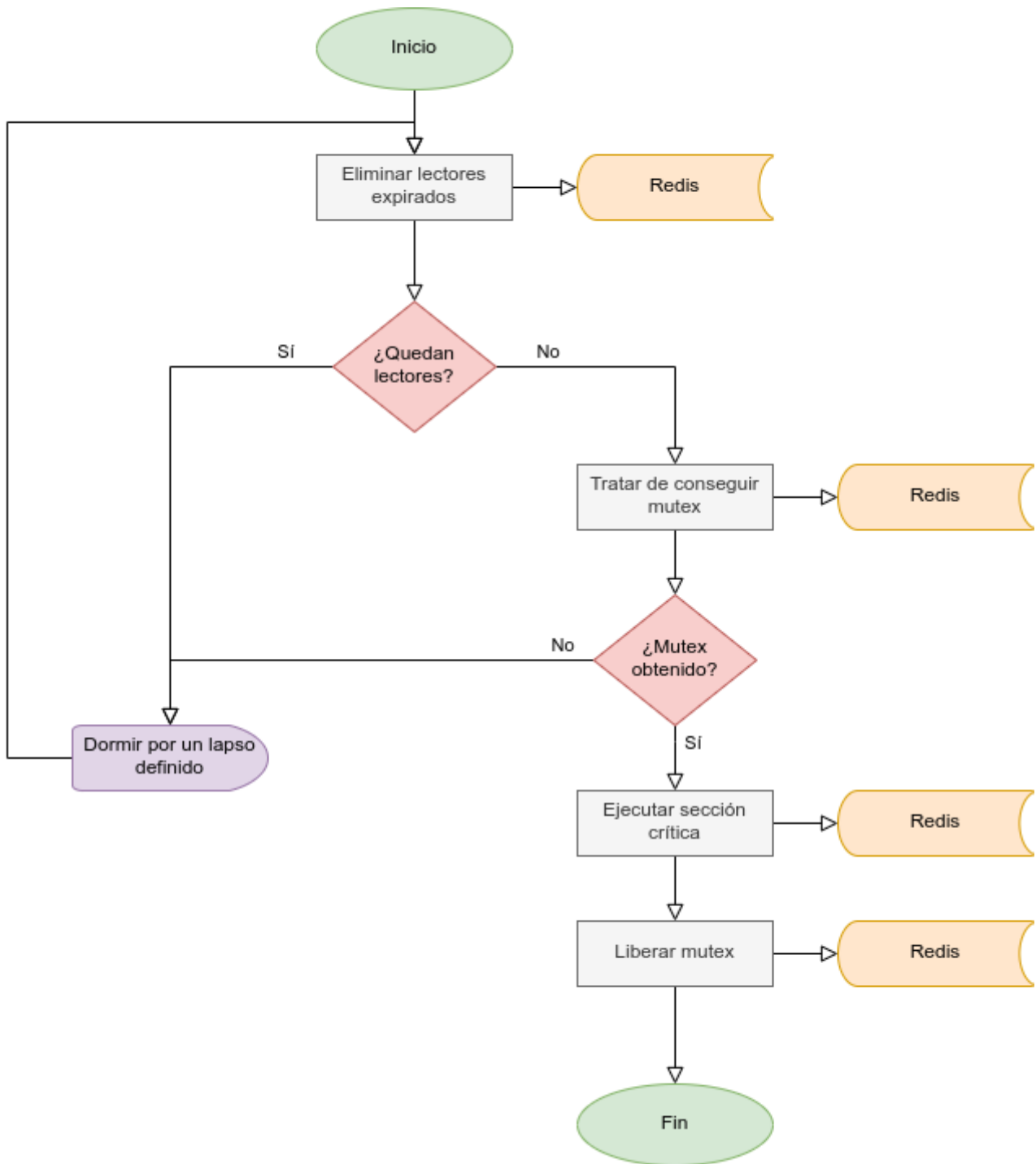


Figura 9: Secuencia de operaciones de un escritor

La fecha de expiración de los registros en el conjunto de lectores no es un dato caprichoso: de no incluirse, un lector que por algún motivo haya quedado detenido o haya fallado sin llegar a darse de baja del conjunto conllevaría a la inanición de los escritores. Cabe mencionar también que el esquema aplicado favorece a los lectores, ya que mientras uno o más escritores se encuentran esperando a que estos dejen de retener el recurso, más pueden seguir apareciendo.

Por otra parte, en todos los casos en que una solicitud debe esperar a conseguir un bloqueo, no importa de qué tipo, se implementó un sencillo mecanismo basado en dormir y volver a intentarlo posteriormente. Soluciones más sofisticadas capaces de notificar a una secuencia de ejecución particular requerirían introducir una sobrecarga injustificada tomando en consideración que el nivel de concurrencia esperado es bajo y por ende el tiempo de espera probable para conseguir un bloqueo también lo es.

## 5.7 Cliente por línea de comandos

Para la confección de esta componente se aprovechó la librería *click* [13] en su versión 7.0. La misma está especializada en la creación de interfaces por línea de comandos y es una dependencia de Flask, por lo que ya estaba incluida en el proyecto cuando se tomó la decisión de utilizarla.

La elaboración y transmisión de mensajes HTTP a un servidor de comando y control se implementó mediante el módulo *requests* [69] en su versión 2.22.0 (cabe mencionar que el mencionado subsistema también hace uso de este para enviar solicitudes a los distintos entornos).

*click* cuenta con facilidades para documentar el código escrito y con ello producir automáticamente las típicas ayudas de las herramientas por línea de comandos, lo cual fue aprovechado para este producto. La documentación acerca de los comandos disponibles como así sus opciones y argumentos se ha adaptado en el Apéndice C.

## 5.8 Cliente gráfico

Desde que se planteó la elaboración de una interfaz gráfica para operar con el verificador de transparencia, la intención fue la de brindar una experiencia similar a la de una aplicación de escritorio. Sin embargo, el creador de este trabajo no contaba con antecedentes de implementación de este tipo de programas, aunque sí en desarrollo web. Dado que hoy en día herramientas como Electron [25] permiten crear programas de escritorio con tecnologías web en los que se embebe el entorno de ejecución necesario para su funcionamiento, esta vía fue determinada como la más adecuada ya que permitía aprovechar la experiencia del autor.

Debido a que la componente de un servidor de comando y control ya gestiona y entrega la información necesaria para que un cliente pueda operar cómodamente, se buscó evitar introducir un back-end adicional para entregar el contenido de la interfaz gráfica. De esta forma, toda la lógica de presentación está autocontenida en el cliente y este solo necesita consumir la API del servidor central especificado para recuperar los datos necesarios.

Para ello se decidió utilizar algún framework de front-end con la intención de facilitar la composición y renderización de los elementos que conformarían al cliente. Las tres alternativas más reconocidas en este ámbito son Angular [4], React [65] y Vue [83]. El

autor de este proyecto nunca había trabajado con ninguna de ellas, por lo cual fue necesario recurrir a comparativas que explicaran las virtudes y limitaciones de cada una. Particularmente se destaca el análisis realizado por la compañía Binariks que refleja las opiniones más difundidas acerca del tópico [6]. El producto escogido terminó siendo Vue, debido a su destacada facilidad de aprendizaje, a su carácter intrínsecamente reactivo y a su forma de definir componentes de un sistema web mediante módulos, en cada uno de los cuales se especifican la plantilla HTML, el estilo descrito en CSS y el comportamiento programado en Javascript de una componente particular.

Para asemejar lo más posible el cliente a una aplicación de escritorio nativa, este fue planteado como una aplicación de página única (*single-page application*). Una vez iniciada, todas las transiciones entre las pantallas que la conforman son efectuadas mediante la carga dinámica y reemplazo de componentes definidas usando Vue. Por otro lado, los datos que el sistema muestra son recuperados de un servidor C&C gracias a la API estándar y asíncrona Fetch [26].

Adicionalmente se usó BootstrapVue [9], una reimplementación del popular framework Bootstrap [8] a través de componentes de Vue, con el fin de simplificar la distribución del contenido visual de la interfaz, como también así su estilizado. Por último, se usó Vue CLI Plugin Electron Builder [48] para transformar fácilmente el desarrollo realizado en una aplicación de Electron.

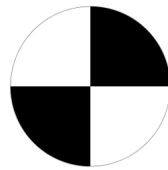
El cliente gráfico finalmente obtenido, el cual fue denominado como Secchiware Desktop Client, presenta una apariencia en la que predominan los colores blanco y negro, la cual refleja una temática basada en es el aspecto clásico de un disco Secchi.

## **5.8.1 Pantallas**

Se describen brevemente en esta sección las distintas vistas que componen al cliente gráfico.

### **5.8.1.1 Bienvenida**

Al iniciar Secchiware Desktop Client lo primero que se le da la bienvenida al usuario con un pequeño formulario en donde debe especificar la URL del servidor de comando y control con el que pretende trabajar y la contraseña de cliente especificada para el mismo, tal como puede verse en la Figura 10. El logo que en ella se expone es un disco Secchi, lo que hace honor al nombre del producto.



Welcome to Secchiware Desktop Client!

Specify a Command and Control server:

URL  
http://127.0.0.1/5000

Password

Start!

*Figura 10: Pantalla de bienvenida de Secchiware Desktop Client*

### **5.8.1.2 Gestión del servidor de comando y control**

Tras ingresar los datos del servidor central, el usuario es derivado inmediatamente a la vista principal de la aplicación. La misma consta de una distribución conformada por una barra de navegación a la izquierda con accesos a los tres paneles que engloban la funcionalidad del cliente, los cuales se renderizan a la derecha de la susodicha barra.

El primero de estos paneles es el que permite configurar nuevamente los parámetros correspondientes al servidor C&C con el cual se opera; a su vez proporciona elementos para consultar y gestionar su repositorio de pruebas. Esta pantalla se ilustra en la Figura 11. En la misma, puede observarse que la información de configuración, enmarcada dentro de una tarjeta, ya está establecida con los datos que se ingresaron en la bienvenida. Esto revela que, a diferencia de lo que sucedía con el cliente CLI, ese contenido conforma parte del estado del proceso y no es necesario que el usuario vuelva a ingresarlo cada vez que requiera efectuar una operación.

Por otro lado, también se aprecia una tarjeta destinada a asuntos pertinentes al almacén de tests que tiene tres pestañas: una para visualizar su composición (que se obtiene mediante el endpoint GET /test\_sets, ver Apéndice B), otra que permite subir un archivo tar.gz para ampliar o modificar su contenido (a través del endpoint "PATCH /test\_sets", ver Apéndice B) y finalmente una para remover paquetes (haciendo uso del endpoint "DELETE /test\_sets/{package}", ver Apéndice B).

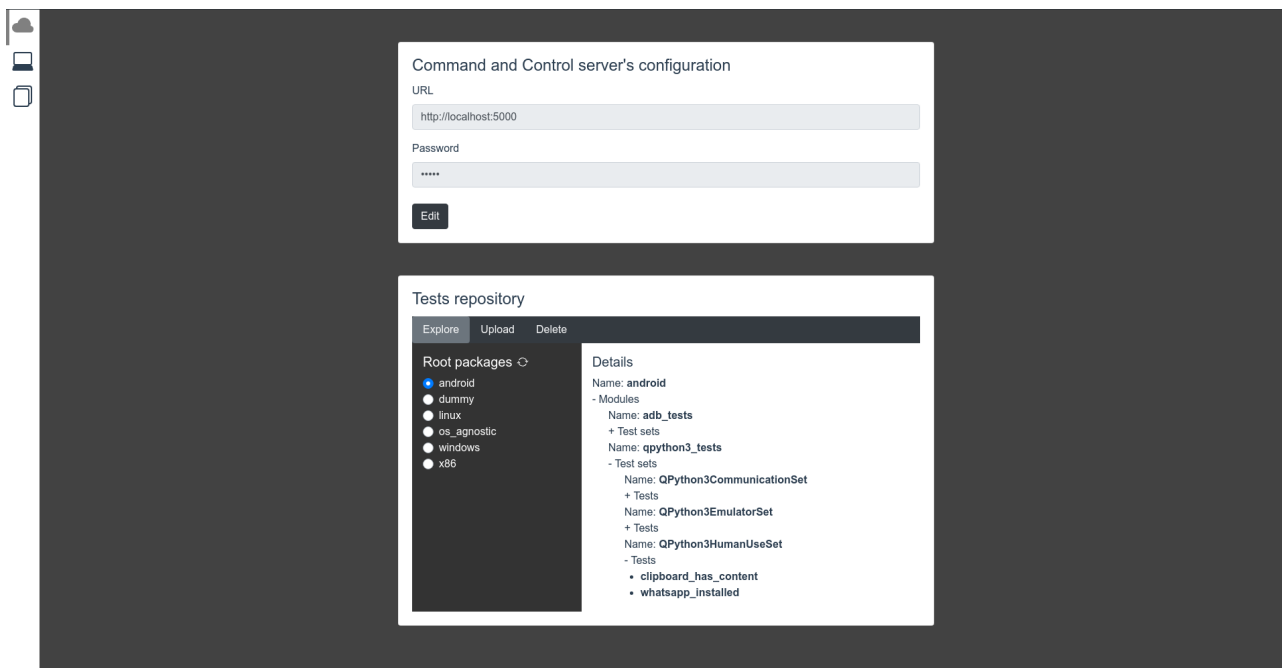


Figura 11: Pantalla de gestión del servidor de comando y control

### 5.8.1.3 Entornos

El segundo panel principal del sistema es el que reúne aspectos que involucran directamente a un nodo en funcionamiento, el cual se ha visto reflejado en la Figura 12. A la izquierda se halla una columna que contiene un selector para elegir el entorno con el cual trabajar.

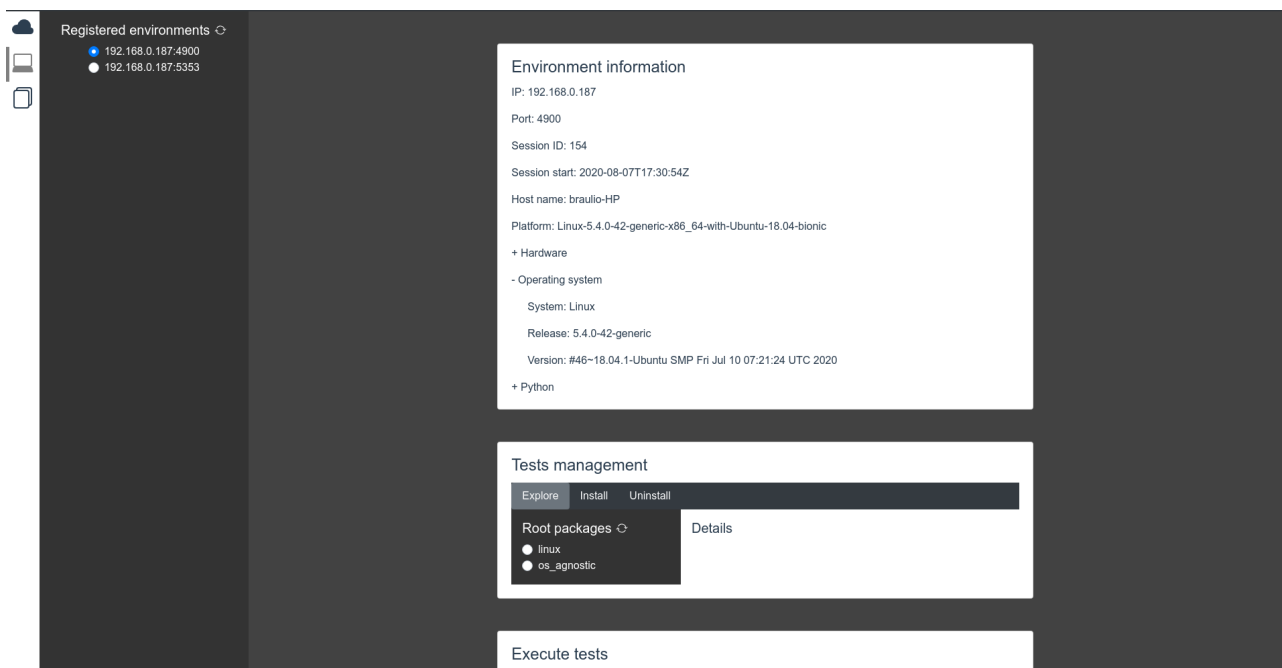


Figura 12: Pantalla de operatoria con nodos



La primera tarjeta que se aprecia al elegir un entorno recopila su información tal como es devuelta por el endpoint “GET /environments/{ip}/{port}/info” (ver Apéndice B). Tras esta se encuentra una tarjeta para consultar y gestionar el repositorio de pruebas particular en el entorno. Las componentes que conforman las pestañas “Explore” y “Uninstall” son compartidas con la tarjeta equivalente de la pantalla anterior, lo único que cambia son los datos de entrada que se les provee a las mismas (endpoint “GET /environments/{ip}/{port}/installed” y endpoint “DELETE /environments/{ip}/{port}/installed/{package}” respectivamente, ver Apéndice B). La pestaña “Install” sí que es distinta, ya que por cómo está definido el endpoint “PATCH /environments/{ip}/{port}/installed” del servidor de comando y control para instalar pruebas en un nodo (ver Apéndice B), estas solo pueden provenir del almacén de la entidad central.

La tercera y última tarjeta de este panel trabaja con el endpoint “GET /environments/{ip}/{port}/reports” (ver Apéndice B). Presenta dos pestañas: una para seleccionar las pruebas a ejecutar y otra para visualizar los reportes con los resultados obtenidos. La implementación de la primera, que se puede apreciar en la Figura 13, es digna de mención.

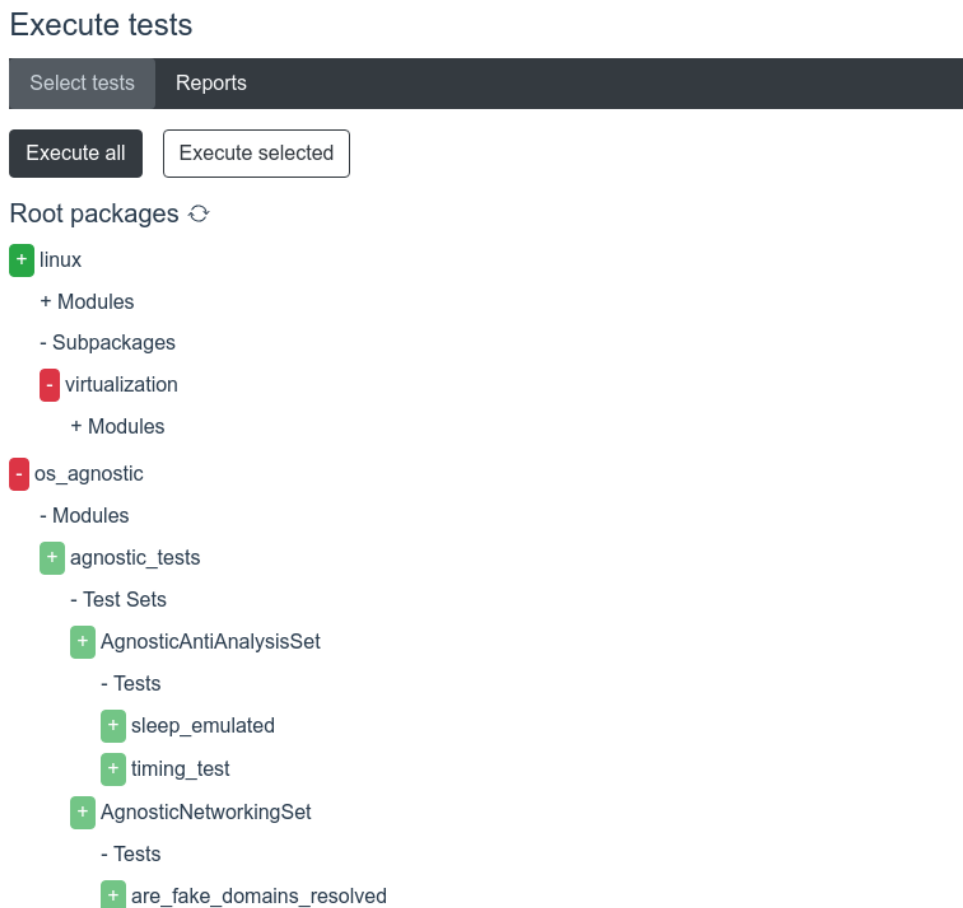
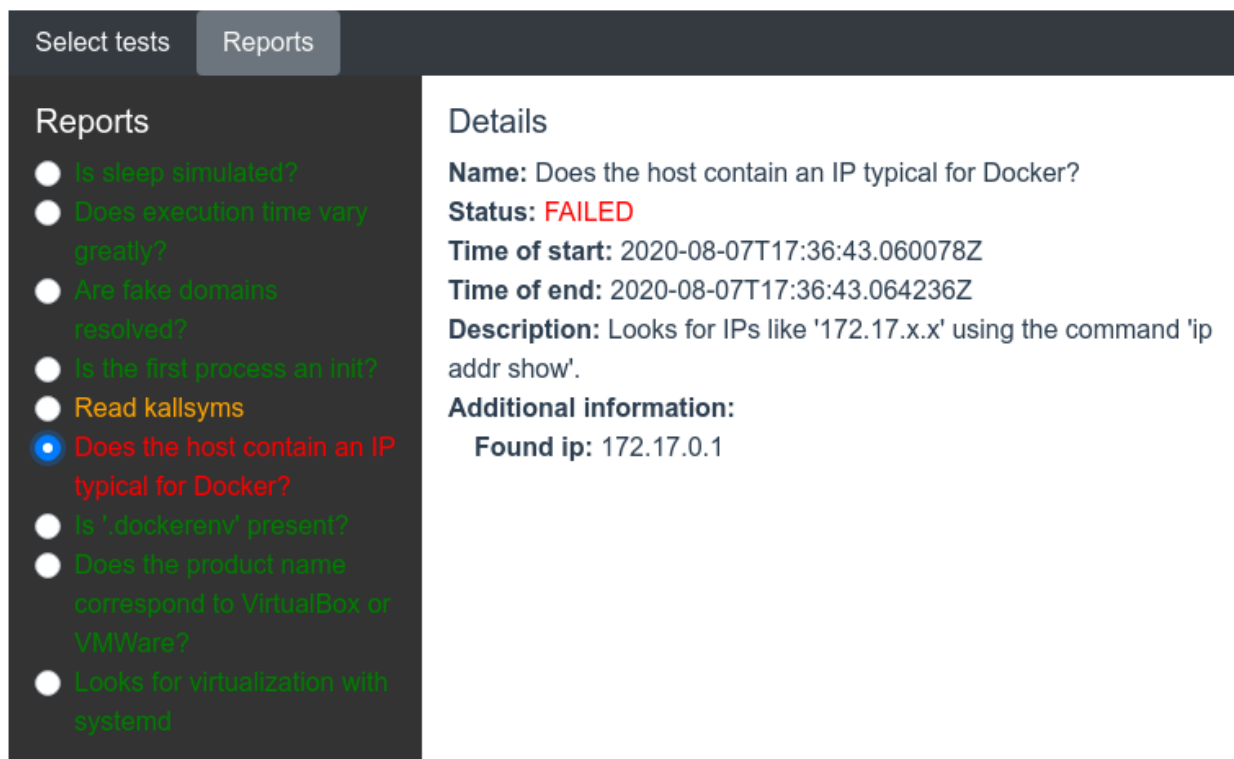


Figura 13: Selector de pruebas a ejecutar

Aquella permite seleccionar pruebas a los cuatro niveles mencionados previamente (paquete, módulo, conjunto y test particular) mediante un menú de árbol expandible. Para llevar el registro de las entidades elegidas se utilizaron cuatro colecciones, una por cada tipo. Al seleccionar un elemento de un nivel macro todos sus descendientes son deshabilitados en el árbol y son removidos de las colecciones correspondientes con el fin de evitar enviar información duplicada al servidor de comando y control; por supuesto, todos los descendientes son nuevamente habilitados cuando se desmarca la opción superior.

La otra pestaña, la de visualización de reportes, consta de un menú a la izquierda que lista los nombres de los tests realizados y a través del cual se puede escoger uno de ellos para consultar su detalle, el cual es expuesto en el panel derecho, tal como aparece en la Figura 14. Los colores verde, rojo y naranja con los que se exhiben los nombres de las pruebas indican si las mismas fueron exitosas, fallaron o no pudieron ser determinadas respectivamente.

## Execute tests



The screenshot shows a web interface with two tabs: 'Select tests' and 'Reports'. The 'Reports' tab is active. On the left, there is a list of reports with radio buttons. The report 'Does the host contain an IP typical for Docker?' is selected and highlighted in red. On the right, the details for this report are shown:

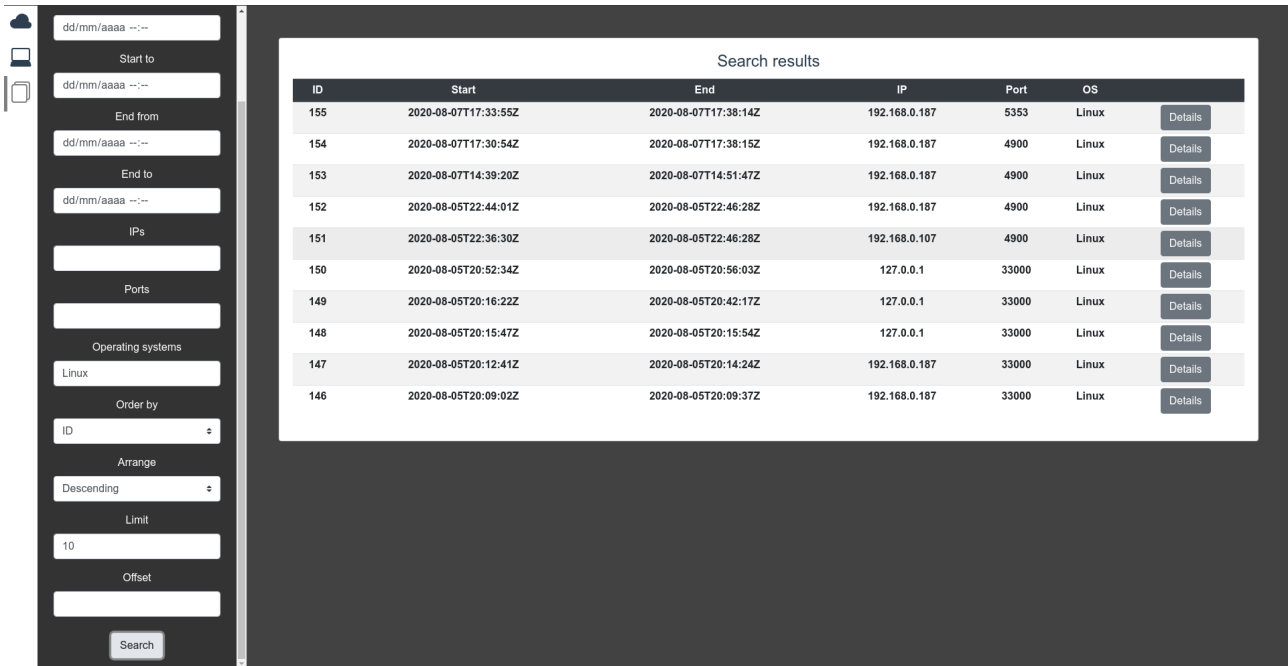
**Details**  
**Name:** Does the host contain an IP typical for Docker?  
**Status:** FAILED  
**Time of start:** 2020-08-07T17:36:43.060078Z  
**Time of end:** 2020-08-07T17:36:43.064236Z  
**Description:** Looks for IPs like '172.17.x.x' using the command 'ip addr show'.  
**Additional information:**  
Found ip: 172.17.0.1

Figura 14: Reportes de una ejecución

### 5.8.1.4 Búsqueda y consulta de sesiones

El panel final consta de dos subpaneles; el primero de ellos está destinado a la búsqueda de sesiones mediante el endpoint "GET /sessions" (ver Apéndice B). Como puede observarse en la Figura 15, los campos a completar para filtrar las búsquedas se

encuentran en una columna a la izquierda del panel y reflejan los parámetros aceptados por el ya referido endpoint. A la derecha, los resultados son devueltos como filas de una tabla, en la cual se exhibe un conjunto de datos mínimo de la sesión correspondiente.



The screenshot displays a search results subpanel. On the left is a sidebar with search filters: Start to, End from, End to, IPs, Ports, Operating systems (set to Linux), Order by (set to ID), Arrange (set to Descending), Limit (set to 10), and Offset. The main area shows a table titled "Search results" with columns: ID, Start, End, IP, Port, OS, and a Details button. The table contains 10 rows of session data.

ID	Start	End	IP	Port	OS	Details
155	2020-08-07T17:33:55Z	2020-08-07T17:38:14Z	192.168.0.187	5353	Linux	Details
154	2020-08-07T17:30:54Z	2020-08-07T17:38:15Z	192.168.0.187	4900	Linux	Details
153	2020-08-07T14:39:20Z	2020-08-07T14:51:47Z	192.168.0.187	4900	Linux	Details
152	2020-08-05T22:44:01Z	2020-08-05T22:46:28Z	192.168.0.187	4900	Linux	Details
151	2020-08-05T22:36:30Z	2020-08-05T22:46:28Z	192.168.0.107	4900	Linux	Details
150	2020-08-05T20:52:34Z	2020-08-05T20:56:03Z	127.0.0.1	33000	Linux	Details
149	2020-08-05T20:16:22Z	2020-08-05T20:42:17Z	127.0.0.1	33000	Linux	Details
148	2020-08-05T20:15:47Z	2020-08-05T20:15:54Z	127.0.0.1	33000	Linux	Details
147	2020-08-05T20:12:41Z	2020-08-05T20:14:24Z	192.168.0.187	33000	Linux	Details
146	2020-08-05T20:09:02Z	2020-08-05T20:09:37Z	192.168.0.187	33000	Linux	Details

Figura 15: Subpanel de búsqueda de sesiones

Tras pulsar el botón “Details” de una fila se traspasa al subpanel de consulta de la sesión. Aquel contiene una tarjeta para visualizar la información del entorno asociado (obtenida del endpoint “GET /sessions/{session\_id}”, ver Apéndice B) que fue confeccionada reaprovechando componentes de la tarjeta equivalente de la pantalla de nodos y a la que se le ha agregado un botón para eliminar la sesión de la base de datos del servidor de comando y control (ver endpoint “DELETE /sessions/{session\_id}” del apéndice B). Abajo de esta se encuentra un listado de ejecuciones adjudicadas recuperadas gracias al endpoint “GET /executions” (ver Apéndice B); para cada una de ellas se destina otra tarjeta que alberga dos pestañas, una con información básica de la ejecución que incluye un botón para eliminarla (gracias al endpoint DELETE /executions/{execution\_id}, ver Apéndice B) y otra con sus respectivos reportes. Esta última reutiliza la componente que ya fue explicada en la pantalla anterior. Finalmente, en la parte superior del subpanel se encuentra una flecha que regresa a la aplicación a la vista de búsqueda. La vista descrita ha sido plasmada en la Figura 16.

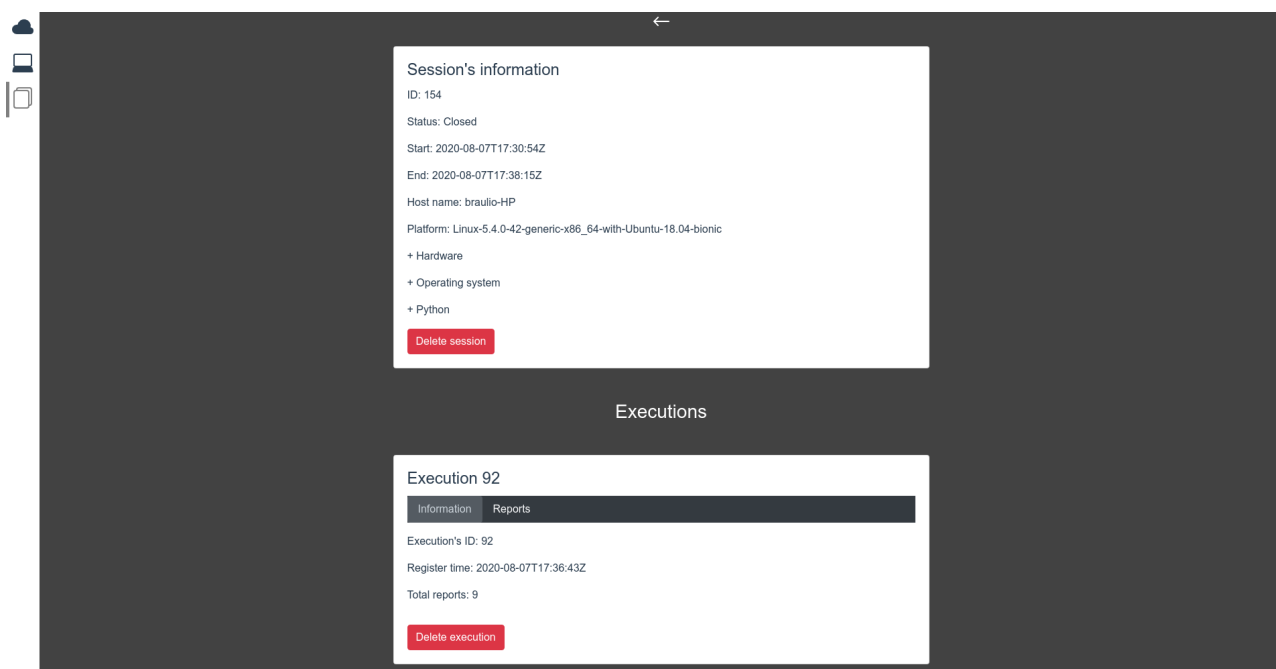


Figura 16: Subpanel de consulta de sesión

## 5.9 Selección e implementación de pruebas

Se escogieron ciertas técnicas del compendio elaborado (ver Apéndice A) para ser traducidas en tests del verificador. La intención de la elección fue conformar un grupo lo suficientemente variado tanto a nivel técnico, para demostrar el alcance posible de las pruebas que pueden ser implementadas en Secchiware, como respecto a las clasificaciones expuestas en el apartado 2.4, con el fin de poder aplicar posteriormente las pruebas en entornos diferenciados y analizar los resultados obtenidos en cada uno de ellos. Los números correspondientes a las técnicas seleccionadas son los siguientes: 10, 12, 16, 21, 25, 27, 29, 32, 38, 39, 43, 44, 46, 50, 55, 65, 70, 79, 81, 85, 92, 98, 100, 101, 105, 107, 116, 118, 121, 122, 123, 127 y 129.

Resulta llamativa la forma en la que se programaron algunos tests. Para Android se desarrollaron ciertas pruebas pensadas para ser ejecutadas con QPython aprovechando SL4A. Gracias a ello se pudieron elaborar fácilmente pruebas con Python que aprovechan funcionalidades particulares del sistema operativo orientado a móviles. Un ejemplo pintoresco es la detección de llamadas entrantes (técnica 29, ver Apéndice A), el cual se ha incluido en la Figura 17.

```
34     @TestSet.test(  
35         name="Is the phone participating in calls?",  
36         description="Waits for a phone call during 10 seconds.")  
37     def detect_call(self) -> TestResult:  
38         self.droid.startTrackingPhoneState()  
39         i = 0  
40         while i <= 9 and self.droid.readPhoneState().result['state'] == "idle":  
41             sleep(1)  
42             i += 1  
43         if i <= 9 or self.droid.readPhoneState().result['state'] != "idle":  
44             result = TestSet.TEST_PASSED  
45         else:  
46             result = TestSet.TEST_FAILED  
47         self.droid.stopTrackingPhoneState()  
48         return result
```

Figura 17: Ejemplo de prueba que utiliza SL4A

Para confeccionar pruebas que requerían trabajar con el lenguaje ensamblador de la arquitectura x86, se programaron las mismas como ensamblador *in-line* del lenguaje C. Según el caso, el código generado se compiló como un binario independiente cuya ejecución es iniciada desde un nodo escrito en Python, el cual captura el retorno del mismo, o como librería compartida que es importada desde el lenguaje interpretado y trabajada como si de un módulo propio se tratase gracias a la librería *ctypes*. En la Figura 18 se ha plasmado el programa en C correspondiente a la técnica 79 (ver Apéndice A) y en la Figura 19 como este es utilizado desde Python.

Para finalizar, también cabe mencionar que se usó esta misma metodología de programar en C una librería a ser cargada dinámicamente e invocada desde Python para desarrollar algunos tests particulares de Windows que requerían interactuar con la API del sistema. Para ello se aprovechó la librería *windows* [86] disponible para el lenguaje compilado.

```
7     int CPUID_HypervisorBitTest()  
8     {  
9         uint32_t ecx;  
10        asm volatile (  
11            "cpuid" \  
12            : "=c" (ecx) \  
13            : "a" (0x01));  
14        return (ecx >> 31) & 0x01 == 1 ? TEST_FAILED : TEST_PASSED;  
15    }
```

Figura 18: Ejemplo de programación de una prueba con lenguaje ensamblador

```
7 class X86VirtualizationSet(TestSet):
8
9     def __init__(self):
10         package_dir = os.path.dirname(os.path.abspath(__file__))
11         shared_bin_dir = os.path.join(package_dir, "shared", "bin")
12         self.dll = ctypes.CDLL(os.path.join(shared_bin_dir, "x86_shared.so"))
13
14     @TestSet.test(
15         name="Is CPUID hypervisor bit on?",
16         description=
17             "Calls the x86 instruction CPUID and checks the value of the ECX "
18             "register's 31th bit.")
19     def cpuid_hypervisor_bit_on(self) -> TestResult:
20         return self.dll.CPUID_HypervisorBitTest()
```

Figura 19: Ejemplo de aplicación de ctypes en la implementación de pruebas

## 5.10 Distribución

Secchiware ha sido puesto a disposición del público a través de dos repositorios en GitHub:

- <https://github.com/Bravlin/Secchiware>
- <https://github.com/Bravlin/secchiware-desktop-client>

El primero de ellos contiene el código particular de las componentes de nodo, servidor de comando y control y cliente por línea de comandos, como también de las librerías comunes creadas específicamente para este proyecto; a su vez, resguarda las pruebas implementadas y la documentación en formato OpenAPI de los endpoints diseñados. Se han documentado los módulos, clases y funciones desarrollados usando las cadenas de documentación de Python con el ya mencionado fin de facilitar su comprensión y aprovechamiento en caso de que un interesado desee expandir el producto o realizar una implementación derivada propia.

En el archivo “README” asociado ha quedado explicado el propósito y composición del sistema, como también la forma de instalar cada una de sus partes. A su vez, se han elaborado ficheros con las dependencias externas de cada componente y sus versiones para agilizar así su instalación mediante el gestor de paquetes *pip* y asegurar el correcto funcionamiento del producto.

Por otra parte, dada la extensión del cliente gráfico elaborado este ha quedado alojado de forma independiente en el segundo repositorio. El fichero “README” de aquel contiene la explicación de como instalar sus dependencias, iniciarlo en modo de prueba o generar un build.

Debido a que Secchiware Desktop Client ha sido programado utilizando únicamente tecnologías front-end, se puede compilar el proyecto en formato web para ser fácilmente abierto en un navegador o, incluso, ser proporcionado por un servidor. Esto ha sido aprovechado para alojar gratuitamente un build en GitHub Pages, al cual se puede

acceder a través del enlace siguiente: <https://bravlin.github.io/secchware-desktop-client/>. De esta manera, el cliente puede ser probado en línea, aunque esta versión particular presenta la limitación de no poder comunicarse con servidores C&C que operen con HTTP plano debido a la política de los navegadores web actuales de no permitir accesos asíncronos a servidores HTTP si el que aloja la página originalmente peticionada trabaja con la versión segura del protocolo. Algunos navegadores permiten configurar este comportamiento y así solventar este problema. De todas formas, el programa puede compilarse como una aplicación Electron para su uso independiente, tal como fue planificado originalmente.

Ambos repositorios han sido publicados bajo la licencia GPLv3, con la cual se busca asegurar que cualquier producto derivado mantenga el carácter libre de Secchware.

## Capítulo 6: Verificación de transparencia de entornos selectos

Tras el desarrollo de Secchiware, resultó interesante comprobar la transparencia de distintos ambientes conformados por distintos sistemas operativos, mecanismos de virtualización e instrumentos de análisis ahora que se contaba con una herramienta adecuada para ello. La intención de esto no fue solamente verificar el producto en sí, sino también, afirmar los conceptos que fueron explicados en el Capítulo 2.

En el Apéndice D se informan las características de los siete ambientes preparados para esta evaluación y los resultados obtenidos de las pruebas efectivamente aplicadas en cada uno de ellos. Algunos comentarios y conclusiones se pueden desprender a partir de los resultados obtenidos.

Quizás lo más llamativo han sido los valores devueltos por el test correspondiente a la técnica 118 (ver apéndice A). Su implementación fue llevada a cabo tomando cien veces el tiempo de ejecución de un grupo de instrucciones para realizar el cálculo estadístico del coeficiente de variación de la muestra obtenida (calculada como la división entre la dispersión estándar y la media del conjunto de datos). Se tomó el criterio típico para determinar la similitud de los valores: un coeficiente menor a 0.8 implica que la muestra es homogénea; en caso contrario se asume heterogénea.

La aplicación de este análisis estadístico partió de la premisa de que en un entorno real el conjunto de datos obtenido debería ser homogéneo, mientras que en uno virtualizado sucedería lo contrario. Sin embargo, su ejecución repetida dio resultados lo suficientemente dispares en todos los ambientes como para desestimar su validez. Se requiere profundizar en las posibles causas de este fenómeno, pero una suposición inicial es que este test se ve excesivamente influenciado por las decisiones del planificador del sistema operativo al momento de su concreción. Sea cual sea la razón, se puede determinar que el estudio de los tiempos de ejecución de la forma en que fue implementada no es un correcto indicador de transparencia.

Por otro lado, algunas pruebas específicamente pensadas para sistemas operativos de escritorio o servidores basados en Linux pudieron ser iniciadas en Android debido a que también este sistema se sustenta del susodicho núcleo. Desafortunadamente, no todas pudieron ser ejecutadas; se sospecha que esto puede deberse a ciertas leves modificaciones realizadas sobre el kernel, ya sea en su funcionalidad o en sus restricciones de seguridad. Resulta destacable también el caso de la prueba 118 (consulta de la cantidad de núcleos del procesador usando el comando *nproc*), que pudo ser ejecutada en un emulador pero no en un dispositivo real.

En lo que respecta a la técnica 29, la cual involucra identificar llamadas, se aprovechó que el emulador de Android permite generarlas. A propósito entonces se evitó recibir



llamadas en el smartphone real y sí se hizo en el simulado. Con esto se intentó mostrar que las pruebas basadas en la detección del uso del sistema por parte de un humano escapan a la virtualización, reafirmando así que la transparencia es un concepto más complejo que solo este aspecto. Los tests para Windows número 92 (análisis de la velocidad de movimiento del cursor) y 129 (detección de reemplazo de la ventana enfocada) se han condicionado con un propósito semejante: el primero de ellos ha sido exitoso en el ambiente virtualizado y ha fracasado en el real, mientras que lo contrario ha sucedido con el segundo.

La prueba referida a la comprobación del contenido del portapapeles (número 98 del compendio del Apéndice A), ha mostrado un resultado extraño, ya que en los dos entornos bajo Android se le proporcionó contenido, pero solo fue reconocido correctamente en el virtualizado. La razón de esta anomalía también ha quedado pendiente de ser investigada.

Los resultados obtenidos a partir de los ambientes sobre Linux han sido casi en su totalidad los esperados. En particular se destacan los valores arrojados por el entorno E5, un contenedor de Docker, el cual se ha mostrado transparente frente a las pruebas de reconocimiento de virtualización de hardware mientras que ha fallado en aquellas que específicamente apuntan a contenedores.

El principal imprevisto ha sido que el entorno E3, el cual no fue simulado, falló en el test 101, el cual busca si en el sistema existen direcciones IP registradas típicamente asociadas a Docker. Este ambiente ha hecho de anfitrión de otros, incluido el basado en contenedores. Por como trabaja Docker, se conforma una red en el anfitrión a la cual todo contenedor pertenece por defecto y por ende direcciones IP acordadas se le son otorgadas. Debido a que el anfitrión termina operando como puerta de enlace para los contenedores, este también adquiere una dirección IP correspondiente a esa red, razón por la cual el entorno E3 se mostró opaco ante la prueba. Esto implica que la técnica aplicada con el fin de identificar un ambiente sobre Docker mediante el reconocimiento de direcciones de red presenta falsos positivos si se aplica en un anfitrión.

Finalmente, la implementación de la técnica 105, que usa un comando particular de Windows para consultar la temperatura de la CPU, fue desarrollado esperando que este devolviera un error en un ambiente virtualizado. En cambio, el mismo ni siquiera estaba presente en el entorno E7, correspondiente a un equipo simulado con VirtualBox. Se desconoce si esta situación ocurre en todo sistema virtualizado con Windows 10 o solo se da en las imágenes provistas por Microsoft; de resultar ser el primer caso, el test podría modificarse para asumir que la ausencia del comando implica virtualización.

Durante la ejecución de las pruebas se identificaron errores de programación que fueron solucionados, con la consecuente repetición de aquellas. Esto proporciona confianza en los resultados obtenidos, por lo cual se puede determinar que las anomalías que fueron explicadas se deben a potenciales características de los entornos preparados que no fueron consideradas o a falencias detrás de los conceptos o implementaciones de

los tests, que los hace intrínsecamente imprecisos. Es por ello que se considera exitosa la validación del marco teórico de este proyecto, como a su vez se determina que Secchiware cumple adecuadamente con el propósito por el cual fue creado.

## Capítulo 7: Gestión del proyecto

En esta sección se abordan cuestiones acerca de como se encaró la planificación del proyecto y se realiza una inspección de su ejecución.

### 7.1 Proceso de desarrollo

El proceso de elaboración seleccionado para Secchiware ha seguido una estrategia iterativa e incremental. Cockburn [14] define al desarrollo iterativo como “una estrategia de planificación de retrabajo en la cual el tiempo es dejado de lado en pos de revisar y mejorar partes del sistema”; por otra parte, también describe al desarrollo incremental como “una estrategia de especificación de etapas y de planificación en la cual varias partes del sistema son desarrolladas en diferentes momentos o a ritmos distintos y que son integradas a medida que son completadas”.

La metodología incremental se caracteriza por la creación de versiones preliminares del producto deseado, llamadas incrementos, que incorporan progresivamente mejoras y nueva funcionalidad. Al menos aplicada en su estado puro, necesita que los requerimientos que definen la generalidad del producto puedan ser definidos claramente desde el comienzo del proyecto y que se mantengan estables durante el transcurso del mismo para poder establecer adecuadamente los distintos incrementos. El propósito de esta segmentación es la de poder encarar anticipadamente los aspectos funcionales y estructurales de mayor riesgo primero. De esta forma se identifican limitaciones de diseño, tecnológicas y de implementación lo más rápido posible, lo cual busca evitar que los errores de diseño se propaguen a etapas avanzadas del proyecto, en las cuales el costo de corrección es mucho mayor.

En el caso particular de este trabajo, dado que los requisitos no provinieron de un cliente particular, sino de un mercado general cuyas necesidades no presentan evoluciones en el corto plazo, la condición previamente mencionada se cumplió, lo que hizo adecuado la aplicación de un esquema incremental. Una vez finalizada la investigación sobre el dominio del problema al que este proyecto responde (ver Capítulo 2) y tras haber dejado en claro los requerimientos del mercado y los objetivos particulares del producto (ver Capítulo 3), se definió la arquitectura general del sistema y las tecnologías principales que se aplicarían en el mismo. Dado que el éxito del proyecto dependía principalmente de estos conceptos, el primer y más voluminoso incremento consistió en la implementación básica de la funcionalidad fundamental del nodo, servidor de comando y control y ambos clientes. Así fue que se logró gestionar la fuente de mayor riesgo para el producto mediante la validación temprana de su arquitectura.

El segundo factor más crítico del proyecto se originó en la inexperiencia del autor en una gran cantidad de las tecnologías aplicadas. Esto convirtió al desarrollo de Secchiware en un proceso de aprendizaje para él, por lo cual haber asumido que las características

del sistema podrían llegar a ser implementadas en primera instancia correctamente y con la debida calidad hubiera sido un gravísimo error. De haber incurrido en él, la magnitud del retrabajo necesario para adecuar el producto hubiera sido extremadamente difícil de gestionar; incluso podría haber necesitado comenzar de nuevo el desarrollo. Por ende, se partió asumiendo que los refinamientos iban a ser inescapables. He aquí la fortaleza de aplicar una estrategia iterativa en conjunto a una incremental: al aplicar revisiones en cada incremento, se acota el alcance del retrabajo potencial y se garantiza la calidad del sistema elaborado hasta ese momento antes de avanzar a la siguiente fase.

Se presenta a continuación el listado de incrementos finalmente ejecutados en su orden cronológico:

- I. Diseño e implementación de la arquitectura general del sistema. Se subdividió en las siguientes iteraciones:
  - 1) Iteración inicial. Los aspectos abarcados fueron:
    - Carga dinámica de módulos de test.
    - Comunicaciones generales entre un nodo en un servidor de comando control.
    - Inspección de paquetes de pruebas y el enunciado de su composición.
    - Transmisión de archivos comprimidos que contienen nuevas pruebas.
    - Diseño e implementación del pequeño framework para desarrollo de pruebas.
    - Implementación básica tanto del cliente CLI como del gráfico.
  - 2) Revisión. Algunos de los aspectos involucrados fueron:
    - Pulido general del sistema implementado hasta el momento.
    - Ejecución selectiva de pruebas.
    - Gestión de conexiones al finalizar un nodo o un servidor de comando y control.
    - Especificación y documentación inicial de las APIs de los nodos y del servidor de comando y control e implementación de códigos de respuesta HTTP.
- II. Análisis, diseño e implementación de autenticación de mensajes HTTP.
- III. Elaboración del compendio de técnicas de reconocimiento de entornos y su selección e implementación en forma de pruebas de transparencia.
- IV. Diseño e implementación de la base de datos y elaboración de endpoints para búsqueda y consulta de datos históricos.

V. Pulido general. Algunos de los aspectos involucrados fueron:

- Documentación del código.
- Refactorizaciones en el módulo “test\_utils”.
- Configuración de NAT habilitada para nodos dentro de un entorno virtualizado que requieren comunicar la dirección IP y puerto desde los cuales escucha desde su anfitrión.

VI. Continuación del desarrollo del cliente gráfico. Se subdividió en las siguientes iteraciones:

- 1) Iteración inicial en la que se puso a tono al cliente con los avances realizados en el resto del sistema desde el incremento inicial (exceptuando la búsqueda y consulta de reportes y ejecuciones).
- 2) Revisión y pulido general del cliente.

VII. Análisis, diseño e incorporación del repositorio en memoria. Se subdividió en las siguientes iteraciones:

- 1) Implementación del caché en memoria del servidor de comando y control.
- 2) Análisis de condiciones de carrera y aplicación de bloqueos.

VIII. Pulido general. Algunos de los aspectos involucrados fueron:

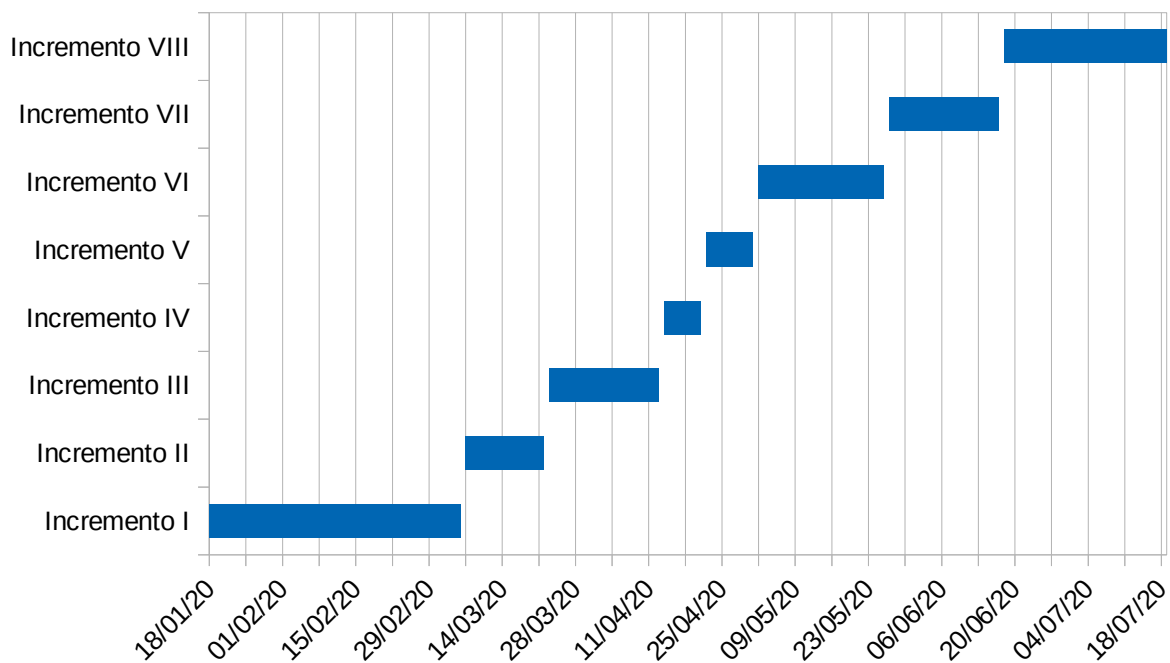
- Ampliación de los endpoints de búsqueda de sesiones y ejecuciones.
- Mayor granularidad en la selección de tests a ejecutar.
- Reemplazo de estructuras de datos usados en la carga de pruebas para evitar repetir la ejecución de una misma prueba.
- Incorporación de búsqueda y visualización de sesiones al cliente gráfico y su reorganización general.
- Reorganización del código del servidor de comando y control.
- Documentación remanente.

Los incrementos V y VIII se incorporaron como revisiones generales del desarrollo, ubicadas aproximadamente a la mitad del mismo y al final respectivamente. Mientras que el quinto tuvo la intención de poder aplicar la experiencia adquirida hasta el momento para aplicar refinamientos antes de proseguir con el proyecto, el octavo se planteó para terminar de otorgarle la calidad requerida para su publicación, lo cual estuvo motivado por el objetivo de causar una primera impresión agradable entre la comunidad. A su vez, se incluyó la tarea de documentar el código del producto, debido a la estabilidad manifestada por el sistema en cuanto a las interfaces de la funcionalidad hasta ese momento ya implementada se refiere.

En la Tabla 5 se ha indicado la distribución de los incrementos a lo largo del tiempo, información que también se ha presentado en forma gráfica en la Figura 20.

*Tabla 5: Inicio y finalización de cada incremento*

<b>Incremento</b>	<b>Fecha de inicio</b>	<b>Fecha de fin</b>
Incremento I	18/01/2020	06/03/2020
Incremento II	07/03/2020	22/03/2020
Incremento III	23/03/2020	13/04/2020
Incremento IV	14/04/2020	21/04/2020
Incremento V	22/04/2020	01/05/2020
Incremento VI	02/05/2020	26/05/2020
Incremento VII	27/05/2020	17/06/2020
Incremento VIII	18/06/2020	19/07/2020



*Figura 20: Evolución temporal de los incrementos*

## 7.2 Comparativa entre la planificación original y la ejecución concretada del proyecto

En el apartado E.1 del Apéndice E se ha incluido el diagrama de Gantt del cronograma originalmente planeado para el proyecto, mientras que en la Tabla 6 se presenta parte de la información allí contenida para simplificar su consulta en este capítulo. El contenido equivalente para la evolución temporal real del trabajo han sido presentados en el apartado E.2 del Apéndice E y en la Tabla 7.

*Tabla 6: Tareas y su duración en horas originalmente planificadas*

Nº	Tarea	Cantidad de horas
1	Investigación de técnicas para detectar entornos de análisis	50
2	Definición del marco teórico	40
3	Definición de requerimientos funcionales y no funcionales	30
4	Diseño de la arquitectura general del sistema	40
5	Definición del esquemas de transmisión de resultados	5
6	Definición del reporte de resultados	5
7	Especificación de la integración de nuevos módulos	10
8	Estudio y selección de tecnologías	15
9	Desarrollo del verificador de transparencia	100
10	Selección de vulnerabilidades representativas	13
11	Implementación modular de tests acordes	14
12	Preparación de entornos de prueba representativos	16
13	Verificación funcional del producto	27
14	Documentación del desarrollo del producto	60
15	Redacción de conclusiones	10
16	Revisión del informe terminado	20
<b>Total horas</b>		<b>455</b>

Tabla 7: Tareas y su duración en horas realmente ejecutadas

Nº	Tarea	Cantidad de horas
1	Investigación de técnicas para detectar entornos de análisis	50
2	Definición del marco teórico	50
3	Definición de requerimientos funcionales y no funcionales	10
4	Diseño de la arquitectura general del sistema	60
5	Especificación de la integración de nuevos módulos	10
6	Estudio y selección de tecnologías	17
7	Definición de APIs REST	15
8	Desarrollo del verificador de transparencia	200
9	Selección de vulnerabilidades representativas	14
10	Implementación modular de tests acordes	20
11	Preparación y prueba de entornos representativos	7
12	Documentación del desarrollo del producto	60
13	Redacción de conclusiones	5
14	Revisión del informe terminado	20
<b>Total horas</b>		<b>538</b>

Como puede observarse, el tiempo total originalmente planteado para el proyecto iba a ser de 455 horas, mientras que realmente fueron destinadas 538, lo que supone un incremento del 18,24%. Por otro lado, la finalización del trabajo se había estimado para fines de abril del año 2020, pero la realidad es que este evento se produjo a fines de agosto, lo que implica un retraso de tres meses.

Ambas desviaciones tienen orígenes bien identificados. La primera de ellas, la diferencia en horas trabajadas, proviene principalmente de haber subestimado la complejidad del producto. Particularmente fue el requerimiento de la gestión de múltiples entornos que irrumpió con la simpleza originalmente esperada en el verificador de transparencia, lo cual tuvo un efecto profundo en su arquitectura y consecuentemente en el resto de su construcción.

Otra razón derivada de la anterior fue la abrumadora cantidad de tecnologías ajenas para el autor que fueron aplicadas en el proyecto. Antes de elaborar el producto él jamás había realizado un proyecto en Python y en consecuencia nunca había trabajado con su librería estándar ni con los módulos de terceros que fueron mencionados en el Capítulo 5. A su vez, tampoco había codificado en el lenguaje ensamblador de la arquitectura x86 (o x86\_64), usado la librería de Windows para C o empleado S4LA para operar con la API de Android (o trabajado con ella de forma alguna), cuestiones que fueron requeridas para el desarrollo de algunas pruebas de transparencia. Por otro lado, el autor no había utilizado



antes ni SQLite ni Redis, aunque la primera herramienta no supuso mayores complicaciones al tratarse de un gestor de bases de datos relacionales estándar, algo para lo cual ya contaba con experiencia. Fue necesario también que aprendiera acerca de la especificación OpenAPI para documentar las APIs REST diseñadas.

De entre todas las herramientas aplicadas, con la cual al autor más le costó familiarizarse fue Vue, lo cual contribuyó a que sea la componente del cliente gráfico la que más trabajo imprevisto tuvo. Otros motivos de este agravante fue la ya mencionada complejidad inesperada subyacente del sistema y la falta de destreza general del creador en lo que respecta a la elaboración de interfaces gráficas.

Todas las cuestiones mencionados hicieron que el tiempo de desarrollo para Secchiware pasara de 100 horas a 200, por lo cual esta actividad es la que sufrió la mayor desviación de todo el proyecto.

En cuanto a la demora de la finalización de aquel, su causa es más simple. En principio se había estimado una dedicación semanal de 20 horas, teniendo en consideración que la única actividad obligatoria que el autor iba tener era una ayudantía en una materia de la facultad. Sin embargo, a mediados de febrero comenzó una pasantía a medio tiempo que aún a fecha de hoy (agosto de 2020) sigue vigente y en marzo se incorporó como ayudante en una asignatura más. A su vez, la cuarentena impuesta también en marzo trajo consigo la virtualización de las clases, lo que implicó un proceso de aprendizaje y adaptación a la nueva modalidad que produjo cambios en la manera en la que se realizaban ciertas actividades e incluso algunas nuevas. Esta carga laboral imprevista terminó reduciendo la dedicación que se le pudo dar al trabajo a 10 horas semanales en promedio entre los meses de marzo y julio.

Por último, se pueden apreciar algunas diferencias en las actividades del cronograma. Las tareas de selección e implementación de tests de transparencia fueron adelantadas ya que durante la planificación de los incrementos se estableció que tenían un riesgo notable. La definición de esquemas de comunicación y del reporte de resultados se combinaron en la especificación de las APIs REST. Finalmente, la verificación del producto se integró al desarrollo siguiendo así los lineamientos de un desarrollo iterativo e incremental.

### **7.3 Lecciones aprendidas**

Quizás el error más grave cometido al planificar el proyecto fue no haber indagado preliminarmente en los requerimientos del sistema para comprender mejor las características que debía poseer y así estimar su tiempo de elaboración. En este caso particular, dado que los requisitos surgieron a raíz de las necesidades observadas en el mercado, haber profundizado más en el dominio del problema de fondo y su contexto habría revelado de forma temprana el requerimiento de la administración de múltiples entornos en simultáneo, el cual tuvo el mayor impacto en el planteo inicial.

Desde su propuesta ya se había previsto que el trabajo iba a tener una pesada carga de investigación, razón por la cual se asignó una gran cantidad de horas para determinar el marco teórico referido al reconocimiento de entornos de análisis y la identificación de técnicas con este propósito. Ambas actividades pudieron realizarse en paralelo, algo que no había sido previsto, y a su vez la investigación necesaria para ambas desveló los requerimientos del sistema sin esfuerzos adicionales, razón por la cual la estimación de esta última tarea se alejó de la realidad. Sin embargo, el tiempo total planificado para este conjunto de actividades que conforman la etapa destinada al análisis solo se diferencia en 10 horas con el real, por lo que se puede decir la estimación en general fue acertada.

Por otro lado, aunque se había contemplado el aprendizaje de nuevas tecnologías en la estimación del tiempo de desarrollo original, esta aproximación fue superada ampliamente. De todas formas, la aplicación de un proceso de desarrollo iterativo e incremental ayudó a mitigar en cierta medida esta situación. Gracias a ello se pudieron focalizar las refactorizaciones necesarias para elevar la calidad de Secchiware en momentos claves de la experiencia del creador que previnieron mayores esfuerzos futuros.

Esto implicó también que la actividad de desarrollo absorbiera a la de verificación del producto en el cronograma realmente ejecutado, lo cual llevó a que se realizaran comprobaciones al final de cada iteración más acotadas en su alcance, algo que a la larga terminó beneficiando al proceso. La elección de la metodología de desarrollo se considera un éxito entonces, ya que la misma ayudó a gestionar adecuadamente la complejidad requerida en el sistema y logró reducir en gran parte los riesgos que con ella aparecieron.

Un aspecto también destacable que se dio durante la construcción de Secchiware fue su documentación parcial en instancias claves. Más allá de que se realizó con el fin último de explicar el sistema a sus destinatarios, sirvió como un valioso insumo para administrar el proyecto en sí, ya que facilitó la consulta y visualización de los diseños y módulos confeccionados hasta el momento y del alcance del verificador en general. Su repercusión más notable fue quizás en la integración de nueva funcionalidad en incrementos subsecuentes.

Otro efecto positivo de haber realizado ese esfuerzo de documentación es que simplificó la redacción de este informe, dado que mucho contenido ya elaborado pudo ser adaptado y trasladado aquí rápidamente. Si se comparan la planificación y la ejecución final, puede apreciarse que a pesar de que la complejidad del producto fue superior a la esperada, la dedicación puesta sobre esta etapa del proyecto se mantuvo esencialmente igual a la estimada.

## Capítulo 8: Conclusiones

La problemática que presenta el malware que aplica técnicas de reconocimiento de entornos mediante las cuales consigue eludir ser estudiado supone un gran impedimento en la labor de investigadores en seguridad informática. A su vez, perjudica gravemente la utilidad de sandboxes usados con fines defensivos basados en la ejecución de análisis dinámico sobre programas sospechosos.

Existen un sin fin de formas aplicadas o potencialmente aplicables por malware para discriminar entornos y comportarse benignamente en caso de determinar que se encuentra en un contexto hostil. Los expertos en ciberseguridad y los atacantes se encuentran inmersos en un juego del gato y el ratón, en el cual los primeros se hallan permanentemente tratando de mitigar las técnicas utilizadas por los delincuentes, mientras que estos idean nuevas formas para que sus programas malintencionados esquiven los esfuerzos de inspección que se realicen sobre ellos.

Expertos como Bulazel et al. afirman que la mejor vía de acción para responder a este problema es la construcción de entornos de análisis transparentes [10]. Esto no es tarea fácil, debido a la gran cantidad de variables presentes en un sistema computacional. A su vez, la transparencia puede ser un detrimento en la eficacia del análisis que efectivamente se consiga realizar; o, como proponen Garfinkel et al., incluso puede ser inalcanzable en algunos aspectos [31]. Esto lleva a deducir que la transparencia absoluta, entendida como la imposibilidad total de discernir dos entornos, es una utopía.

En este trabajo se ha llevado a cabo una investigación cuyo objetivo fue la comprensión de la situación expuesta y de sus conceptos de trasfondo. En base a este estudio se ha presentado una definición de transparencia que pone el foco en las características que un proceso maligno particular espera encontrar. A su vez, se han confeccionado criterios de clasificación de técnicas de evaluación de transparencia que buscan brindar una perspectiva mejor delimitada que la que ha sido aportada en otros documentos académicos. De esta forma, se han presentado los tópicos pertinentes a la evasión de análisis dinámico bajo un enfoque práctico, dando así un importante primer paso, al menos en el contexto de este proyecto, para la elaboración de una solución.

En base a esto se ha desarrollado Secchiware, un verificador de transparencia de entornos pensados para análisis dinámico de malware, el cual busca brindar una herramienta que permita a la comunidad de la seguridad informática preparar sandboxes que sean capaces de estudiar muestras efectivamente. Su funcionamiento se fundamenta en la ejecución de las mismas técnicas aplicadas por procesos no confiables para determinar de antemano las debilidades y fortalezas de un ambiente.

La investigación ya mencionada derivó en el entendimiento del segmento de mercado que necesita de un producto así, como a su vez las características esperadas del mismo. Quizás el requerimiento que más ha influenciado en la elaboración de Secchiware ha sido

el de poder gestionar varios entornos simultáneamente, aspecto que fue determinante al momento de diseñar la arquitectura del sistema. Aquel surgió al identificar que varios investigadores han realizado evaluaciones sobre entornos diferentes y que han propuesto el uso simultáneo de estos a la hora de realizar análisis dinámico con el fin de aumentar su probabilidad de éxito [50].

Un aspecto importante implementado en el sistema es la posibilidad de incorporar fácilmente nuevas pruebas definidas por sus usuarios, cuyo fin es el de permitirles actualizar el verificador en base a las novedades en el mundo del malware y a sus propias necesidades. Por otro lado, se han incorporado varias características ausentes en soluciones semejantes que favorecen el uso realmente profesional de Secchiware. Otras cuestiones que influían negativamente en la seguridad, rendimiento y correctitud funcional del producto fueron detectadas y atendidas debidamente.

El carácter modular, libre y gratuito del producto favorecen su ampliación y modificación por parte de cualquier interesado. Con la intención de fomentar esto es que también se ha documentado en detalle el código y las interfaces de las componentes que conforman al producto.

Más de cien técnicas de anti-análisis basadas en el reconocimiento de entornos pudieron ser recopiladas y categorizadas a partir de la bibliografía consultada, de entre las cuales se seleccionaron e implementaron alrededor de treinta para demostrar el potencial del verificador. Se ha podido usar Secchiware para ejecutar estas pruebas en ambientes diferenciados con éxito. Esto, por un lado, ha permitido demostrar la efectividad del verificador, cuyo estado al momento de la redacción de este trabajo ha cumplido con los requerimientos deducidos del mercado, dentro de los límites particulares que se han establecido para este contexto. A su vez, se pudieron confirmar algunos conceptos teóricos encontrados durante la revisión bibliográfica.

Para finalizar, este proyecto ha sido una gran oportunidad para su creador para aplicar los conocimientos y habilidades adquiridos durante su transcurso por la carrera. Además de eso, fue una voluminosa fuente de aprendizaje, tanto por lo novedoso que le resultó el dominio del problema, por todas las tecnologías que fueron utilizadas durante el desarrollo del sistema y como también por los errores y aciertos en la gestión de su puesta en marcha.

## **8.1 Trabajos futuros**

En este apartado se exploran algunas de las propuestas mediante las cuales Secchiware podría seguir puliéndose y se mencionan ideas para trasladar lo aprendido en este trabajo a proyectos con fines comerciales.

### **8.1.1 Mejoras a incorporar**

Aunque el estado de Secchiware al momento de la redacción de este trabajo cumple con los requerimientos deducidos a partir del mercado y por ende está preparado para ser

utilizado por el público, como todo sistema presenta aspectos que pueden ser pulidos. Entre ellos se encuentran las componentes cliente: una implementación del cliente por línea de comandos alternativa que trabaje con una sesión y no con operaciones aisladas resultaría mucho más fácil de utilizar que la que se ha propuesto para este proyecto; a su vez, un equipo más experimentado en interfaces gráficas que el autor podría desarrollar un cliente visual más atractivo y cómodo. La naturaleza modular del verificador de transparencia y su documentación dejan abierta la puerta a estas posibilidades.

Otro tópico a tratar es la existencia de herramientas comerciales cerradas para análisis dinámico basadas en sandboxes que no permiten alterar, al menos no fácilmente, su composición. Estos instrumentos típicamente habilitan al usuario a subir un archivo binario que es ejecutado y estudiado bajo confinamiento y devuelven los resultados obtenidos tras este proceso.

Debido a que los nodos de Secchiware están escritos en Python y por ende dependen de un intérprete de este lenguaje para funcionar, los mismos no pueden ser iniciados con las herramientas previamente mencionadas. Dos vías de acción tentativas son planteadas a continuación para solventar esta situación:

- La primera de ellas sería implementar una versión de la componente de nodo en un lenguaje compilado, potencialmente C. Esto implica tener que buscar librerías equivalentes a las de Python que fueron utilizadas al desarrollar al nodo, como así volver a programar toda la funcionalidad desarrollada exclusivamente para este proyecto. Además, requeriría implementar pruebas en el lenguaje compilado directamente, lo que como consecuencia obligaría a cambiar profundamente el diseño del repositorio de tests. Este esfuerzo se vería mayoritariamente recompensado si se altera Secchiware para que sea agnóstico del lenguaje, lo cual le brindaría de un mayor potencial; el uso de HTTP como protocolo de comunicación y JSON como formato de mensajes entre subsistemas ya conforman cimientos sólidos para esta posibilidad.
- La segunda alternativa es la de explorar maneras de empaquetar un intérprete de Python y módulos escritos en este lenguaje en un binario ejecutable independiente. Este canal supone un esfuerzo muchísimo menor, aunque presenta como riesgo las limitaciones inherentes que puedan presentar las soluciones de terceros disponibles para este fin. De no existir un medio lo suficientemente bueno, se podría desarrollar un empaquetador propio, lo que elevaría significativamente la complejidad de esta vía.

Por otro lado, si el verificador llegara a alcanzar una cierta popularidad y de haber partes interesadas en financiarlo sería un importante añadido un almacén de paquetes en línea para que los desarrolladores de pruebas puedan compartirlas con la comunidad de forma sencilla. Esto también traería algunos cambios importantes en el esquema de los repositorios; uno de ellos sería pasar de priorizar a los archivos físicos presentes en el

sistema como elemento informativo de los tests alojados a trabajar con metadatos asociados a un paquete de pruebas como elementos descriptivos del mismo.

Otra característica mejorable tras la propuesta anterior es la manera en que son inspeccionados los paquetes del repositorio de pruebas. Actualmente ese proceso se lleva a cabo importando directamente sus contenidos, lo que acarrea un problema de seguridad debido a que cualquier código que se encuentre en el ámbito más externo de un módulo de Python es ejecutado durante este proceso. Ya que Secchiware está pensado para ser usado dentro de una red local bien controlada esto no supone un gran riesgo dado que los tests con los que se nutren a los repositorios son desarrollados por la misma gente que trabaja con el sistema. A lo sumo, podrían llegar a ser descargados desde fuentes externas, debidamente revisados por los usuarios y recién ahí añadidos a un servidor de comando y control. La situación tomaría más relevancia si en el futuro se implementase el susodicho almacén en línea; la manera de asegurar el sistema dependería en gran medida de como este sea finalmente aplicado.

### **8.1.2 Expansión comercial**

Secchiware ha quedado a disposición de cualquier interesado como una herramienta de software libre y gratuita. A pesar de ello, la experiencia adquirida en este rubro puede ser puesta en juego para proyectos derivados que sigan modelos de negocio típicamente aplicados a la computación en la nube. A continuación se describen brevemente dos posibles:

- Un Software como Servicio que gestionaría un conjunto de entornos de distinta naturaleza para poder ejecutar en ellos pruebas subidas por un usuario. El sistema sería de utilidad para investigadores en ciberseguridad y administradores de sistemas que necesiten conocer la transparencia de varias alternativas frente a un conjunto conocido de técnicas a las cuales deben hacer frente. De esta manera, podrían determinar rápidamente qué características deberían presentar los ambientes que mejor responden a sus casos de uso particulares.
- Una Infraestructura como Servicio mediante la cual ofrecer sandboxes implementados con distintas tecnologías y configurables con el fin de realizar análisis dinámico de malware en ellos. Su principal fortaleza sería el agrupamiento de estos entornos con el propósito de realizar el estudio de una misma muestra mediante computación en malla. La evaluación de transparencia se podría ofrecer para asistir al cliente en la selección de tecnologías y en la preparación de entornos. Aunque este proyecto parece similar a otros ya provistos por grandes proveedores de infraestructura, en los cuales se ofrecen mecanismos de virtualización con fines genéricos, el problema que intenta resolver es lo suficientemente particular para poder realizar una serie de optimizaciones en su aplicación. A su vez, por la naturaleza de los objetos bajo estudio sería imprescindible realizar una inversión mayor en seguridad.

## Bibliografía

1. Adrian L. (2019). AMD-V: La tecnología de virtualización del equipo rojo. Recuperado el 2 de agosto de 2020 de <https://www.profesionalreview.com/2019/11/01/amd-v/>
2. Adrian L. (2019). Intel VT: ¿Qué es y para que sirve? | Virtualización ágil y eficiente. Recuperado el 2 de agosto de 2020 de <https://www.profesionalreview.com/2019/10/19/intel-vt/>
3. Afianian, A., Niksefat, S., Sadeghiyan, B. y Baptiste, D. (2019). Malware Dynamic Analysis Evasion Techniques: A Survey. *ACM Computing Surveys*, 52(6), Artículo 126, 1-28. doi: 10.1145/3365001
4. Angular. (s.f.). Recuperado el 18 de julio de 2020 de <https://angular.io/>
5. Besler, F., Willems, C. y Hund, R. (2018). Countering Innovative Sandbox Evasion Techniques Used By Malware. Recuperado el 10 de agosto de 2020 de <https://www.first.org/resources/papers/conf2017/Countering-Innovative-Sandbox-Evasion-Techniques-Used-by-Malware.pdf>
6. Binariks. (2019). What to choose in 2020: Angular vs React vs Vue.JS Comparison? Recuperado el 2 de agosto de 2020 de <https://binariks.com/blog/what-to-choose-in-2020-angular-vs-react-vs-vuejs-comparison/>
7. Boelen, M. (2019). The 101 of ELF files on Linux: Understanding and Analysis. Recuperado el 10 de agosto de 2020 de <https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/>
8. Bootstrap. (s.f.). Recuperado el 17 de julio de 2020 de <https://getbootstrap.com/>
9. BootstrapVue. (s.f.). Recuperado el día 7 de agosto de 2020 de <https://bootstrap-vue.org/>
10. Bulazel, A. y Yener, B. (2017). A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion. *ROOTS: Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*, Artículo 2, 1-21. doi:10.1145/3150376.3150378
11. Bursztein, E., Malyshev, A., Pietraszek, T. y Thomas, K. (2016). Picasso: Lightweight Device Class Fingerprinting for Web Clients. *SPSM '16: Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, 93-102. doi: 10.1145/2994459.2994467

12. Cavage, M. y Sporny, M. (2019). Signing HTTP Messages. Recuperado el 15 de marzo de <https://tools.ietf.org/html/draft-cavage-http-signatures-12>
13. click. (s.f.). Recuperado el 31 de julio de 2020 de <https://click.palletsprojects.com/en/7.x/>
14. Cockburn, A. (2008). Using Both Incremental and Iterative Development. Recuperado el 19 de agosto de 2020 de <http://www.se.rit.edu/~swen-256/resources/UsingBothIncrementalandIterativeDevelopment-AlistairCockburn.pdf>
15. Command & control (C&C). (s.f.). Recuperado el 11 de julio de 2020 de <https://blog.malwarebytes.com/glossary/cc/>
16. Control de acceso HTTP (CORS). (2019). Recuperado el 10 de julio de 2020 de [https://developer.mozilla.org/es/docs/Web/HTTP/Access\\_control\\_CORS](https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS)
17. Cozzi, E., Graziano, M., Fratantonio, Y. y Balzarotti, D. (2018). Understanding Linux Malware. *S&P: Proceedings of the IEEE Symposium on Security and Privacy*. doi: 10.1109/SP.2018.00054
18. ctypes - A foreign function library for Python. (s.f.). Recuperado el 10 de julio de 2020 de <https://docs.python.org/3/library/ctypes.html#module-ctypes>
19. Cuckoo Sandbox - Automated Malware Analysis. (s.f.). Recuperado el 12 de agosto de 2020 de <https://cuckoosandbox.org/>
20. Debian - El sistema operativo universal. (s.f.). Recuperado el 20 de agosto de 2020 de <https://www.debian.org/index.es.html>
21. Disco Secchi. (s.f.). Wikipedia. Recuperado el 10 de agosto de 2020 de [https://es.wikipedia.org/wiki/Disco\\_Secchi](https://es.wikipedia.org/wiki/Disco_Secchi)
22. Django. (s.f.). Recuperado el 10 de julio de 2020 de <https://www.djangoproject.com/>
23. Docker. (s.f.). Recuperado el 15 de agosto de 2020 de <https://www.docker.com/>
24. Eby, P. J. (2010). PEP 3333 -- Python Web Server Gateway Interface v1.0.1. Recuperado el 12 de julio de 2020 de <https://www.python.org/dev/peps/pep-3333/>
25. Electron. (s.f.). Recuperado el 17 de julio de 2020 de <https://www.electronjs.org/>
26. Fetch API. (s.f.). Recuperado el 17 de julio de 2020 de [https://developer.mozilla.org/es/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/es/docs/Web/API/Fetch_API)
27. Flask-Cors 3.0.8. (2019). Recuperado el 10 de julio de 2020 de <https://pypi.org/project/Flask-Cors/>



28. Flask. (s.f.). Recuperado el 13 de julio de 2020 de <https://palletsprojects.com/p/flask/>
29. Foltyn, T. (2020). Malware en Linux: una preocupación que crece. Recuperado el 9 de agosto de 2020 de <https://www.welivesecurity.com/la-es/2020/02/21/malware-linux-preocupacion-crece/>
30. Gajrani, J., Sarswat, J., Tripathi, M., Laxmi, V., Gaur, M. S. y Conti, M. (2015). A Robust Dynamic Analysis System Preventing SandBox Detection by Android Malware. *SIN '15: Proceedings of the 8th International Conference on Security of Information and Networks*, 290-295. doi: 10.1145/2799979.2800004
31. Garfinkel, T., Adams, K., Warfield, A. y Franklin, J. (2007). Compatibility is Not Transparency: VMM Detection Myths and Realities. Recuperado el 11 de agosto de 2020 de [https://www.usenix.org/legacy/events/hotos07/tech/full\\_papers/garfinkel/garfinkel\\_html/index.html](https://www.usenix.org/legacy/events/hotos07/tech/full_papers/garfinkel/garfinkel_html/index.html)
32. GDB: The GNU Project Debugger. (s.f.). Recuperado el 25 de julio de 2020 de <https://www.gnu.org/software/gdb/>
33. Goldsborough, P. (2016). The LD\_PRELOAD trick. Recuperado el 27 de julio de 2020 de [http://www.goldsborough.me/c/low-level/kernel/2016/08/29/16-48-53-the\\_ld\\_preload\\_trick/](http://www.goldsborough.me/c/low-level/kernel/2016/08/29/16-48-53-the_ld_preload_trick/)
34. González, C. (2019). ¿Qué es ART? Android Runtime, el sucesor de Dalvik. Recuperado el día 15 de julio de 2020 de <https://androidayuda.com/android/que-es/art-android-runtime/>
35. Hellinger, D., Ming Xuan, L. y Gahlot, P. (2018). Dynamic Analysis of Evasive Malware with a Linux Container Sandbox. Recuperado el 11 de agosto de 2020 de [https://www.researchgate.net/publication/330500642\\_Dynamic\\_Analysis\\_of\\_Evasive\\_Malware\\_with\\_a\\_Linux\\_Container\\_Sandbox](https://www.researchgate.net/publication/330500642_Dynamic_Analysis_of_Evasive_Malware_with_a_Linux_Container_Sandbox)
36. IBM X-Force® Research. (2017). Evading the malware sandbox. Recuperado el 11 de agosto de 2020 de <https://safewayconsultoria.com/wp-content/uploads/2017/11/security-ibm-security-services-se-technical-white-paper-sew03166usen-20171009-2.pdf>
37. importlib. (s.f.) Recuperado el 23 de julio de 2020 de <https://docs.python.org/3/library/importlib.html>

38. Infosec Institute. (2015). De-Obfuscating and Reversing the User-Mode Agent Dropper. Recuperado el 11 de agosto de 2020 de <https://resources.infosecinstitute.com/step-by-step-tutorial-on-reverse-engineering-malware-the-zeroaccessmaxsmiscer-crimeware-rootkit/>
39. ISO/IEC 25010. (s.f.). Recuperado el 9 de julio de 2020 de <https://iso25000.com/index.php/normas-iso-25000/iso-25010>
40. Jablonski, J. (s.f.). Python 3's f-Strings: An Improved String Formatting Syntax. Recuperado el 9 de julio de 2020 de <https://realpython.com/python-f-strings/>
41. Java Native Access. (s.f.). Recuperado el 13 de julio de 2020 de <https://github.com/java-native-access/jna>
42. Java Native Interface. (s.f.). Recuperado el 13 de julio de 2020 de [https://en.wikipedia.org/wiki/Java\\_Native\\_Interface](https://en.wikipedia.org/wiki/Java_Native_Interface)
43. Julian. (2019). Analyzing ELF Binaries with Malformed Headers Part 1 - Emulating Tiny Programs. Recuperado el 13 de julio de 2020 de <https://binaryresearch.github.io/2019/09/17/Analyzing-ELF-Binaries-with-Malformed-Headers-Part-1-Emulating-Tiny-Programs.html>
44. Kedrowitsch, A., Yao, D., Wang, G. y Cameron, K. (2017). A First Look: Using Linux Containers for Deceptive Honeypots. *SafeConfig '17: Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense*, 15-22. doi: 10.1145/3140368.3140371
45. Keshet, Y. (2020). Half of the Malware Detected In 2019 Was Classified As Zero-Day Threats, Making It The Most Common Malware To Date. Recuperado el día 17 de agosto de 2020 de <https://www.cynet.com/blog/half-of-the-malware-detected-in-2019-was-classified-as-zero-day-threats-making-it-the-most-common-malware-to-date/>
46. Kirat, D., Vigna, G. y Kruegel, C. (2011). BareBox: Efficient Malware Analysis on Bare-Metal. *ACSAC '11: Proceedings of the 27th Annual Computer Security Applications Conference*, 403-412. doi: 10.1145/2076732.2076790
47. Kirat, D., Vigna, G. y Kruegel, C. (2014). BareCloud: Bare-metal Analysis-based Evasive Malware Detection. *SEC'14: Proceedings of the 23rd USENIX conference on Security Symposium*. Recuperado el 11 de agosto de 2020 de

- <https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-kirat.pdf>
48. Klayman, N. (2018). Vue CLI Plugin Electron Builder. Recuperado el 7 de agosto de 2020 de <https://nklayman.github.io/vue-cli-plugin-electron-builder/>
49. Levine, J. G., Grizzard, J. B., Hutto, P. W. y Owen, H. L. (2004). A Methodology to Characterize Kernel Level Rootkit Exploits that Overwrite the System Call Table. *IEEE SoutheastCon, 2004: Proceedings*. doi: 10.1109/SECON.2004.1287894
50. Lindorfer, M., Kolbitsch, C. y Milani Comparetti, P. (2011) Detecting Environment-Sensitive Malware. *RAID: Proceedings of the International Symposium on Recent Advances in Intrusion Detection*. doi: 10.1007/978-3-642-23644-0\_18
51. Linux. (s.f.). Wikipedia. Recuperado el día 19 de agosto de 2020 de <https://en.wikipedia.org/wiki/Linux>
52. López, D. (2020). Una caja de arena para probar sin riesgos: así funciona un 'sandbox'. Recuperado el 22 de agosto de 2020 de <http://blog.orange.es/otros/sandbox/>
53. LSB Workgroup, The Linux Foundation. (2015). Filesystem Hierarchy Standard. Recuperado el día 3 de agosto de [https://refspecs.linuxfoundation.org/FHS\\_3.0/fhs-3.0.pdf](https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf)
54. Memcached. (s.f.). Recuperado el 13 de julio de 2020 de <https://www.python.org/dev/peps/pep-3333/>
55. mrexodia, Lavrijsen, M., Sutherland, G. y hfiref0x. (s.f.). Al-Khaser. Recuperado el 3 de agosto de 2020 de <https://github.com/LordNoteworthy/al-khaser>
56. NDK de Android. (s.f.). Recuperado el 11 de agosto de 2020 de <https://developer.android.com/ndk>
57. Newman, C. y Klyne, G. (2002). RFC3339: Date and Time on the Internet: Timestamps. doi: 10.17487/RFC3339
58. Ortega, A. (s.f.) Pafish. Recuperado el 2 de agosto de 2020 de <https://github.com/a0rtega/pafish>
59. Paleari, R., Martignoni, L., Fresi Roglia, G. y Bruschi, D. (2009). A fistful of red-pills: How to automatically generate procedures to detect CPU emulators. *WOOT'09: Proceedings of the 3rd USENIX conference on Offensive technologies*. Recuperado

- el 11 de agosto de 2020 de [https://static.usenix.org/event/woot09/tech/full\\_papers/paleari.pdf](https://static.usenix.org/event/woot09/tech/full_papers/paleari.pdf)
60. Petsas, T., Voyatzis, G. y Athanasopoulos, E. (2014). Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware. *EuroSec '14: Proceedings of the Seventh European Workshop on System Security*, Artículo 5, 1-6. doi: 10.1145/2592791.2592796
61. Política del mismo origen. (s.f.). Wikipedia. Recuperado el 10 de julio de 2020 de [https://es.wikipedia.org/wiki/Pol%C3%ADtica\\_del\\_mismo\\_origen](https://es.wikipedia.org/wiki/Pol%C3%ADtica_del_mismo_origen)
62. Proceso de firma Signature Version 4. (s.f.). Recuperado de [https://docs.aws.amazon.com/es\\_es/general/latest/gr/signature-version-4.html](https://docs.aws.amazon.com/es_es/general/latest/gr/signature-version-4.html)
63. QEMU. (s.f.) Recuperado el 15 de agosto de 2020 de <https://www.qemu.org/>
64. QPython. (s.f.). Recuperado el 13 de julio de 2020 de <https://www.qpython.org/>
65. React. (s.f.). Recuperado el 18 de julio de 2020 de <https://es.reactjs.org/>
66. Redis. (s.f.). Recuperado el 13 de julio de 2020 de <https://redis.io/>
67. redis. (s.f.). Recuperado el 27 de julio de 2020 de <https://pypi.org/project/redis/>
68. Registro de Windows. (s.f.). Wikipedia. Recuperado el 20 de julio de 2020 de [https://es.wikipedia.org/wiki/Registro\\_de\\_Windows](https://es.wikipedia.org/wiki/Registro_de_Windows)
69. requests. (s.f.). Recuperado el 31 de julio de 2020 de <https://requests.readthedocs.io/en/master/>
70. Roccia, T., Rivero Lopez, M. y Shah, C. (2019). Evolution of Malware Sandbox Evasion Tactics - A Retrospective Study. Recuperado el 9 de agosto de 2020 de <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/evolution-of-malware-sandbox-evasion-tactics-a-retrospective-study/>
71. Sandbox\_tester. (s.f.). Recuperado el 2 de agosto de 2020 de [https://github.com/MRGEffitas/Sandbox\\_tester](https://github.com/MRGEffitas/Sandbox_tester)
72. Scripting Layer for Android. (s.f.). Recuperado el 13 de julio de 2020 de [https://en.wikipedia.org/wiki/Scripting\\_Layer\\_for\\_Android](https://en.wikipedia.org/wiki/Scripting_Layer_for_Android)
73. Seals, T. (2014). Malware No Longer Avoids Virtual Machines. *Infosecurity Magazine*. Recuperado el 13 de julio de 2020 de <https://www.infosecurity-magazine.com/news/malware-no-longer-avoids-virtual/>
74. sems. (s.f.). Recuperado el 3 de agosto de 2020 de <https://github.com/AlicanAkyol/sems>

75. Shrivathsan, M. (2012). What is MRD? Market Requirements Document. Recuperado el 9 de agosto de 2020 de <http://pmblog.accompa.com/2012/05/24/what-is-mrd-market-requirements-document/>
76. Skuratovich, S. (s.f.). InviZzible. Recuperado el 2 de agosto de 2020 de <https://github.com/CheckPointSW/InviZzible>
77. sqlite3 - DB-API 2.0 interface for SQLite databases. (s.f.). Recuperado el 10 de julio de 2020 de <https://docs.python.org/3.6/library/sqlite3.html>
78. strace. (s.f.). Recuperado el 17 de agosto de 2020 de <https://strace.io/>
79. The OpenAPI Specification: a broadly adopted industry standard for describing modern APIs. (s.f.). Recuperado el 7 de agosto de 2020 de <https://www.openapis.org/>
80. VirtualBox. (s.f.). Recuperado el 5 de julio de 2020 de <https://www.virtualbox.org/>
81. VMDE Project. (2013). VMDE. Recuperado el 6 de agosto de 2020 de <https://github.com/hfiref0x/VMDE>
82. VMWare. (s.f.). Recuperado el 5 de julio de 2020 de <https://www.vmware.com/ar.html>
83. Vue. (s.f.). Recuperado el 18 de julio de 2020 de <https://vuejs.org/>
84. What Is SQLite? (s.f.). Recuperado el 10 de julio de 2020 de <https://www.sqlite.org/index.html>
85. What's LXC? (s.f.). Recuperado el 19 de agosto de 2020 de <https://linuxcontainers.org/lxc/introduction/>
86. windows.h. (s.f.). Wikipedia. Recuperado el 12 de agosto de 2020 de <https://en.wikipedia.org/wiki/Windows.h>
87. Wireshark. (s.f.). Recuperado el 17 de agosto de 2020 de <https://www.wireshark.org/>
88. Wixey, M. (2017). SandGrox: Detecting sandboxes. Recuperado el 3 de agosto de 2020 de <https://www.pwc.co.uk/issues/cyber-security-services/research/sandgrox.html>
89. Xen Project. (s.f.). Recuperado el 15 de agosto de 2020 de <https://xenproject.org/>

## Apéndice A - Compendio de técnicas de anti-análisis basadas en el reconocimiento del entorno

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
1	Tratar de reenviar paquetes ICMP	Android	Identificar emuladores	Detección de anomalías de conectividad	El emulador del SDK de Android es incapaz de reenviar paquetes ICMP ( <i>Internet Control Message Protocol</i> ).	4
2	Verificar la propiedad Build.USER	Android	Identificar emuladores	Recopilación de información a nivel sistema operativo	En emuladores es común que esta propiedad tome un valor predeterminado, como "android-build".	6
3	Verificar el espacio de direcciones de la red	Android	Identificar emuladores	Recopilación de información a nivel sistema operativo	Suele ser "10.0.2/24" en emuladores.	6
4	Verificar la presencia del archivo "/proc/misc"	Android	Identificar emuladores	Recopilación de información a nivel sistema operativo	La presencia de dicho archivo solo se da en un emulador.	6
5	Verificar la presencia del archivo "/proc/ioports"	Android	Identificar emuladores	Recopilación de información a nivel sistema operativo	La presencia de dicho archivo solo se da en un emulador.	6

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
6	Verificar la presencia del archivo "/sys/devices/virtual/misc/cpu_dma_latency/uevent"	Android	Identificar emuladores	Recopilación de información a nivel sistema operativo	La presencia de dicho archivo solo se da en un emulador.	6
7	Verificar la presencia del archivo "/proc/sys/net/ipv4/tcp_syncookies"	Android	Identificar emuladores	Recopilación de información a nivel sistema operativo	La presencia de dicho archivo solo se da en un emulador.	6
8	Verificar la presencia del directorio "/proc/uid_stat"	Android	Identificar emuladores	Recopilación de información a nivel sistema operativo	La presencia de dicho directorio solo se da en un dispositivo real.	6
9	Verificar el número telefónico	Android	Identificar emuladores	Recopilación de información a nivel sistema operativo	Es común en los emuladores de Android que el número de teléfono tome un valor por defecto, como "155552155ZZ", donde los últimos dos dígitos pueden variar.	6
10	Verificar IMEI	Android	Identificar emuladores	Recopilación de información del hardware	Es común en los emuladores de Android que el número de IMEI ( <i>International Mobile Equipment Identity</i> ) tome un valor por defecto, como "0000000000000000".	6
11	Verificar la propiedad Build.BOARD	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "unknown".	6
12	Verificar la propiedad Build.BRAND	Android	Identificar emuladores	Recopilación de información del hardware	Es común en los emuladores de Android que la marca del equipo tome una denominación por defecto, como "generic".	6

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
13	Verificar la propiedad Build.DEVICE	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "generic".	6
14	Verificar la propiedad Build.FINGERPRINT	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "generic".	6
15	Verificar la propiedad Build.HARDWARE	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "goldfish".	6
16	Verificar la propiedad Build.MANUFACTURER	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "unknown".	6
17	Verificar la propiedad Build.MODEL	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "sdk" o "generic".	6
18	Verificar la propiedad Build.PRODUCT	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "sdk".	6
19	Verificar la propiedad Build.SERIAL	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "null".	6
20	Verificar la propiedad Build.TAGS	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "test-keys".	6



Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
21	Verificar el valor devuelto por BluetoothAdapter.getDefaultAdapter()	Android	Identificar emuladores	Recopilación de información del hardware	El sensor de Bluetooth típicamente se encuentra ausente en emuladores, por lo que el método mencionado devuelve NULL.	6
22	Verificar IMSI	Android	Identificar emuladores	Recopilación de información del hardware	Es común en los emuladores de Android que el número IMSI ( <i>International Mobile Subscriber Identity</i> ) tome un valor por defecto, como "310260000000000".	6
23	Verificar la propiedad Build.HOST	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "android-test".	6
24	Verificar la propiedad Build.ID	Android	Identificar emuladores	Recopilación de información del hardware	En emuladores es común que esta propiedad tome un valor predeterminado, como "FRF91".	6
25	Analizar la lista de contactos	Android	Identificar uso humano histórico	Recopilación de información a nivel sistema operativo	Una lista vacía o limitada incentiva a sospechar que se trata de un entorno de análisis.	6
26	Analizar el registro de llamadas.	Android	Identificar uso humano histórico	Recopilación de información a nivel sistema operativo	Un registro vacío o escaso incentiva a sospechar que se trata de un entorno de análisis.	6
27	Analizar los SMS almacenados	Android	Identificar uso humano histórico	Recopilación de información a nivel sistema operativo	La falta o escasez de SMS presentes en el sistema incentiva a sospechar que se trata de un entorno de análisis.	6

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
28	Detectar eventos de touch	Android	Identificar uso humano vigente	Detección de eventos	Monitorear los eventos de toque, pulsación, arrastre, etc., que se generan en el sistema es un indicio de que una persona está interactuando con el dispositivo y, por ende, no se trata de un ambiente de análisis.	6
29	Identificar llamadas entrantes y/o salientes	Android	Identificar uso humano vigente	Detección de eventos	Que el dispositivo participe activamente de llamadas es un indicio de que se trata de un equipo real que está siendo utilizado por un individuo.	6
30	Identificar SMS enviados y/o recibidos	Android	Identificar uso humano vigente	Detección de eventos	Que el dispositivo activamente reciba o envíe mensajes de texto es un indicio de que se trata de un equipo real que está siendo utilizado por un individuo.	6
31	Analizar el estado de la batería	Android	Identificar uso humano vigente	Estudio de la evolución temporal del entorno	Que el nivel de batería varíe de forma esperable con el tiempo sugiere que el equipo es un dispositivo físico real y que está siendo utilizado por una persona.	6
32	Verificar proveedor de red	Android	Reconocer la aplicación de virtualización	Recopilación de información a nivel sistema operativo	Es común en los emuladores de Android que el proveedor de red tome un nombre genérico, como "Android".	6
33	Verificar la presencia del directorio "/sys/devices/virtual/ppp"	Android	Reconocer la aplicación de virtualización	Recopilación de información a nivel sistema operativo	La presencia de dicho directorio solo se da en un dispositivo real.	6
34	Verificar la presencia del directorio "/sys/devices/virtual/switch/"	Android	Reconocer la aplicación de virtualización	Recopilación de información a nivel sistema operativo	La presencia de dicho directorio solo se da en un dispositivo real.	6

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
35	Verificar la presencia del archivo “/sys/module/alarm/parameters”	Android	Reconocer la aplicación de virtualización	Recopilación de información a nivel sistema operativo	La presencia de dicho archivo solo se da en un dispositivo real.	6
36	Verificar la presencia del directorio “/sys/devices/system/cpu/cpu0/cpufreq”	Android	Reconocer la aplicación de virtualización	Recopilación de información a nivel sistema operativo	La presencia de dicho directorio solo se da en un dispositivo real.	6
37	Verificar la presencia del archivo “/sys/devices/virtual/misc/android_adb”	Android	Reconocer la aplicación de virtualización	Recopilación de información a nivel sistema operativo	La presencia de dicho archivo solo se da en un dispositivo real.	6
38	Consultar cual es el proceso con PID 1	Linux	Identificar contenedores	Recopilación de información a nivel de sistema operativo	Cada contenedor tiene su propio espacio de nombre y, en consecuencia, mantiene su propia numeración de procesos. En Linux, siempre el primer proceso es el de init, el cual es el encargado de generar el resto de procesos. Los inits disponibles son pocos (entre ellos encontramos BSD init, OpenRC, SysV, Solaris SMF, systemd y upstart) por lo que pueden ser fácilmente enumerados. Si el proceso con PID 1 no corresponde a ningún init mencionado previamente, entonces puede determinarse que el proceso ejecutado está dentro de un contenedor. Para implementar esta técnica, se puede recuperar el nombre del proceso de la primera línea del seudo archivo “/proc/1/sched”.	8

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
39	Buscar un archivo <code>"/.dockerenv"</code> en el sistema	Linux	Identificar contenedores	Recopilación de información a nivel de sistema operativo	El mencionado archivo es gestionado internamente por Docker.	8
40	Buscar un archivo <code>"/.dockerinit"</code> en el sistema	Linux	Identificar contenedores	Recopilación de información a nivel de sistema operativo	El mencionado archivo es gestionado internamente por Docker.	8
41	Consultar la existencia de los directorios <code>"/proc/vz"</code> y <code>"/proc/bc"</code>	Linux	Identificar contenedores	Recopilación de información a nivel de sistema operativo	Dichos directorios son necesarios por parte de OpenVZ, por lo cual revelan la naturaleza del sistema.	5
42	Revisar el contenido de <code>"/proc/meminfo/"</code>	Linux	Identificar contenedores	Recopilación de información del hardware	Del seudo archivo mencionado se puede extraer información sobre la memoria real del sistema, a pesar de encontrarse dentro de un contenedor; si la información discrepa de la obtenida por otros medios, se puede inferir que el entorno está conformado por uno.	8
43	Revisar el contenido de <code>"/proc/kallsyms"</code>	Linux	Identificar contenedores	Requerimientos del sistema respecto a privilegios	Si no se cuenta con privilegios de superusuario, las direcciones de memoria que aparecen en <code>"/proc/kallsyms"</code> son todas 0. Debido a que se está interactuando directamente con el kernel, los privilegios asociados a esta consulta responden a los del contenedor en el contexto del sistema operativo anfitrión y no a los del usuario dentro del contenedor. Un valor inesperado de <code>kallsyms</code> indica una discrepancia de permisos y, por ende, implica la presencia de un contenedor.	Propuesta autónoma

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
44	Revisar el valor de TracerPid en “/proc/self/status”	Linux	Identificar depuradores / Identificar rastreadores	Recopilación de información a nivel de sistema operativo	Si existe algún depurador o rastreador monitoreando el proceso, el valor de TracerPid es distinto de 0. El número obtenido es el ID del proceso que está llevando adelante el monitoreo.	8
45	Invocar la llamada al sistema <i>ptrace</i> sobre sí mismo con las banderas PTRACE_TRACEME o PTRACE_ATTACH	Linux	Identificar depuradores / Identificar rastreadores	Trampas	Un proceso puede tener a lo sumo un rastreador adjudicado. Si la susodicha llamada falla (tras lo cual retorna el código de error -1), es porque el proceso ya está siendo monitoreado. También es una técnica efectiva para que evitar que otro proceso pueda analizar a aquel que no quiera ser analizado.	5, 8
46	Reconocer la presencia de la variable de ambiente LD_PRELOAD	Linux	Identificar ganchos	Identificación de parámetros de ejecución	Dicha variable se utiliza para especificar ciertas librerías que debería utilizar un proceso al momento de iniciar su ejecución. En ambientes de análisis suele utilizarse para sobrescribir ciertas librerías del sistema para incorporar funcionalidad de monitoreo y estudio.	5
47	Invocar <i>ptrace</i> sobre sí mismo dos veces	Linux	Identificar ganchos	Trampas	Una solución para evitar que un proceso descubra que está siendo monitoreado mediante el truco de <i>ptrace</i> es reemplazar dicha llamada al sistema con una función que indique que la operación ha sido exitosa. Si la función no esta preparada para que en llamadas subsecuentes se devuelva un error (indicando así que el proceso ya está siendo rastreado), los resultados siguientes son incongruentes con el primero y se desvela la inyección de librería.	8

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
48	Revisar si existe al archivo “/proc/xen/capabilities”	Linux	Identificar hipervisores	Recopilación de información a nivel de sistema operativo	La existencia de dicho archivo revela que el sistema está siendo virtualizado mediante el hipervisor Xen.	5
49	Comparar el contenido de “/proc/1/mountinfo” con el de “/proc/self/mountinfo”	Linux	Identificar jaulas chroot	Recopilación de información a nivel de sistema operativo	En caso de que el proceso en ejecución se encuentre en una jaula chroot, su información de montaje discrepa en relación a la del proceso 1, el cual es el proceso de init en sistemas Linux.	5
50	Revisar el contenido de “/sys/class/dmi/id/product_name”	Linux	Identificar máquinas virtuales alojadas	Recopilación de información del hardware	Un entorno sobre VirtualBox o VMware informa “VirtualBox” o “VMWare” como nombre del producto, según corresponda.	5
51	Comparar el contenido de “/boot/System.map-<versión del kernel>” y “/proc/kallsyms”	Linux	Identificar modificaciones en el entorno	Recopilación de información a nivel de sistema operativo	En “/boot/System.map-<versión del kernel>” queda resguarda la configuración de mapeo de los símbolos del sistema para el kernel analizado, mientras que en “/proc/kallsyms” puede consultarse la configuración actual. Las discrepancias que pueden darse entre ambos es un indicador de la presencia de un gancho o de un rootkit. Se requieren permisos de superusuario para consultar adecuadamente la información de ambos archivos.	Propuesta autónoma
52	Remover toda información sobre las secciones ELF del ejecutable	Linux	Reconocer el uso de herramientas de análisis	Manipulación de cabeceras ELF	La presencia de las susodichas cabeceras no es de interés para el sistema operativo al momento de ejecutar el binario, pero sí puede impedir que un instrumento de análisis sea capaz de iniciarlo debido a ser más riguroso con el formato.	5

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
53	Reportar información falsa en la cabecera del ELF	Linux	Reconocer el uso de herramientas de análisis	Manipulación de cabeceras ELF	Un ejemplo es reportar un sistema operativo distinto al objetivo real del ejecutable (FreeBSD, por ejemplo). Nuevamente el binario puede ser iniciado sin inconvenientes, excepto en el caso de un instrumento que esté trabajando con el mismo, el cual puede ser más estricto que el sistema operativo.	5
54	ELF inválido	Linux	Reconocer el uso de herramientas de análisis	Manipulación de cabeceras ELF	Se trata de generar información de secciones malformada o corrupta. Esto puede deberse a valores inválidos de <code>e_shoff</code> (desplazamiento de la tabla de cabecera de sección), de <code>e_shnum</code> (número de entradas en la susodicha tabla) o de <code>e_shentsize</code> (tamaño de los registros de sección) en la cabecera del ELF. El sistema operativo podría ejecutar el binario a pesar de estas condiciones pero una herramienta de análisis podría verse restringida.	5
55	Ejecutar el comando <code>systemd-detect-virt</code>	Linux	Reconocer la aplicación de virtualización	Recopilación de información a nivel de sistema operativo	El <code>init systemd</code> proporciona el susodicho comando, el cual es capaz de detectar virtualización de varios tipos, desde emuladores e hipervisores hasta contenedores. Particularmente informa el software de virtualización utilizado.	11
56	Consultar el usuario y el grupo del proceso desde <code>/proc/self/status</code>	Linux	Reconocer objetivos particulares	Identificación de parámetros de ejecución	Un malware puede preguntar qué usuario y qué grupo lo están ejecutando para determinar qué acciones tomar. Esta información se puede obtener de los campos <code>"Uid"</code> , <code>"Gid"</code> y <code>"Groups"</code> del seudo archivo mencionado.	5

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
57	Especificar un cargador de ELF	Linux	Reconocer objetivos particulares / Continuar con la ejecución	Manipulación de cabeceras ELF	El formato ELF permite especificar un cargador específico. De no estar presente en el sistema, el programa que lo requiera no puede ser inicializado.	5
58	Especificar determinadas bibliotecas dinámicas necesarias	Linux	Reconocer objetivos particulares / Continuar con la ejecución	Requerimientos del sistema respecto a librerías enlazadas dinámicamente	Los archivos binarios enlazados dinámicamente requieren de la presencia de todas sus bibliotecas solicitadas para poder ser ejecutados correctamente.	5
59	Revisar el contenido de "/sys/class/dmi/id/sys_vendor"	Linux	Reconocer virtualización a nivel de hardware	Recopilación de información del hardware	En QEMU el valor de fabricante arrojado es "QEMU", mientras que en VirtualBox se obtiene "innotek GmbH".	5
60	Revisar el contenido de "/proc/cpuinfo"	Linux	Reconocer virtualización a nivel de hardware	Recopilación de información del hardware	El seudo archivo indicado contiene información sobre el procesador. Discrepancias en datos informados, tales como el modelo, la cantidad de núcleos disponibles o el estado de la bandera de presencia de un hipervisor revelan un entorno virtualizado.	5
61	Revisar el contenido de "/proc/scsi/scsi"	Linux	Reconocer virtualización a nivel de hardware	Recopilación de información del hardware	El seudo archivo indicado mantiene un listado de dispositivos SCSI conectados al equipo. La ausencia del mismo o un contenido sospechoso es indicio de que el sistema está siendo virtualizado.	5



Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
62	Identificar procesos en el sistema	Linux	Varias	Recopilación de información a nivel de sistema operativo	Se puede realizar para identificar herramientas de virtualización, instrumentos de análisis, objetivos, etc. Escanear el contenido del sistema de archivos virtual "/proc" o utilizar el comando ps son alternativas para llevarlo a cabo.	5
63	Necesidad de privilegios de superusuario	Multisistema	Continuar con la ejecución	Requerimientos del sistema respecto a privilegios	El software no es capaz de llevar adelante ciertas operaciones (o incluso ser ejecutado) a menos que cuente con los privilegios suficientes.	5
64	Cambiar el comportamiento según los privilegios obtenidos	Multisistema	Evitar la detección	Requerimientos del sistema respecto a privilegios	Un software malicioso es capaz de actuar de forma distinta según los privilegios que se la han sido otorgados y así burlar ciertos entornos de análisis.	5
65	Intentar resolver dominios falsos	Multisistema	Identificar anomalías de red	Detección de anomalías de conectividad	Esta técnica se basa en solicitarle a un DNS público que resuelva ciertos dominios que se sabe que son falsos. Que esta tarea sea exitosa es un indicio de que el proceso se encuentra en un sandbox.	10
66	Verificar el contenido descargado a través de la web	Multisistema	Identificar anomalías de red	Detección de anomalías de conectividad	Es posible que un sandbox simule conectividad a la red. Para reconocer esta situación, se puede pedir un recurso conocido desde una página web benigna y verificar que el valor obtenido tras aplicar una función resumen a la descarga sea el esperado.	2
67	Comprobar la velocidad de la conexión de internet	Multisistema	Identificar anomalías de red	Pruebas de rendimiento	Una ancho de banda extremadamente alto induce a suponer que el sistema no se trata de un ambiente real.	3, 4

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
68	Buscar listas de revocación de certificados	Multisistema	Identificar anomalías de red	Recopilación de información a nivel de sistema operativo	La presencia de estos archivos en el sistema son un indicador de un uso activo de la red.	4
69	Consultar el caché de DNS	Multisistema	Identificar anomalías de red	Recopilación de información a nivel de sistema operativo	Un caché vacío o con simplemente información referida a localhost es un indicio de que el sistema no ha utilizado la red y por ende es potencialmente un sandbox.	4
70	Consultar los SSID Wi-Fi en caché	Multisistema	Identificar anomalías de red	Recopilación de información a nivel de sistema operativo	La ausencia de este contenido es dispositivos que supuestamente cuentan con placas Wi-Fi induce a pensar de que se tratan de ambientes de análisis.	4
71	Determinar si hay conectividad	Multisistema	Identificar anomalías de red	Requerimientos del sistema respecto a la conexión de red	Un malware podría necesitar de conexión a internet para operar, o simplemente desestimar el sistema alcanzado porque o no lo considera su objetivo por la falta de conectividad o porque lo toma como un sandbox ante este hecho.	1, 2, 4
72	Identificar al proceso padre y recuperar su nombre	Multisistema	Identificar depuradores	Recopilación de información a nivel de sistema operativo	Se puede comparar el nombre del proceso padre contra el de depuradores conocidos. También se puede buscar que el nombre coincida con el de un intérprete de línea comandos para asumir, en caso afirmativo, que el proceso no está bajo estudio. En Windows se puede preguntar por "explorer.exe", asumiendo que un proceso que fue iniciado tras un doble click sobre un ícono no está siendo analizado. Para este propósito se pueden utilizar las funciones CreateToolhelp32Snapshot() o NtQuerySystemInformation().	1

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
73	Buscar el código de operación "0xCC" mediante un análisis de integridad del proceso	Multisistema	Identificar depuradores	Revisión del estado de la memoria y de los registros	El susodicho código de operación corresponde a la instrucción INT3 de los procesadores x86. Se destina su uso a depuradores para que puedan insertar puntos de ruptura en el código.	1
74	Revisar el contenido de los registros DR	Multisistema	Identificar depuradores	Revisión del estado de la memoria y de los registros	En la arquitectura x86, el procesador contiene unos registros destinados para depuración denominados DR, en los cuales se pueden almacenar direcciones de puntos de ruptura. En Windows, la función GetThreadContext() permite consultar el contenido de los registros de un hilo.	1
75	Observar los efectos del caché del procesador	Multisistema	Identificar emuladores	Pruebas de rendimiento	La simulación del caché del procesador es una tarea ardua, por lo que podría no estar soportada en un emulador. Para llevar adelante esta prueba, se debe ejecutar una función un cierto número de veces. Si se trata de un equipo real, la primera invocación debería ser más lenta que las siguientes.	1
76	Ejecución de la instrucción "or %bh, 0x04(%ebx)" en arquitecturas x86	Multisistema	Identificar emuladores	Reconocimiento de artefactos de virtualización de CPU	La instrucción calcula el resultado de la operación o binaria del valor en el registro %bh con el valor almacenado en memoria en la dirección %ebx + 0x04, tras lo cual guarda el resultado en esta última. Debido a un error en QEMU, la instrucción referencia a la dirección equivocada, lo que provoca o un fallo de página o que el valor final en la dirección %ebx + 0x04 no corresponda con el resultado esperado. Esto delata la presencia del emulador previamente mencionado.	1

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
77	Verificar el nombre del archivo ejecutable propio	Multisistema	Identificar herramientas de análisis de malware	Recopilación de información a nivel de sistema operativo	Se trata de comparar el nombre del archivo contra uno esperado. Esta técnica se fundamenta en que algunas plataformas de análisis de malware modifican el nombre de los ficheros que reciben previo a su estudio. También puede verificarse contra nombres por defecto, tales como "sample01", "virus02" o "malware03".	2, 5, 12
78	Indagar el nombre del archivo ejecutable propio	Multisistema	Identificar herramientas de análisis de malware	Recopilación de información a nivel de sistema operativo	Similar al anterior, pero en este caso se buscan subcadenas típicamente impuestas por plataformas de análisis de malware, tal como "Virus", "Sample" o "Sandbox".	9
79	Ejecutar la instrucción CPUID en procesadores x86	Multisistema	Identificar máquinas virtuales alojadas / Identificar hipervisores	Recopilación de información del hardware	Los procesadores de Intel y AMD reservan el bit número 31 del registro ECX para indicar la presencia de un hipervisor (1 en tal caso) tras la ejecución de la instrucción CPUID.	3, 8
80	Detectar parcheo de sueño mediante la consulta de los ticks del procesador	Multisistema	Identificar modificaciones en el entorno	Detección de la aceleración de la ejecución del proceso	Algunos entornos están preparados para identificar la ejecución de Sleep() y contrarrestarlo acelerando el tiempo. Esta medida puede ser reconocida por un malware que utilice Sleep() usando en conjunto la instrucción de máquina x86 RDTSC (Read Time-Stamp Counter), que informa la cantidad de impulsos ( <i>ticks</i> ) del procesador desde que se reinició.	1

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
81	Uso de hilos para reconocer la simulación de llamadas a Sleep()	Multisistema	Identificar modificaciones en el entorno	Detección de la aceleración de la ejecución del proceso	Esta técnica se base en la ejecución de dos hilos: uno realiza ciertos cálculos mientras que el otro duerme. En condiciones normales el hilo durmiente debería ser el último en finalizar; en caso contrario, esta situación es un indicador de que las llamadas a Sleep() están siendo simuladas y por ende el sistema es un ambiente de análisis.	2
82	Identificar la aceleración de la ejecución de código	Multisistema	Identificar modificaciones en el entorno / Identificar herramientas de análisis	Detección de la aceleración de la ejecución del proceso	Algunos ambientes de análisis aceleran la ejecución de sus muestras con el fin de vencer técnicas de estancamiento. Esta situación puede ser identificada verificando que el tiempo real transcurrido de llevar adelante alguna actividad o conjunto de actividades típicamente longevas supere un mínimo esperado. En Windows puede usarse la función GetTickCount() para tomar el tiempo transcurrido desde el último inicio del sistema.	12
83	Comprobar el tipo y la cantidad de archivos presentes en directorios de usuario	Multisistema	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	La ausencia o escasez de ficheros en carpetas como "Escritorio", "Documentos" o "Imágenes", como a su vez incongruencias de su tipos respecto a su ubicación pueden ser tomados como indicios de un ambiente de análisis.	2
84	Comprobar el historial de medios de almacenamiento USB conectados	Multisistema	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	Que no se haya conectado nunca un dispositivo de tal índole sugiere que el sistema se trata de un entorno de análisis.	2

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
85	Buscar programas instalados de uso común	Multisistema	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	Entre los mismos podemos encontrar Skype, navegadores como Chrome y Firefox, herramientas de ofimática como Microsoft Office o LibreOffice, lectores de PDF, etc. En Android, la ausencia de aplicaciones como “Google Play Store”, “Google Maps” o “Google Play Services” es un fuerte indicio de que se está tratando con un entorno de análisis.	4, 6
86	Revisar si existe historial de navegación web	Multisistema	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	La ubicación de esta información varía de navegador a navegador.	4
87	Revisar si existe contenido en la papelera de reciclaje	Multisistema	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	Una papelera con archivos es un indicador de que el sistema fue usado por un humano.	4
88	Revisar si existen volcados de emergencia ( <i>crash dumps</i> )	Multisistema	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	Se presume que la existencia de esta información solo puede darse como consecuencia de la interacción continuada de un humano con el sistema.	4
89	Estudiar los tiempos muertos del usuario	Multisistema	Identificar uso humano vigente	Detección de eventos	Se basa en medir los tiempos entre entradas provistas por el usuario, como la pulsación de una tecla o el movimiento de un ratón, para determinar con una persona está usando el sistema. La ausencia de estos eventos o su comportamiento poco natural implican que se está en un entorno de análisis. En Windows, las funciones GetTickCount() y GetLastInputInfo() son útiles para implementar esta prueba.	1

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
90	Estudiar las pulsaciones del teclado	Multisistema	Identificar uso humano vigente	Detección de eventos	Se espera que el tiempo entre pulsaciones corresponda al esperado en un ser humano.	4
91	Estudiar la distancia recorrida por el cursor	Multisistema	Identificar uso humano vigente	Estudio de la evolución temporal del entorno	Se espera que tras un lapso de tiempo el cursor se haya desplazado una determinada distancia.	2
92	Estudiar los movimientos del cursor	Multisistema	Identificar uso humano vigente	Estudio de la evolución temporal del entorno	Esta técnica se trata de calcular la velocidad de los movimientos del puntero. Si la misma sobrepasa un valor decidido anticipadamente implica que las acciones están siendo automatizadas. GetCursorPos() es una función útil para este test provista en Windows.	1
93	Estudiar el desplazamiento ( <i>scrolling</i> )	Multisistema	Identificar uso humano vigente	Estudio de la evolución temporal del entorno	Se basa en analizar si los desplazamientos corresponden a los generados por un ser humano. El caso más típico es el del desplazamiento que se produce al detenerse en ciertos puntos específicos de un documento.	9
94	Tomar capturas de pantalla durante un tiempo determinado	Multisistema	Identificar uso humano vigente	Estudio de la evolución temporal del entorno	Las imágenes obtenidas se pueden enviar mediante HTTP a algún receptor (si el sistema lo permite), tras lo cual son susceptibles a ser analizadas manualmente para determinar si el sistema estudiado se trata de un sandbox.	2
95	Estudiar el ritmo de creación de procesos del sistema	Multisistema	Identificar uso humano vigente	Estudio de la evolución temporal del entorno	Se espera que aparezcan nuevos procesos producto de la interacción de un ser humano con el sistema.	4

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
96	Consultar hace cuanto tiempo lleva encendido el sistema desde el último reinicio	Multisistema	Identificar uso humano vigente	Recopilación de información a nivel de sistema operativo	Tiempos excesivamente cortos o largos resultan sospechosos, sobre todo si lo que se está intentando infectar es un equipo de escritorio.	2, 4
97	Consultar la fecha de último acceso de ciertos archivos	Multisistema	Identificar uso humano vigente	Recopilación de información a nivel de sistema operativo	Se espera encontrar ficheros con fechas de último acceso recientes.	4
98	Verificar que el portapapeles tenga contenido	Multisistema	Identificar uso humano vigente	Recopilación de información a nivel de sistema operativo	Se espera que en una sesión activa de un usuario el mismo haya copiado texto en algún momento.	4
99	Buscar herramientas de análisis instaladas	Multisistema	Reconocer el uso de herramientas de análisis	Recopilación de información a nivel de sistema operativo	Un ejemplo en Windows es el de Debugging Tools.	2
100	Buscar nombres de procesos activos correspondientes a herramientas de análisis	Multisistema	Reconocer el uso de herramientas de análisis	Recopilación de información a nivel de sistema operativo	En Windows, las funciones provistas FindWindow() y FindProcess() son útiles para dicho fin. Un ejemplo de un proceso a buscar podría ser "cwsandbox.exe".	1, 2, 12
101	Consultar direcciones IP locales	Multisistema	Reconocer la aplicación de virtualización	Recopilación de información a nivel de sistema operativo	Esta prueba se basa en comparar el valor de las direcciones IP del sistema contra las predeterminadas de ciertos instrumentos de virtualización. En emuladores de Android suele ser 10.0.2.15.	1, 4, 6, 12



Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
102	Consultar cuál es el controlador de video	Multisistema	Reconocer la aplicación de virtualización	Recopilación de información a nivel de sistema operativo	Es de esperarse encontrar un controlador de video correspondiente a Intel, AMD o Nvidia en un equipo real. En Windows puede consultarse con el comando <code>wmic path win32_VideoController get name</code> .	3
103	Buscar controladores / módulos asociados a herramientas de virtualización	Multisistema	Reconocer la aplicación de virtualización	Recopilación de información a nivel de sistema operativo	En Linux esta búsqueda puede realizarse con el comando <code>lsmod</code> ; por ejemplo, <code>lsmod   grep -i vbox</code> recupera todos los módulos activos cuyos nombres contengan la subcadena "vbox".	8
104	Verificar la hora actual del sistema	Multisistema	Reconocer la aplicación de virtualización	Recopilación de información a nivel de sistema operativo	Si la hora difiere drásticamente de la hora real, es probable de que se trate de un sandbox. Para obtener la hora real se puede consultar a un servidor NTP ( <i>Network Time Protocol</i> ), en caso de contar con acceso a internet.	2
105	Consultar la temperatura del procesador	Multisistema	Reconocer la aplicación de virtualización	Recopilación de información del hardware	Es esperable que en una máquina virtual esta información no pueda ser consultada o tenga un valor imposible. Particularmente en Windows el comando <code>wmic /namespace:\root\wmi PATH MSAcpi_ThermalZoneTemperature get CurrentTemperature /value</code> debería arrojar un resultado vacío si el sistema se encuentra en una máquina virtual.	12
106	Consultar el tamaño de la memoria RAM	Multisistema	Reconocer la aplicación de virtualización	Recopilación de información del hardware	Una cantidad de memoria limitada, por ejemplo 256 MB, es un indicio de virtualización. En Windows puede consultarse con el comando <code>wmic memorychip get capacity</code> .	2, 3

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
107	Consultar el tamaño del disco	Multisistema	Reconocer la aplicación de virtualización	Recopilación de información del hardware	Es de esperarse encontrar en un equipo actual al menos 20 GB de espacio de almacenamiento. En Windows puede consultarse con el comando <i>wmic diskdrive get size</i> .	2, 3
108	Consultar si se encuentra alguna impresora conectada	Multisistema	Reconocer la aplicación de virtualización / Reconocer objetivos particulares	Recopilación de información del hardware	Se puede asumir que un puesto de trabajo debería haber una impresora conectada. En Windows se puede obtener el nombre de la misma con el comando <i>wmic printer get name</i> .	2, 9
109	Verificar que el nombre del procesador corresponda a uno típicamente usado en servidores	Multisistema	Reconocer la aplicación de virtualización / Reconocer objetivos particulares	Recopilación de información del hardware	Esta técnica puede usarse tanto para evadir equipos que deberían ser puestos de trabajo (y por lo cual se pueden asumir como sandboxes) como para identificar certeramente a un servidor que se desea atacar. En Windows puede consultarse desde el Registro.	2, 9
110	Verificar la presencia de un controlador iSCSI	Multisistema	Reconocer objetivos particulares	Recopilación de información a nivel de sistema operativo	iSCSI (de <i>Internet Small Computer Systems Interface</i> ) es un estándar típicamente usado para construir redes de área de almacenamiento. Goza de cierta popularidad dentro de organizaciones, por lo que la existencia del mencionado controlador es un indicio de que el programa ha logrado infiltrarse en una red privada.	10
111	Encriptación dirigida	Multisistema	Reconocer objetivos particulares	Recopilación de información brindada por el entorno	Se encripta la carga útil con una llave que solamente se puede recuperar del sistema objetivo. Una variante es utilizar una red neuronal que se alimente de datos obtenidos del entorno para generar la clave necesaria para desencriptar, lo que dificulta encontrar una medida contra esta técnica.	1, 3, 4

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
112	Consultar la dirección MAC de una interfaz de red particular	Multisistema	Reconocer objetivos particulares	Recopilación de información del hardware	Si un atacante logra obtener esta información de antemano, puede usarla para especificar el equipo a sabotear.	10
113	Usar un gran número de tablas de página	Multisistema	Reconocer virtualización a nivel de hardware	Pruebas de rendimiento	La <i>shadow page table</i> es una estructura de datos mantenida activamente por el gestor de máquina virtual (VMM). La misma replica las acciones del invitado sobre sus propias tablas de páginas. También imita lo referido a la tarea del VMM de traducir la dirección física del invitado a la dirección física del anfitrión. El limitado caché de shadow page puede ser atacado por cansancio usando muchas tablas de páginas, lo que conlleva a un alto número de fallos de página escondidos y consecuentemente a una penalización de rendimiento fácilmente detectable.	7
114	Buscar nombres de procesos activos correspondientes a máquinas virtuales	Multisistema	Reconocer virtualización a nivel de hardware	Recopilación de información a nivel de sistema operativo	Ejemplos a consultar en Windows son "VMtools.exe", "VMwareUser.exe", "VBoxService.exe", "vbox*.exe" y "vmware*.exe" (donde "*" es cualquier secuencia de caracteres).	1, 3
115	Verificar el nombre del procesador contra una lista	Multisistema	Reconocer virtualización a nivel de hardware	Recopilación de información del hardware	La lista puede contener nombres de procesadores aceptables o nombres comúnmente encontrados en máquinas virtuales.	9

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
116	Consultar el número de núcleos del procesador	Multisistema	Reconocer virtualización a nivel de hardware	Recopilación de información del hardware	Se puede comparar el número obtenido con el esperado según el modelo de la CPU. A su vez, se puede asumir que un procesador moderno debería tener al menos dos núcleos. En Windows puede consultarse con el comando <i>wmic cpu get NumberOfCores</i> . En Linux puede extraerse esta información del seudo archivo “/proc/cpuinfo”.	2, 3, 8, 12
117	Consultar el tamaño de la pantalla	Multisistema	Reconocer virtualización a nivel de hardware	Recopilación de información del hardware	Es de esperarse que un equipo cuente con un monitor de más de 800x600 píxeles. En Windows esta información puede obtenerse con el comando <i>wmic desktopmonitor get screenheight, screenwidth</i> .	2, 3

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
118	Muestreo del tiempo de ejecución de una instrucción, función de una librería o llamada al sistema	Multisistema	Varias	Pruebas de rendimiento	<p>Se proponen dos variantes:                      1- Comparar los valores obtenidos contra uno de referencia.                      2- Determinación de la media y la dispersión de la muestra obtenida en busca de gran variabilidad.</p> <p>Las APIs de Windows proporcionan GetTickCount(), QueryPerformanceCounter() y QueryPerformanceFrequency, útiles para este fin. Los procesadores x86 cuentan con el registro TSC (<i>Time Stamp Counter</i>) que cuenta la cantidad de ciclos desde el último reinicio; a su vez, proporcionan la instrucción <i>RDTC (Read Time Stamp Counter)</i> para consultarlo. Una manera de tomar el tiempo agnóstica del lenguaje y del sistema operativo es consultándole la hora a un medio externo a través de la red mediante NTP (<i>Network Time Protocol</i>).</p>	4, 8
119	Consultar el nombre del equipo	Multisistema	Varias	Recopilación de información a nivel de sistema operativo	Un nombre genérico puede ser indicio de un sandbox.	2, 12
120	Inspeccionar el directorio "C:\Windows\System32\Drivers"	Windows	Identificar máquinas virtuales alojadas	Recopilación de información a nivel de sistema operativo	Se busca identificar archivos correspondientes a máquinas virtuales, como "Vmmouse.sys", "vm3dgl.dll" o "VMToolsHook.dll".	1

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
121	Consultar si existe la clave "SYSTEM\CurrentControlSet\Control\VirtualDeviceDrivers" en el registro de Windows	Windows	Identificar máquinas virtuales alojadas	Recopilación de información a nivel de sistema operativo	Esta clave es un indicador de que el equipo está siendo virtualizado.	1
122	Analizar el campo "BeingDebugged" del PEB	Windows	Identificar depuradores	Identificación de parámetros de ejecución	El Bloque de Entorno del Proceso (PEB, del inglés <i>Process Environment Block</i> ) es una estructura de datos que contiene información de un proceso. Existe un PEB por cada proceso activo en el sistema. El campo "BeingDebugged" indica si el proceso en cuestión está siendo depurado. El mismo puede ser consultado directamente o usando las APIs recomendadas por Microsoft, como <code>IsDebuggerPresent()</code> y <code>CheckRemoteDebuggerPresent()</code> .	1
123	Embeber la instrucción "int 2dh" en el código	Windows	Identificar depuradores	Trampas	Cuando se ejecuta la susodicha instrucción, una excepción es generada. Si hay un depurador analizando activamente al proceso en cuestión, dicha excepción es transferida a él, lo que impide que el programa en ejecución pueda gestionarla y lo que desvela la presencia del depurador.	1
124	Consultar los archivos de Microsoft Office recientemente abiertos	Windows	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	La ausencia de esta información en una cantidad esperable da un indicio de que se está tratando con un entorno de análisis. La entrada en el Registro de Windows que debería buscarse para obtener esta información corresponde a "SOFTWARE\Microsoft\Office\VERSION\PRODUCT\File MRU".	3, 12

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
125	Consultar los recursos recientemente abiertos mediante “Ejecutar”	Windows	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	La ausencia de esta información en una cantidad esperable da un indicio de que se está tratando con un entorno de análisis. La entrada en el Registro de Windows que debería buscarse para obtener esta información corresponde a "SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU".	3
126	Consultar las rutas de archivos recientemente ingresadas	Windows	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	La ausencia de esta información en una cantidad esperable da un indicio de que se está tratando con un entorno de análisis. La entrada en el Registro de Windows que debería buscarse para obtener esta información corresponde a "SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\TypedPaths".	3
127	Consultar las búsquedas realizadas recientemente	Windows	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	La ausencia de esta información en una cantidad esperable da un indicio de que se está tratando con un entorno de análisis. La entrada en el Registro de Windows que debería buscarse para obtener esta información corresponde a "SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\WordWheelQuery".	3
128	Consultar información derivada del uso de los diálogos de Windows	Windows	Identificar uso humano histórico	Recopilación de información a nivel de sistema operativo	La ausencia de esta información en una cantidad esperable da un indicio de que se está tratando con un entorno de análisis. En ella se pueden encontrar archivos seleccionados, archivos guardados, etc. La entrada en el Registro de Windows que debería buscarse para obtener esta información corresponde a "SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\*".	3

Nº	Técnica	Sistema operativo	Intención	Metodología	Descripción	Referencias
129	Comprobar la ventana en primer plano con la función <code>GetForegroundWindow()</code>	Windows	Identificar uso humano vigente	Detección de eventos	La técnica propuesta trata de hallar un cambio en la ventana que se encuentra actualmente en primer plano con el fin de reconocer que una persona está utilizando el sistema.	12
130	Consultar la clave de producto de Windows	Windows	Reconocer la aplicación de virtualización	Recopilación de información a nivel de sistema operativo	La clave de producto de Windows tiene un formato particular; si tras consultarla presenta una forma anómala, entonces este hecho es un indicador de que el sistema está siendo virtualizado. A su vez, ciertas herramientas para este fin establecen dicha clave con un valor predeterminado conocido que puede terminar desvelándolos. Esta información se puede obtener con el comando <code>wmic path softwarelicensingservice get OA3xOriginalProductKey</code> .	2
131	Consultar el nombre de dominio de Windows de la red a la que pertenece el equipo	Windows	Reconocer objetivos particulares	Recopilación de información a nivel de sistema operativo	Esta técnica sirve para actuar solo en dispositivos que estén asociados a una red particular que se desea atacar.	2
132	Comprobar el estado del puerto 445	Windows	Reconocer objetivos particulares	Recopilación de información a nivel de sistema operativo	Dicho puerto es usado en Windows para compartir impresoras, archivos, entre otros. En una red empresarial se espera que se encuentre abierto.	2



## A.1. Bibliografía

1. Afianian, A., Niksefat, S., Sadeghiyan, B. y Baptiste, D. (2019). Malware Dynamic Analysis Evasion Techniques: A Survey. *ACM Computing Surveys*, 52(6), Artículo 126, 1-28. doi: 10.1145/3365001
2. Balazs, Z. (2016). Malware Analysis Sandbox Testing Methodology. *Le Journal de la Cybercriminalité & des Investigations Numériques*, 1(1). doi: 10.18464/cybin.v1i1.3
3. Besler, F., Willems, C. y Hund, R. (2018). Countering Innovative Sandbox Evasion Techniques Used By Malware. Recuperado el 10 de agosto de 2020 de <https://www.first.org/resources/papers/conf2017/Countering-Innovative-Sandbox-Evasion-Techniques-Used-by-Malware.pdf>
4. Bulazel, A. y Yener, B. (2017). A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion. *ROOTS: Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*, Artículo 2, 1-21. doi:10.1145/3150376.3150378
5. Cozzi, E., Graziano, M., Fratantonio, Y. y Balzarotti, D. (2018). Understanding Linux Malware. *S&P: Proceedings of the IEEE Symposium on Security and Privacy*. doi: 10.1109/SP.2018.00054
6. Gajrani, J., Sarswat, J., Tripathi, M., Laxmi, V., Gaur, M. S. y Conti, M. (2015). A Robust Dynamic Analysis System Preventing SandBox Detection by Android Malware. *SIN '15: Proceedings of the 8th International Conference on Security of Information and Networks*, 290-295. doi: 10.1145/2799979.2800004
7. Garfinkel, T., Adams, K., Warfield, A. y Franklin, J. (2007). Compatibility is Not Transparency: VMM Detection Myths and Realities. Recuperado el 11 de agosto de 2020 de [https://www.usenix.org/legacy/events/hotos07/tech/full\\_papers/garfinkel/garfinkel\\_html/index.html](https://www.usenix.org/legacy/events/hotos07/tech/full_papers/garfinkel/garfinkel_html/index.html)
8. Hellinger, D., Ming Xuan, L. y Gahlot, P. (2018). Dynamic Analysis of Evasive Malware with a Linux Container Sandbox. Recuperado el 11 de agosto de 2020 de [https://www.researchgate.net/publication/330500642\\_Dynamic\\_Analysis\\_of\\_Evasive\\_Malware\\_with\\_a\\_Linux\\_Container\\_Sandbox](https://www.researchgate.net/publication/330500642_Dynamic_Analysis_of_Evasive_Malware_with_a_Linux_Container_Sandbox)

9. IBM X-Force® Research. (2017). Evading the malware sandbox. Recuperado el 11 de agosto de 2020 de <https://safewayconsultoria.com/wp-content/uploads/2017/11/security-ibm-security-services-se-technical-white-paper-sew03166usen-20171009-2.pdf>
10. Kirat, D., Vigna, G. y Kruegel, C. (2014). BareCloud: Bare-metal Analysis-based Evasive Malware Detection. SEC'14: Proceedings of the 23rd USENIX conference on Security Symposium. Recuperado el 11 de agosto de 2020 de <https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-kirat.pdf>
11. Poettering, L. (2013). systemd for Administrators, Part XIX. Recuperado el 11 de agosto de 2020 de <http://0pointer.de/blog/projects/detect-virt.html>
12. Roccia, T., Rivero Lopez, M. y Shah, C. (2019). Evolution of Malware Sandbox Evasion Tactics - A Retrospective Study. Recuperado el 9 de agosto de 2020 de <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/evolution-of-malware-sandbox-evasion-tactics-a-retrospective-study/>

## Apéndice B - APIs

### B.1. Esquemas JSON

Aquí se describen los esquemas JSON comúnmente utilizados en el cuerpo de las peticiones y de las respuestas de las APIs tanto para nodos como para servidores de comando y control.

#### EnvironmentPlatformInformation

Representa al ambiente en el cual se ejecuta un nodo.

Clave	Tipo	Descripción	Requerido
<b>platform</b>	string	Resumen del sistema operativo y del hardware del entorno.	sí
<b>node</b>	string	Nombre del anfitrión.	sí
<b>os</b>	OSInfo	Información del sistema operativo.	sí
<b>hardware</b>	HardwareInfo	Información del procesador y su arquitectura.	sí
<b>python</b>	PythonInfo	Información del intérprete de Python.	sí

#### ExecutionSearchItem

Expone información acerca de una ejecución.

Clave	Tipo	Descripción	Requerido
<b>execution_id</b>	integer	Identificador unívoco de la ejecución.	sí
<b>session_id</b>	integer	Nombre del anfitrión.	sí
<b>timestamp_registered</b>	string (date-time)	Fecha y hora de registro de la ejecución en el sistema.	sí
<b>reports</b>	array de TestReport	Reportes de las pruebas llevadas a durante la ejecución en cuestión.	no

#### HardwareInfo

Esquema que describe al hardware en el cual se ejecuta un nodo.

Clave	Tipo	Descripción	Requerido
<b>machine</b>	string	Identificador del tipo de hardware. Ej: i386.	sí
<b>processor</b>	string	Identificador del procesador.	sí

## HTTPError

Esquema común de respuesta al producirse un error HTTP.

Clave	Tipo	Descripción	Requerido
error	string	Información acerca del error.	sí

## ActiveEnvironmentsItem

Aporta los datos mínimos de un entorno actualmente activo.

Clave	Tipo	Descripción	Requerido
session_id	integer	Identificador de la sesión correspondiente al entorno activo.	sí
session_start	string (date-time)	Fecha y hora de inicio de la sesión.	sí
ip	string	IP desde la cual el entorno escucha por solicitudes entrantes.	sí
port	integer	Puerto desde el cual el entorno escucha por solicitudes entrantes.	sí

## ModuleInfo

Representa el contenido de un módulo Python. Solo se destacan las clases derivadas de TestSet incluidas en el mismo.

Clave	Tipo	Descripción	Requerido
name	string	Nombre del módulo Python.	sí
test_sets	array de TestSetInfo	Lista de representaciones de cada clase extendida de TestSet contenida en el módulo.	no

## OSInfo

Esquema que describe al sistema operativo en el cual se ejecuta un nodo.

Clave	Tipo	Descripción	Requerido
system	string	Tipo de sistema. Ej: Linux, Windows.	sí
release	string	Número de lanzamiento.	sí
version	string	Versión en detalle del sistema.	sí

## PythonInfo

Esquema que describe al intérprete de Python mediante el cual se ejecuta un nodo.

Clave	Tipo	Descripción	Requerido
<b>build</b>	array de string	Arreglo de dos componentes: la primera describe la versión del intérprete y la segunda la fecha de compilación.	sí
<b>compiler</b>	string	Compilador utilizado para construir el intérprete.	sí
<b>implementation</b>	string	Denominación de la implementación particular del intérprete. Ej: CPython.	sí
<b>version</b>	string	Número de versión del lenguaje.	sí

## SessionInfo

Representa la información de una sesión. Sus ejecuciones asociadas son resumidas en un total.

Clave	Tipo	Descripción	Requerido
<b>session_id</b>	integer	Identificador de la sesión.	sí
<b>session_start</b>	string (date-time)	Fecha y hora de inicio de la sesión.	sí
<b>session_end</b>	string (date-time)	Fecha y hora de fin de la sesión.	no
<b>ip</b>	string	IP del nodo asociado a la sesión.	sí
<b>port</b>	integer	Puerto del nodo asociado a la sesión.	sí
<b>platform_info</b>	EnvironmentPlatformInformation	Información del sistema donde se alojó el nodo.	sí

## SessionSearchItem

Esquema de datos devuelto al realizar una búsqueda de un conjunto de sesiones.

Clave	Tipo	Descripción	Requerido
<b>session_id</b>	integer	Identificador de la sesión.	sí
<b>session_start</b>	string (date-time)	Fecha y hora de inicio de la sesión.	sí
<b>session_end</b>	string (date-time)	Fecha y hora de fin de la sesión.	no
<b>ip</b>	string	IP del nodo asociado a la sesión.	sí
<b>port</b>	integer	Puerto del nodo asociado a la sesión.	sí
<b>platform_os_system</b>	string	Sistema operativo del entorno de ejecución del nodo asociado a la sesión.	sí

## TestsPackageInfo

Representa el contenido de un paquete Python. Detalla sus módulos y sus subpaquetes.

Clave	Tipo	Descripción	Requerido
<b>name</b>	string	Nombre del paquete Python.	sí
<b>modules</b>	array de ModuleInfo	Lista de representaciones de cada módulo contenido en el paquete.	no
<b>subpackages</b>	array de TestsPackageInfo	Lista de representaciones de los subpaquetes contenidos.	no

## TestReport

Esquema común para representar los resultados de un test.

Clave	Tipo	Descripción	Requerido
<b>test_name</b>	string	Nombre de la prueba.	sí
<b>tests_description</b>	string	Información acerca de la prueba.	sí
<b>timestamp_start</b>	string (date-time)	Fecha y hora de inicio de la prueba.	sí
<b>timestamp_end</b>	string (date-time)	Fecha y hora de fin de la prueba.	sí
<b>result_code</b>	integer	< 0: fracaso 0: advertencia > 0: éxito	sí
<b>additional_info</b>	diccionario	Conjunto de claves string y valores de tipo variado que proporcionan más detalles sobre el resultado de la prueba.	no

## TestSetInfo

Representa el contenido de una clase extendida de TestSet.

Clave	Tipo	Descripción	Requerido
<b>name</b>	string	Nombre de la clase.	sí
<b>tests</b>	array de string	Lista de nombres de los métodos de test contenidos en la clase.	no

## B.2. Endpoints de un nodo

### B.2.1. DELETE /

Detiene la ejecución del nodo.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación.

#### Respuestas

Código	Descripción	Tipo de contenido	Cuerpo
204	Petición recibida correctamente	application/json	-
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError

### B.2.2. GET /reports

Ejecuta las pruebas instaladas en un nodo y devuelve los reportes correspondientes. Si están presentes los parámetros packages, modules o test\_sets, solo las pruebas contenidas en las entidades determinadas son ejecutadas.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
packages	array de string	consulta	no	Los nombres canónicos de los paquetes a ser ejecutados enteramente, separados por comas.
modules	array de string	consulta	no	Los nombres canónicos de los módulos a ser ejecutados enteramente, separados por comas.
test_sets	array de string	consulta	no	Los nombres canónicos de los conjuntos de prueba a ser ejecutados enteramente, separados por comas.
tests	array de string	consulta	no	Los nombres canónicos de los tests específicos a ser ejecutados, separados por comas.

## Respuestas

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	array de TestReport
400	Parámetros de consulta incorrectos.	application/json	HTTPError
404	Alguna de las entidades especificadas no fue hallada.	application/json	HTTPError

### B.2.3. GET /test\_sets

Devuelve una lista en la que cada componente es una representación recursiva de cada uno de los paquetes instalados en el nodo.

## Respuestas

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	array de TestPackageInfo

### B.2.4. PATCH /test\_sets

Sube nuevos paquetes o actualiza los ya presentes desde un archivo comprimido.

## Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación. Debe incluir "digest" como cabecera.
Digest	string	cabecera	sí	Resumen del cuerpo de la petición, calculado mediante la función SHA-256 y codificado en base 64.



## Cuerpo de la petición

- **Tipo de contenido:** multipart/form-data
- **Contenido:**
  - **Tipo:** diccionario
  - **Claves:**
    - **packages:**
      - **Tipo:** string (formato binario)
      - **Descripción:** Archivo en formato tar.gz

## Respuestas

Código	Descripción	Tipo de contenido	Cuerpo
204	El archivo fue correctamente descomprimido y los paquetes correspondientes han sido agregados o actualizados acordemente.	-	-
400	Contenido de la petición inválido.	application/json	HTTPError
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
415	Tipo del contenido de la petición no soportado.	application/json	HTTPError

### B.2.5. DELETE /test\_sets/{package}

Elimina el paquete dado (incluido todo su contenido) del nodo.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
package	string	ruta	sí	El nombre del paquete a eliminar.
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación.

## Respuestas

Código	Descripción	Tipo de contenido	Cuerpo
204	El paquete fue removido exitosamente.	-	-
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
404	El paquete no pudo ser hallado.	application/json	HTTPError

## B.3. Endpoints de un servidor de comando y control

### B.3.1. GET /environments

Recupera un diccionario con información respecto a los entornos actualmente activos registrados en el servidor de comando y control.

#### Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	array de ActiveEnvironmentsItem

### B.3.2. POST /environments

Permite a un nodo registrarse en el servidor.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
<b>Authorization</b>	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación. Debe incluir "digest" como cabecera.
<b>Digest</b>	string	cabecera	sí	Resumen del cuerpo de la petición, calculado mediante la función SHA-256 y codificado en base 64.

## Cuerpo de la petición

- **Tipo de contenido:** application/json
- **Contenido:**
  - **Tipo:** diccionario
  - **Claves:**
    - **ip:**
      - **Tipo:** string
      - **Descripción:** ip desde donde el nodo se encontrará escuchando peticiones.
    - **port:**
      - **Tipo:** string
      - **Descripción:** puerto desde donde el nodo se encontrará escuchando peticiones.
    - **platform\_info:**
      - **Tipo:** EnvironmentPlatformInfo
      - **Descripción:** La información del sistema desde donde se comunica el nodo solicitante.

## Respuestas

Código	Descripción	Tipo de contenido	Cuerpo
204	El entorno fue registrado exitosamente.	-	-
400	Contenido de la petición inválido.	application/json	HTTPError
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
415	Tipo del contenido de la petición no soportado.	application/json	HTTPError

### B.3.3. DELETE /environments/{ip}/{port}

Permite a un nodo darse de baja del servidor.

## Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
ip	string	ruta	sí	La dirección IP del entorno registrado.
port	string	ruta	sí	El puerto del entorno registrado.
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación.

## Respuestas

Código	Descripción	Tipo de contenido	Cuerpo
204	El entorno fue removido exitosamente.	-	-
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
404	No se encontró un entorno registrado con la dirección IP y el puerto solicitados.	application/json	HTTPError

### B.3.4. GET /environments/{ip}/{port}/info

Recupera la información acerca de la plataforma del entorno en la dirección y puerto dados.el

## Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
ip	string	ruta	sí	La dirección IP del entorno registrado.
port	string	ruta	sí	El puerto del entorno registrado.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	EnvironmentPlatformInformation
404	No se encontró un entorno registrado con la dirección IP y el puerto solicitados.	application/json	HTTPError

### B.3.5. GET /environments/{ip}/{port}/installed

Lista todos los paquetes raíz instalados en el entorno en la dirección y puerto dados.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
ip	string	ruta	sí	La dirección IP del entorno registrado.
port	string	ruta	sí	El puerto del entorno registrado.

#### Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	TestsPackageInfo
404	No se encontró un entorno registrado con la dirección IP y el puerto solicitados.	application/json	HTTPError

### B.3.6. PATCH /environments/{ip}/{port}/installed

Sube nuevos paquetes o actualiza los ya existentes en el entorno en la dirección y puerto dados.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
ip	string	ruta	sí	La dirección IP del entorno registrado.
port	string	ruta	sí	El puerto del entorno registrado.
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación. Debe incluir "digest" como cabecera.
Digest	string	cabecera	sí	Resumen del cuerpo de la petición, calculado mediante la función SHA-256 y codificado en base 64.

## Cuerpo de la petición

- **Tipo de contenido:** application/json
- **Contenido:**
  - **Tipo:** array de string
  - **Descripción:** listado de nombres de paquetes raíz a trasladar del servidor de comando y control al nodo correspondiente.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
204	Los paquetes fueron transferidos o actualizados correctamente.	-	-
400	Se encontró un paquete inválido dentro del listado enviado.	application/json	HTTPError
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
404	No se encontró un entorno registrado con la dirección IP y el puerto solicitados.	application/json	HTTPError
415	Tipo del contenido de la petición no soportado.	application/json	HTTPError
500	Ocurrió algo inesperado al gestionar la petición.	application/json	HTTPError
502	Se recibió una respuesta inesperada desde el nodo solicitado.	application/json	HTTPError
504	El entorno solicitado no pudo ser contactado.	application/json	HTTPError

### B.3.7. DELETE /environments/{ip}/{port}/installed/{package}

Elimina el paquete especificado del entorno en la dirección y puerto dados.el

## Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
ip	string	ruta	sí	La dirección IP del entorno registrado.
port	string	ruta	sí	El puerto del entorno registrado.
package	string	ruta	sí	El nombre del paquete raíz a eliminar.
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
204	El paquete fue desinstalado exitosamente.	-	-
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
404	No se encontró el recurso solicitado.	application/json	HTTPError
502	Se recibió una respuesta inesperada desde el nodo solicitado.	application/json	HTTPError
504	El entorno solicitado no pudo ser contactado.	application/json	HTTPError

### B.3.8. GET /environments/{ip}/{port}/reports

Ejecuta las pruebas especificadas en el entorno dado y recupera los reportes correspondientes.

## Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
<b>ip</b>	string	ruta	sí	La dirección IP del entorno registrado.
<b>port</b>	string	ruta	sí	El puerto del entorno registrado.
<b>packages</b>	array de string	consulta	no	Los nombres canónicos de los paquetes a ser ejecutados enteramente, separados por comas.
<b>modules</b>	array de string	consulta	no	Los nombres canónicos de los módulos a ser ejecutados enteramente, separados por comas.
<b>test_sets</b>	array de string	consulta	no	Los nombres canónicos de los conjuntos de prueba a ser ejecutados enteramente, separados por comas.
<b>tests</b>	array de string	consulta	no	Los nombres canónicos de los tests específicos a ser ejecutados, separados por comas.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
<b>200</b>	OK.	application/json	TestReport
<b>400</b>	Solicitud inválida.	application/json	HTTPError
<b>404</b>	No se encontró un entorno registrado con la dirección IP y el puerto solicitados o una de las entidades especificadas no fue hallada en el nodo.	application/json	HTTPError
<b>500</b>	Ocurrió algo inesperado al gestionar la petición.	application/json	HTTPError
<b>502</b>	Se recibió una respuesta inesperada desde el nodo solicitado.	application/json	HTTPError
<b>504</b>	El entorno solicitado no pudo ser contactado.	application/json	HTTPError

### B.3.9. GET /executions

Busca ejecuciones en base al criterio brindado.



## Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
<b>ids</b>	array de integer	query	no	Una lista de identificadores de ejecuciones por los cuales filtrar la búsqueda. Entre cada identificador se debe ubicar una coma para delimitarlos.
<b>sessions</b>	array de integer	query	no	Indica a qué sesiones deben corresponder las ejecuciones a seleccionar. Para indicar más de una sesión, las mismas deben separarse mediante comas.
<b>registered_from</b>	string (date-time)	query	no	Filtra ejecuciones mediante la especificación de la fecha de registro más antigua admitida.
<b>registered_to</b>	string (date-time)	query	no	Filtra ejecuciones mediante la especificación de la fecha de registro más reciente admitida.
<b>order_by</b>	string	query	no	Ordena los datos por el campo indicado. Los valores aceptados son: <ul style="list-style-type: none"> <li>• <b>id</b>: Ordena por el identificador de la ejecución.</li> <li>• <b>session</b>: Ordena por el identificador de la sesión.</li> <li>• <b>registered</b>: Ordena por fecha de registro.</li> </ul>
<b>arrange</b>	string	query	no	Especifica si el orden de los datos es ascendente ( <b>asc</b> ) o descendente ( <b>desc</b> ). Solo puede usarse este parámetro en conjunto con "order_by". Si dicho parámetro está presente y "arrange" ha sido omitido, el orden tomado por defecto es ascendente.
<b>limit</b>	integer min: 1	query	no	Indica la cantidad máxima de instancias del recurso solicitado que puede devolver la búsqueda.
<b>offset</b>	integer min: 0	query	no	Se usa en conjunto con "limit" para especificar cual debería ser el comienzo del conjunto de resultados. Solo puede usarse con "limit". Si dicho parámetro está presente y "offset" ha sido omitido, el valor por defecto de este es 0.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	Array de ExecutionSearchItem
400	Error encontrado en un parámetro.	application/json	HTTPError

### B.3.10. DELETE /executions/{execution\_id}

Elimina la ejecución especificada y todos sus reportes asociados.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
execution_id	integer	ruta	sí	El identificador de la ejecución almacenada a eliminar.
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
204	La ejecución fue eliminada con éxito.	-	-
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
404	No se encontró la ejecución solicitada.	application/json	HTTPError

### B.3.11. GET /sessions

Busca ejecuciones en base al criterio brindado.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
ids	array de integer	query	no	Una lista de identificadores de sesiones por los cuales filtrar la búsqueda. Entre cada identificador se debe ubicar una coma para delimitarlos.

Nombre	Tipo	Ubicación	Requerido	Descripción
<b>start_from</b>	string (date-time)	query	no	Filtra sesiones mediante la especificación de la fecha de inicio más antigua admitida.
<b>start_to</b>	string (date-time)	query	no	Filtra sesiones mediante la especificación de la fecha de inicio más reciente admitida.
<b>end_from</b>	string (date-time)	query	no	Filtra sesiones mediante la especificación de la fecha de finalización más antigua admitida.
<b>end_to</b>	string (date-time)	query	no	Filtra sesiones mediante la especificación de la fecha de finalización más reciente admitida.
<b>ips</b>	array de string	query	no	Una lista de direcciones IP a las que deberían estar asociadas las sesiones a consultar. Una coma es usada como delimitador de valores.
<b>ports</b>	array de integer	query	no	Una lista de puertos a los que deberían estar asociadas las sesiones a consultar. Una coma es usada como delimitador de valores.
<b>order_by</b>	string	query	no	Ordena los datos por el campo indicado. Los valores aceptados son: <ul style="list-style-type: none"> <li>• <b>id</b>: Ordena por el identificador de la sesión.</li> <li>• <b>start</b>: Ordena por la fecha de inicio de la sesión.</li> <li>• <b>end</b>: Ordena por la fecha de finalización de la sesión.</li> <li>• <b>ip</b>: Ordena por dirección IP asociada a la respectiva sesión.</li> <li>• <b>port</b>: Ordena por puerto asociado a la respectiva sesión.</li> <li>• <b>system</b>: Ordena por sistema operativo correspondiente a la respectiva sesión.</li> </ul>
<b>arrange</b>	string	query	no	Especifica si el orden de los datos es ascendente ( <b>asc</b> ) o descendente ( <b>desc</b> ). Solo puede usarse este parámetro en conjunto con <b>order_by</b> . Si dicho parámetro está presente y este ha sido omitido, el orden tomado por defecto es ascendente.
<b>limit</b>	integer min: 1	query	no	Indica la cantidad máxima de instancias del recurso solicitado que puede devolver la búsqueda.
<b>offset</b>	integer min: 0	query	no	Se usa en conjunto con "limit" para especificar cual debería ser el comienzo del conjunto de resultados. Solo puede usarse con "limit". Si dicho parámetro está presente y "offset" ha sido omitido, el valor por defecto de este es 0.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	Array de SessionSearchItem
400	Error encontrado en un parámetro.	application/json	HTTPError

### B.3.12. GET /sessions/{session\_id}

Recupera la información de la sesión especificada.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
session_id	integer	ruta	sí	El identificador de la sesión almacenada a consultar.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	SessionInfo
404	No se encontró la sesión solicitada.	application/json	HTTPError

### B.3.13. DELETE /sessions/{session\_id}

Elimina la sesión finalizada especificada y todas sus ejecuciones y reportes asociados.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
session_id	integer	ruta	sí	El identificador de la sesión almacenada a eliminar.
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
204	La sesión fue eliminada con éxito.	-	-
400	La sesión solicitada aún se encuentra activa.	application/json	HTTPError
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
404	No se encontró la sesión solicitada.	application/json	HTTPError

### B.3.14. GET /test\_sets

Devuelve información acerca de todos los paquetes raíz contenidos en el repositorio del servidor de comando y control.

## Respuestas

Código	Descripción	Tipo de contenido	Cuerpo
200	OK.	application/json	Array de TestsPackageInfo

### B.3.15. PATCH /test\_sets

Sube nuevos paquetes al repositorio o actualiza los ya existentes.

## Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación. Debe incluir "digest" como cabecera.
Digest	string	cabecera	sí	Resumen del cuerpo de la petición, calculado mediante la función SHA-256 y codificado en base 64.

## Cuerpo de la petición

- **Tipo de contenido:** multipart/form-data
- **Contenido:**
  - **Tipo:** diccionario
  - **Claves:**
    - **packages:**
      - **Tipo:** string (formato binario)
      - **Descripción:** Archivo en formato tar.gz

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
204	El archivo fue correctamente descomprimido y los paquetes correspondientes han sido agregados o actualizados acordemente.	-	-
400	Contenido de la petición inválido.	application/json	HTTPError
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
415	Tipo del contenido de la petición no soportado.	application/json	HTTPError

### B.3.16. DELETE /test\_sets/{package}

Elimina el paquete raíz dado (junto a todo su contenido) del repositorio.

#### Parámetros

Nombre	Tipo	Ubicación	Requerido	Descripción
package	string	ruta	sí	El nombre del paquete raíz a eliminar.
Authorization	SecchiwareAuth	cabecera	sí	Mecanismo de autenticación.

## Respuestas

Todas las respuestas incluyen el parámetro *Access-Control-Allow-Origin* en su cabecera.

Código	Descripción	Tipo de contenido	Cuerpo
204	El paquete fue desinstalado exitosamente.	-	-
401	No autorizado. Incluye la cabecera "WWW-Authenticate".	application/json	HTTPError
404	No se encontró el paquete solicitado.	application/json	HTTPError

# Apéndice C - Comandos del cliente CLI

## C.1. Argumentos comunes

- ip: dirección IP de un nodo particular.
- port: puerto asociado a un nodo particular.

## C.2. Opciones comunes

- --c2-url, -u: URL del servidor de comando y control al cual se busca contactar. Si se omite toma por defecto el valor "http://127.0.0.1:5000".
- --password: contraseña de cliente acordada con el servidor de comando y control. De omitirse al momento de invocar el comando, lo primero que realiza la componente es preguntársela al usuario mediante un aviso acompañado de una entrada de texto oculto.

## C.3. Comandos

- **Comando:** lsavailable
  - **Argumentos:** -
  - **Opciones:**
    - --c2-url, -u
  - **Descripción:** Muestra todos los paquetes disponibles en el repositorio de pruebas del servidor C&C.
- **Comando:** upload
  - **Argumentos:**
    - file\_path: la ruta del archivo a subir.
  - **Opciones:**
    - --c2-url, -u
    - --password
  - **Descripción:** Permite subir paquetes de tests en un archivo comprimido en formato tar.gz al servidor C&C.
- **Comando:** remove
  - **Argumentos:**
    - packages: un listado de nombres de paquetes a eliminar.



- **Opciones:**
  - --c2-url, -u
  - --password
- **Descripción:** Elimina los paquetes raíz especificados del servidor central.
- **Comando:** lsend
  - **Argumentos:** -
  - **Opciones:**
    - --c2-url, -u
  - **Descripción:** Lista los nodos actualmente asociados al servidor C&C.
- **Comando:** sessions\_search
  - **Argumentos:** -
  - **Opciones:**
    - --c2-url, -u
    - --session\_id: filtra por identificador. Soporta múltiples valores.
    - --start\_from: filtra por fecha mínima de comienzo permitida.
    - --start\_to: filtra por fecha máxima de comienzo permitida.
    - --end\_from: filtra por fecha mínima de finalización permitida.
    - --end\_to: filtra por fecha máxima de finalización permitida.
    - --ip: filtra por IP. Soporta múltiples valores.
    - --port: filtra por puerto. Soporta múltiples valores.
    - --system: filtra por sistema operativo. Soporta múltiples valores.
    - --order\_by: especifica el parámetro por el cual ordenar los resultados. Los valores aceptados son "id", "start", "end", "ip", "port", "system".
    - --arrange: determina si la secuencia ordenada debe ser ascendente ("asc") o descendente ("desc").
    - --limit: establece cantidad máxima de resultados permitida.
    - --offset: al usarse en conjunto con *limit* indica cuál es el primer registro del conjunto de resultados que se debe tomar.
  - **Descripción:** Consulta la información de las sesiones que cumplen con el criterio dado.
- **Comando:** session\_get

- **Argumentos:**
  - session: el identificador de la sesión deseada.
- **Opciones:**
  - --c2-url, -u
- **Descripción:** Recupera la información acerca de la sesión indicada.
- **Comando:** sessions\_delete
  - **Argumentos:**
    - sessions: el listado de sesiones a eliminar.
  - **Opciones:**
    - --c2-url, -u
    - --password
  - **Descripción:** Elimina todas las sesiones indicadas, como así sus ejecuciones y reportes asociados.
- **Comando:** executions\_search
  - **Argumentos:** -
  - **Opciones:**
    - --c2-url, -u
    - --execution\_id: filtra por identificador de ejecución. Soporta múltiples valores.
    - --session: filtra por sesión a la que está asociada la ejecución. Soporta múltiples valores.
    - --registered\_from: filtra por fecha mínima de registro aceptada.
    - --registered\_to: filtra por fecha máxima de registro aceptada.
    - --order\_by: especifica el parámetro por el cual ordenar los resultados. Los valores aceptados son "id", "session", "registered".
    - --arrange: determina si la secuencia ordenada debe ser ascendente ("asc") o descendente ("desc").
    - --limit: establece cantidad máxima de resultados permitida.
    - --offset: al usarse en conjunto con *limit* indica cuál es el primer registro del conjunto de resultados que se debe tomar.
  - **Descripción:** Consulta la información de las ejecuciones (y en consecuencia sus reportes asociados) que cumplan con el criterio especificado.

- **Comando:** executions\_delete
  - **Argumentos:**
    - executions: el listado de ejecuciones a eliminar.
  - **Opciones:**
    - --c2-url, -u
    - --password
  - **Descripción:** Elimina todas las ejecuciones indicadas y reportes asociados.
- **Comando:** info
  - **Argumentos:**
    - ip
    - port
  - **Opciones:**
    - --c2-url, -u
  - **Descripción:** Consulta la información acerca de la plataforma sobre la que se está ejecutando el nodo en la dirección dada.
- **Comando:** lsinstalled
  - **Argumentos:**
    - ip
    - port
  - **Opciones:**
    - --c2-url, -u
  - **Descripción:** Muestra los tests instalados actualmente en el entorno en la dirección dada.
- **Comando:** install
  - **Argumentos:**
    - ip
    - port
    - packages: listado de paquetes a instalar.
  - **Opciones:**
    - --c2-url, -u

- --password
- **Descripción:** Copia los paquetes especificados desde el servidor central al nodo en la dirección dada.
- **Comando:** uninstall
  - **Argumentos:**
    - ip
    - port
    - packages: listado de paquetes a desinstalar.
  - **Opciones:**
    - --c2-url, -u
    - --password
  - **Descripción:** Elimina los paquetes especificados del entorno en la dirección dada.
- **Comando:** reports\_get
  - **Argumentos:**
    - ip
    - port
  - **Opciones:**
    - --c2-url, -u
    - --package: nombre canónico de un paquete particular a ejecutar en su totalidad. Soporta múltiples valores.
    - --module: nombre canónico de un módulo particular a ejecutar en su totalidad. Soporta múltiples valores.
    - --test\_set: nombre canónico de un conjunto de pruebas a ejecutar en su totalidad. Soporta múltiples valores.
    - --test: nombre canónico de una prueba a ejecutar. Soporta múltiples valores.
  - **Descripción:** Ejecuta y recupera los reportes de las pruebas solicitadas en el nodo en la dirección indicada.

## Apéndice D - Verificación de transparencia de entornos selectos

### D.1. Entornos

Entorno	Virtualización	Procesador y su arquitectura	Memoria	Sistema Operativo	Python	Detalle
E1	No	Qualcomm Snapdragon 625 - ARMv8-A 64 bits	2 GB	Android 8.1.0, kernel 3.18.71-perf-g7b625a0	3.6.6 (QPython)	Smartphone personal del autor.
E2	Android Emulator (QEMU-KVM)	Android Virtual Processor - x86	2 GB	Android 10, kernel 5.4.43-00621-g90087296b3b1-i686-with-libc	3.6.6 (QPython)	Google Pixel emulado sobre E3. Creado mediante Android Virtual Device Manager.
E3	No	AMD Ryzen 2500U - x86_64	8 GB	Ubuntu 18.04.5, kernel 5.4.0-42-generic	3.6.9	Equipo personal del autor. Nodo iniciado como super-usuario.
E4	VirtualBox	AMD Ryzen 2500U (un núcleo monohilo) - x86_64	483 MB	Debian 4.19.118-2, kernel 4.19.0-9-amd64-x86_64-with-debian-10.4	3.7.3	Virtualizado sobre el entorno E3.
E5	Docker	AMD Ryzen 2500U - x86_64	8 GB	Ubuntu 18.04.5, kernel 5.4.0-42-generic	3.8.3	Virtualizado sobre el entorno E3.
E6	No	AMD Ryzen 2500U - x86_64	8 GB	Windows 10 2004	3.8.5	Equipo personal del autor. Nodo iniciado como administrador.
E7	VirtualBox	AMD Ryzen 2500U (dos núcleos monohilo) - x86_64	3 GB	Windows 10 2004	3.8.5	Virtualizado sobre el entorno E3. Máquina virtual obtenida de <a href="https://developer.microsoft.com/es-es/windows/downloads/virtual-machines/">https://developer.microsoft.com/es-es/windows/downloads/virtual-machines/</a>

## D.2. Resultados de las pruebas

Prueba	T	E1	E2	E3	E4	E5	E6	E7
android.qpython3_tests.QPython3EmulatorSet.imei_from_emulator	10	P	P	NA	NA	NA	NA	NA
android.adb_tests.ADBEmulatorSet.brand_from_emulator	12	P	P	NA	NA	NA	NA	NA
android.adb_tests.ADBEmulatorSet.manufacturer_from_emulator	16	P	P	NA	NA	NA	NA	NA
android.adb_tests.ADBEmulatorSet.has_bluetooth	21	P	P	NA	NA	NA	NA	NA
android.qpython3_tests.QPython3CommunicationSet.contacts_registered	25	P	F	NA	NA	NA	NA	NA
android.qpython3_tests.QPython3CommunicationSet.sms_stored	27	P	F	NA	NA	NA	NA	NA
android.qpython3_tests.QPython3CommunicationSet.detect_call	29	F	P	NA	NA	NA	NA	NA
android.adb_tests.ADBEmulatorSet.network_operator_name_from_emulator	32	P	F	NA	NA	NA	NA	NA
linux.virtualization.container_tests.ContainerSet.is_first_process_an_init	38	I	I	P	P	F	NA	NA
linux.virtualization.container_tests.DockerSet.dockerenv_present	39	P	P	P	P	F	NA	NA
linux.virtualization.container_tests.ContainerSet.read_kallsyms	43	I	I	P	I	F	NA	NA
linux.general_tests.MonitoringSet.is_tracer_attached	44	P	P	F	P	P	NA	NA
linux.general_tests.HooksAndInjectedLibrariesSet.id_preload_present	46	P	P	F	P	P	NA	NA
linux.virtualization.hosted_hypervisor_tests.X86VirtualizationSet.product_name	50	I	I	P	F	P	NA	NA
linux.virtualization.systemd_tests.SystemdSet.systemd_detect_virt_test	55	I	I	P	F	I	NA	NA
os_agnostic.agnostic_tests.AgnosticNetworkingSet.are_fake_domains_resolved	65	P	P	P	P	P	P	P
linux.general_tests.NetworkingSet.ssids_present	70	I	I	P	I	I	NA	NA
x86.x86_tests.X86VirtualizationSet.cpuid_hypervisor_bit_on	79	NA	NA	P	F	P	P	F
os_agnostic.agnostic_tests.AgnosticAntiAnalysisSet.sleep_emulated	81	P	P	P	P	P	P	P
android.qpython3_tests.QPython3HumanUseSet.whatsapp_installed	85	P	F	NA	NA	NA	NA	NA

Prueba	T	E1	E2	E3	E4	E5	E6	E7
windows.windows_api.windows_api.WindowsAPIHumanUseSet.windows_api_huma n_like_cursor_speed	92	NA	NA	NA	NA	NA	F	P
android.qpython3_tests.QPython3HumanUseSet.clipboard_has_content	98	F	P	NA	NA	NA	NA	NA
linux.general_tests.MonitoringSet.wireshark_running	100	P	P	F	P	P	NA	NA
linux.virtualization.container_tests.DockerSet.docker_like_ip	101	P	P	F	P	F	NA	NA
windows.wmic_tests.WMICHardwareRecognitionSet.can_the_temperature_be_reco vered	105	NA	NA	NA	NA	NA	P	I
windows.wmic_tests.WMICHardwareRecognitionSet.disk_with_minimum_size	107	NA	NA	NA	NA	NA	P	P
linux.general_tests.HardwareSet.at_least_two_cores	116	I	P	P	F	P	NA	NA
windows.wmic_tests.WMICHardwareRecognitionSet.at_least_two_cores	116	NA	NA	NA	NA	NA	P	P
os_agnostic.agnostic_tests.AgnosticAntiAnalysisSet.timing_test	118	I	I	I	I	I	I	I
windows.windows_api.windows_api.WindowsAPIVirtualizationSet.windows_api_virt ual_device_drivers_exist	121	NA	NA	NA	NA	NA	P	P
windows.windows_api.windows_api.WindowsAPIDebuggerSet.windows_api_is_deb ugger_present	122	NA	NA	NA	NA	NA	F	P
windows.x86.x86_tests.DebuggerSet.int2dh	123	NA	NA	NA	NA	NA	P	P
windows.windows_api.windows_api.WindowsAPIHumanUseSet.windows_api_word wheelquery_has_content	127	NA	NA	NA	NA	NA	F	F
windows.windows_api.windows_api.WindowsAPIHumanUseSet.windows_api_foreg round_window_changes	129	NA	NA	NA	NA	NA	P	F

**Aclaraciones:**

- La columna “Prueba” contiene el nombre canónico del test implementado en Python.
- La columna “T” referencia al número de técnica correspondiente al compendio del Apéndice A.
- Resultados posibles:
  - NA: La prueba no se aplica al entorno dado.
  - P: El entorno pasó la prueba y por ende es transparente frente a la técnica de detección aplicada.
  - F: El entorno falló la prueba.
  - I: Se produjo una indeterminación ya sea o porque no se pudo ejecutar el test adecuadamente o porque no se obtuvieron resultados concluyentes.





## E.2. Gantt de la evolución real del proyecto

Nro	Tarea	Cantidad de horas	Diciembre				Enero				Febrero				Marzo				Abril				Mayo				Junio				Julio				Agosto			
			1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Investigación de técnicas para detectar entornos de análisis	50	■	■	■	■																																
2	Definir el marco teórico	50	■	■	■	■																																
3	Definición de requerimientos funcionales y no funcionales	10					■	■																														
4	Diseño de la arquitectura general del sistema	60					■	■	■	■	■	■	■	■																								
5	Especificación de la integración de nuevos módulos	10						■	■																													
6	Estudio y selección de tecnologías	17						■	■	■																												
7	Definición de APIs REST	15										■	■	■																								
8	Desarrollo del verificador de transparencia	200									■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■								
9	Selección de vulnerabilidades representativas	14															■	■																				
10	Implementación modular de tests acordes	20															■	■																				
11	Preparación y prueba de entornos representativos	7																														■	■					
12	Documentación del desarrollo del producto	60																														■	■	■				
13	Redacción de conclusiones	5																															■					
14	Revisión del informe	20																														■	■	■				
	<b>Total horas</b>	<b>538</b>																																				