UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Generación de artefactos forenses en un cyber range

Informe de Proyecto de Grado presentado por

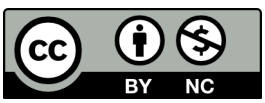
Rodrigo Aguillón, Ignacio Alesina

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Gustavo Betarte
Juan Diego Campo
Marcelo Rodríguez

Montevideo, 27 de abril de 2026



Agradecimientos

En primer lugar, queremos expresar nuestro más sincero agradecimiento a nuestros tutores, Juan Diego Campo y Marcelo Rodríguez, por su guía, dedicación y paciencia durante todo el desarrollo de este proyecto. Su experiencia y disposición fueron fundamentales para orientar nuestras decisiones técnicas y mantener el rumbo del trabajo para que este resultara de manera satisfactoria.

Rodrigo quiere agradecer especialmente a sus padres, Miguel Aguillón y Laris Stoletniy, cuyo apoyo incondicional hizo posible llegar hasta aquí. A su abuela Renée Romero, por estar presente en cada logro y por el orgullo genuino que siempre supo transmitir. Y a la memoria de su abuelo Miguel Aguillón, cuya presencia sigue siendo parte de cada paso dado. Este trabajo también es de ustedes.

Ignacio quiere agradecer a sus padres, Alejandro Alesina y Helena Acuña, y a su hermana Clara Alesina, por haber sido un sostén constante desde casa, brindándole todo lo necesario para poder estudiar y estando siempre presentes con una sonrisa incluso en los días más difíciles. A su abuela Miria Carbajal, por su continuo apoyo, por estar siempre que la precisó y por los innumerables consejos y aprendizajes que le brindó. A su abuelo Federico Alesina, quien hoy no se encuentra presente, pero que siempre soñó con él este momento, y a Gladys por acompañar tan de cerca todo su proceso. Por último, a su novia Eugenia, quien lo acompaña y lo ayuda a volver a tierra cuando no logra ver la solución. No habría alcanzado esta meta de no ser por ellos.

Finalmente, agradecemos a los amigos y allegados que estuvieron en el camino, los que aguantaron las ausencias, los que preguntaron cómo iba aunque no entendieran la respuesta, y los que estuvieron para festejar cuando correspondía. Su presencia hizo que el proceso fuera más llevadero y los logros más significativos.

Resumen

La formación práctica en Análisis Forense Digital y Respuesta a Incidentes (DFIR) requiere entornos donde los estudiantes puedan analizar evidencia digital en condiciones realistas. Tectonic, el cyber range desarrollado por el Grupo de Seguridad Informática del Instituto de Computación, automatiza el despliegue de laboratorios de ciberseguridad mediante Infraestructura como Código, pero no cuenta con mecanismos para generar los artefactos forenses que dichos escenarios necesitan. La creación manual de artefactos realistas genera una alta carga operativa para los instructores y dificulta la incorporación de actividad de fondo coherente que permita a los estudiantes distinguir artefactos relevantes del ruido circundante, replicando las condiciones reales de una investigación forense.

Este proyecto amplía las capacidades de Tectonic para soportar escenarios de DFIR. Se realizó una investigación del estado del arte en las distintas vertientes del análisis forense y en los enfoques de generación de artefactos sintéticos en cyber ranges, de la cual se derivaron los requerimientos de la solución.

Para satisfacer estos requerimientos se diseñó una arquitectura en cuatro niveles de abstracción. Una capa transversal (Capa 0) que provee datos sintéticos pseudoaleatorios que dotan de realismo a los artefactos. Sobre esta base, la Capa 1 define primitivas atómicas, la Capa 2 las combina en flujos de actividad coherentes, y la Capa 3 encapsula perfiles de comportamiento para desplegar escenarios completos mediante una única declaración de alto nivel. La implementación, integrada al ecosistema de Tectonic mediante un lenguaje de especificación, cubrió las vertientes de red y sistema de archivos.

La solución fue validada mediante un caso de estudio, generando una captura de tráfico de red y una imagen de disco. Los artefactos resultantes fueron analizados exitosamente con herramientas estándar de la industria, confirmando que los artefactos generados son técnicamente correctos, reproducibles y analizables en condiciones reales de práctica forense.

Palabras clave: Cyber Range, DFIR, Generación Sintética de Artefactos, Sistema de Archivos, Red, Infraestructura como Código, Reproducibilidad.

Índice general

Capítulo 1

Introducción

La digitalización masiva de las actividades cotidianas —operaciones bancarias, atención hospitalaria, servicios gubernamentales— hizo que el campo de la seguridad asuma un rol de creciente importancia en la sociedad: proteger y preservar los datos almacenados en estos sistemas. Atender ese desafío requiere profesionales que hayan estudiado, comprendido y puedan dedicarse a la disciplina para resguardar los activos digitales expuestos en la red. El World Economic Forum estima que para 2030 podría existir una escasez global de talento —en todos los sectores— superior a 85 millones de trabajadores, con un costo potencial de $8.5 trillones anuales en ingresos perdidos. En el campo específico de la ciberseguridad, ese déficit alcanza cerca de 4 millones de profesionales (World Economic Forum, 2024). El impacto es concreto: cerca del 90 % de los líderes organizacionales declararon haber experimentado al menos una brecha de seguridad que pueden atribuir parcialmente a la falta de habilidades en ciberseguridad en sus equipos (Fortinet, 2024). La industria es clara al respecto: la experiencia *hands-on* previa es el principal criterio al evaluar la idoneidad de un candidato, citado por el 73 % de los empleadores, muy por encima de los títulos obtenidos (38 %) (ISACA, 2024), lo que hace insuficiente cualquier enfoque de formación exclusivamente teórico.

Para trabajar en la práctica con DFIR, se requieren desarrollar habilidades como: reconstruir secuencias de eventos maliciosos, determinar el alcance de un compromiso y preservar evidencia con validez legal, que exige integrar conocimientos de sistemas de archivos, memoria volátil y tráfico de red (Johansen, 2020), bajo procedimientos rigurosos de cadena de custodia (Kent, Chevalier, Grance, y Dang, 2006). Estas competencias solo se consolidan mediante práctica repetida en entornos controlados: trabajar sobre evidencia de incidentes reales resulta éticamente inaceptable y operacionalmente riesgoso, ya que cualquier error puede comprometer investigaciones en curso o generar pérdida irreversible de datos (Johansen, 2020).

Los cyber ranges ofrecen una alternativa: entornos virtuales aislados que replican infraestructuras reales y permiten entrenar sin consecuencias en producción (Pham, Tang, Chinen, y Beuran, 2016). Organizaciones como CISA los han

adoptado activamente para capacitación en respuesta a incidentes (Cybersecurity and Infrastructure Security Agency, 2024). Tectonic, implementa este enfoque mediante Infraestructura como Código, automatizando el despliegue reproducible de escenarios de entrenamiento (Grupo de Seguridad Informática (GSI), FING, UDELAR, 2025). Sin embargo, la plataforma no dispone de mecanismos para generar los artefactos forenses que los escenarios de DFIR requieren, lo que obliga a los instructores a construirlos manualmente: capturas de tráfico de red, sistemas de archivos con indicadores de compromiso, volcados de memoria con rastros de malware. La literatura académica reconoce esta carencia y destaca la necesidad de herramientas que produzcan artefactos forenses sintéticos técnicamente precisos y reproducibles para contextos educativos (Scanlon, Du, y Lillis, 2017).

Esta problemática se manifiesta en tres planos concretos. El primero es la carga operativa: construir manualmente los artefactos de cada escenario es laborioso y no escala cuando múltiples estudiantes deben recibir instancias distintas en forma simultánea. El segundo es la ausencia de actividad de fondo coherente: los conjuntos de datos sintéticos típicamente carecen del comportamiento benigno acumulado —conocido como *wear-and-tear*— que caracteriza a los sistemas reales, y que es imprescindible para que los estudiantes aprendan a distinguir evidencia relevante de actividad de fondo, replicando las condiciones reales de una investigación forense. El tercero es la reproducibilidad: los entornos pedagógicos exigen a la vez determinismo dentro de cada instancia —para que el instructor pueda validar los hallazgos de un estudiante— y variabilidad entre instancias —para impedir la transferencia de soluciones—, combinación que la mayoría de las plataformas no provee de forma nativa (Göbel, Schäfer, Hachenberger, Türr, y Baier, 2020). El análisis de antecedentes del Capítulo 3 completa este diagnóstico con problemáticas adicionales, entre ellas la coherencia temporal entre artefactos de distintas fuentes y la diversidad de tipos de evidencia necesaria para replicar escenarios en múltiples dimensiones forenses.

1.1. Objetivos

El objetivo de este proyecto es expandir Tectonic, para que cuente con la capacidad de especificar y generar artefactos forenses sintéticos, integrándose al ecosistema existente de la plataforma Tectonic. Para alcanzarlo se plantearon cuatro objetivos específicos:

1. Realizar una investigación del estado del arte en DFIR y de DFIR en Cyber Ranges, sintetizando distintos marcos de referencia para la clasificación y generación de artefactos digitales, que permita analizar la viabilidad y diseñar una solución para generar escenarios forenses.

2. Diseñar una solución que permita la creación de escenarios de tipo DFIR compatible con Tectonic.

3. Implementar un prototipo funcional de la solución diseñada.

4. Validar la solución mediante la construcción de un caso de estudio práctico, materializado en un escenario de laboratorio que permita probar las capacidades agregadas.

1.2. Resultados

Los resultados de este proyecto abarcan tanto contribuciones conceptuales como técnicas, culminando en una solución funcional y validada. El aporte principal no es solo la extensión de Tectonic para la generación de escenarios forenses, sino la creación de un marco de trabajo completo para la enseñanza de DFIR dentro del mismo.

1. Una investigación del estado del arte en DFIR y en DFIR en diversos Cyber Ranges, que finalizó con un análisis completo de los distintos tipos de artefactos existentes y de diferentes formas de integrarlos con el Cyber Range.

2. La creación de un lenguaje de especificación que permite declarar artefactos desde un nivel atómico hasta escenarios complejos, el cual fue diseñado para artefactos en sistema de archivos, red y memoria.

3. La creación de una solución en capas, utilizando Python para la generación de artefactos, Ansible como herramienta de orquestación y Jinja2 como motor de plantillas, con implementación funcional para artefactos de sistema de archivos y red.

4. La creación de un escenario de análisis forense, que valida la solución desarrollada en un caso de uso real dentro del Cyber Range Tectonic.

1.3. Organización del Informe

El resto del informe desarrolla estas contribuciones de la siguiente manera:

- Capítulo 2: Marco Teórico — Presenta los fundamentos conceptuales del proyecto.
- Capítulo 3: Análisis de la Solución — Expone los antecedentes, el análisis de artefactos forenses y los requerimientos derivados.
- Capítulo 4: Diseño de la Solución — Describe la arquitectura en capas y las decisiones de diseño.
- Capítulo 5: Implementación — Detalla la implementación de las vertientes de red y sistema de archivos.
- Capítulo 6: Experimentación — Presenta el caso de estudio de ransomware y sus resultados.

- Capítulo 7: Conclusiones y Trabajo Futuro

El código fuente completo de la solución, integrado con la plataforma Tectonic, junto con documentos anexos —manual de usuario, estado del arte y diseño de la solución— se encuentran disponibles en el repositorio del proyecto: `https://gitlab.fing.edu.uy/gsi/dfir-tectonic`.

Capítulo 2

Marco Teórico

El presente capítulo expone los fundamentos conceptuales y tecnológicos que sustentan el desarrollo de este proyecto. Se describen inicialmente los ecosistemas de entrenamiento conocidos como *cyber ranges*, con especial énfasis en la plataforma Tectonic. Posteriormente, se profundiza en el Análisis Forense Digital y Respuesta ante Incidentes (DFIR), estableciendo los marcos metodológicos y las herramientas estándar de la industria que guían los artefactos y la generación de los mismos.

2.1. Cyber ranges

Los *cyber ranges* emergieron como respuesta a la creciente sofisticación de los ataques informáticos y la necesidad de formar profesionales capaces de enfrentarlos en entornos seguros y controlados. El Instituto Nacional de Estándares y Tecnología (NIST) los define como 'representaciones interactivas y simuladas de la red local, sistemas, herramientas y aplicaciones de una organización' (Ukwandu y cols., 2020). En términos más amplios, un ejercicio de seguridad sobre un cyber range consiste en una actividad de entrenamiento que ejecuta escenarios de ataque y/o defensa en entornos virtuales y/o físicos, con el objetivo de mejorar las habilidades ofensivas y/o defensivas de los participantes (Yamin, Katt, y Gkioulos, 2019).

Los cyber ranges cumplen múltiples objetivos que pueden agruparse en cuatro grandes categorías. En primer lugar, el entrenamiento, que permite desarrollar habilidades prácticas mediante la ejecución de ejercicios en entornos controlados. En segundo lugar, la educación, que facilita el aprendizaje de conceptos a través de laboratorios y escenarios guiados. En tercer lugar, la investigación, que proporciona plataformas para experimentar con nuevas técnicas, herramientas y metodologías en condiciones reproducibles y aisladas del entorno productivo. Finalmente, la evaluación, que permite medir el desempeño de sistemas, herramientas o usuarios frente a distintos escenarios, constituyendo una base objetiva para la mejora continua de las capacidades defensivas de una organización.

Estos objetivos reflejan el rol de los cyber ranges como plataformas versátiles que pueden adaptarse a distintos contextos, incluyendo entornos académicos, industriales y militares (Yamin y cols., 2019).

El escenario constituye el componente central de un cyber range. Se define como el elemento que establece el entorno de ejecución y la narrativa que guía los pasos de un ejercicio de entrenamiento o prueba. Un escenario se clasifica a partir de cuatro dimensiones: cuál es su propósito, dónde se ejecuta, cómo se ejecuta y qué herramientas utiliza. El ciclo de vida de un escenario comprende las etapas de creación, generación, edición, despliegue, ejecución y destrucción, con módulos especializados para cada fase e implementaciones de orquestación que integran los componentes de forma coordinada (Yamin y cols., 2019).

La arquitectura de un cyber range se organiza en tres capas principales: tecnologías de núcleo, tecnologías de infraestructura y tecnologías de *front-end*. Las tecnologías de núcleo incluyen emulación, virtualización, simulación y contenedorización. La capa de infraestructura gestiona y controla el cyber range, ubicándose entre el núcleo y la capa de usuario. La capa de *front-end* constituye el puente entre el usuario y el núcleo, implementada típicamente mediante tecnologías web (Ukwandu y cols., 2020).

Funcionalmente, los cyber ranges se organizan en componentes que cubren distintas necesidades operativas. El componente de gestión administra roles, recursos computacionales y el ciclo completo de ejercicios y experimentos. El módulo de monitoreo recopila y analiza registros de múltiples fuentes durante la ejecución, capturando el progreso de los participantes y evaluando su desempeño a lo largo de los distintos escenarios, validando asimismo el estado de la plataforma y los servicios disponibles. El módulo de entrenamiento y aprendizaje provee tutorías, mecanismos de puntuación y análisis post-ejercicio. Finalmente, el módulo de almacenamiento de datos guarda definiciones de escenarios, reglas de ejercicio, resultados y herramientas requeridas para la ejecución, actuando como biblioteca de referencia para el diseño de nuevos ejercicios ajustados al nivel de los participantes (Yamin y cols., 2019).

Los cyber ranges descritos en la literatura general están diseñados principalmente para el entrenamiento en ataque y defensa. El entrenamiento en análisis forense digital y respuesta a incidentes, por su parte, requiere que el entorno sea capaz de proveer artefactos forenses sobre los cuales el estudiante pueda ejercitar la adquisición y el análisis. Con ese fin, existe un conjunto de *frameworks* de generación de artefactos orientados a este propósito. Entre los más relevantes se encuentran EviPlant (Scanlon y cols., 2017), hystck (Göbel y cols., 2020), TraceGen (Du, Hargreaves, Sheppard, y Scanlon, 2021) y ForTrace (Göbel, Breitinger, y Baier, 2024). Estos sistemas no son cyber ranges en el sentido arquitectónico mencionado anteriormente, sino plataformas de generación de conjuntos de artefactos forenses.

2.2. Tectonic

Tectonic es un cyber range académico desarrollado por el Grupo de Seguridad Informática de la Facultad de Ingeniería (UdelaR) orientado a la creación y gestión de escenarios de ciberseguridad para entornos educativos (Grupo de Seguridad Informática (GSI), FING, UDELAR, 2025). Su diseño adopta el paradigma de infraestructura como código (IaC): los escenarios se especifican de forma declarativa en archivos YAML, y a partir de esa definición la plataforma orquesta automáticamente el despliegue, la configuración y el ciclo de vida completo de cada instancia. En la figura 2.1 se presenta la arquitectura general del sistema.

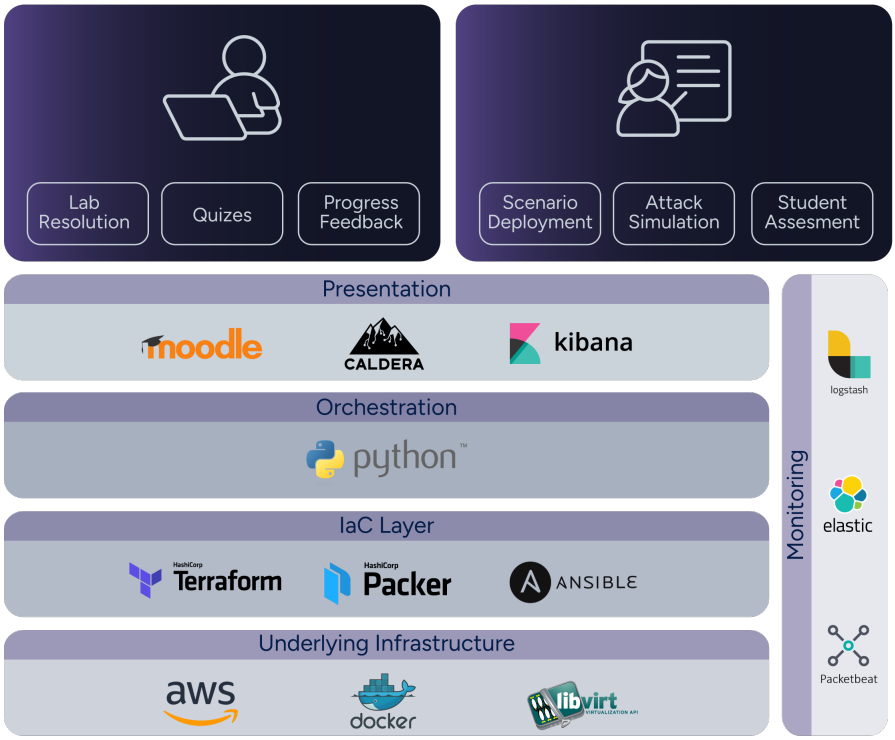


Figura 2.1: Arquitectura de Tectonic. Obtenida de (Grupo de Seguridad Informática (GSI), FING, UDELAR, 2025)

2.2.1. Ciclo de vida de un escenario

El proceso mediante el cual un escenario pasa de su especificación a un entorno ejecutable comprende cuatro etapas. En la primera, Packer construye las imágenes base de los sistemas operativos a partir de una plantilla única, generando discos virtuales que incluyen el sistema operativo, software preinstalado y

configuración inicial; estas imágenes son el punto de partida compartido del que se clonan las instancias individuales de cada participante (HashiCorp, 2025a). En la segunda etapa, Terraform materializa la topología de red definida en el escenario, creando las instancias de máquinas virtuales y los recursos de red asociados de forma declarativa (HashiCorp, 2025b). En la tercera etapa, Ansible ejecuta los playbooks del escenario sobre las instancias ya desplegadas, aplicando configuraciones específicas como la instalación de herramientas, la inicialización de servicios y la inyección de artefactos propios del ejercicio (Red Hat, Inc., 2025); esta es la única capa cuyo contenido es responsabilidad directa del instructor que diseña el escenario. Finalmente, la plataforma activa los servicios de monitoreo y evaluación mediante la pila Elastic como SIEM (Elastic N.V., 2025) y, cuando el escenario lo requiere, el framework de emulación adversaria Caldera (MITRE Corporation, 2025).

La gestión del ciclo de vida completo —despliegue, encendido, apagado y destrucción de instancias— se realiza a través de la interfaz de línea de comandos de Tectonic, que permite al instructor operar sobre réplicas individualizadas y aisladas para cada participante o grupo.

2.2.2. Ansible y Jinja como punto de extensión

Dado que los playbooks de Ansible constituyen la capa que el instructor controla directamente, comprender su funcionamiento es relevante para este trabajo. Ansible es una herramienta de automatización sin agentes que se comunica con los nodos gestionados mediante SSH y describe el estado final deseado de cada sistema en archivos YAML denominados playbooks (Red Hat, Inc., 2025). Cada playbook está compuesto por uno o más *plays* que asocian un conjunto de tareas a un grupo de hosts objetivo; las tareas invocan módulos nativos de Ansible para gestionar paquetes, archivos, servicios y usuarios sin necesidad de scripts imperativos. Una propiedad central de este modelo es la idempotencia: si el sistema ya se encuentra en el estado descrito, la tarea no realiza ninguna acción, lo que hace que las ejecuciones repetidas sean predecibles y seguras ante interrupciones.

Cuando la lógica requerida excede las capacidades de los módulos estándar, Ansible permite incorporar módulos personalizados escritos en algún lenguaje de programación, entre los cuales se encuentra Python (Canny y python-docx Contributors, 2013). Estos módulos reciben sus argumentos desde el playbook a través de la clase `AnsibleModule`, ejecutan lógica arbitraria sobre el nodo gestionado y devuelven su resultado en formato JSON; ubicados en el directorio `library/`, quedan disponibles para todas las tareas sin configuración adicional. Esta extensibilidad es el mecanismo que este trabajo aprovecha para incorporar nueva funcionalidad a Tectonic.

Para la generación dinámica de archivos de configuración, Ansible se apoya en Jinja, un motor de plantillas que permite parametrizar contenido a partir de variables definidas en el escenario (Ronacher, 2025). La combinación de playbooks declarativos, módulos personalizados en Python y plantillas Jinja constituye el punto de extensión sobre el que se construye la solución propuesta.

2.3. Fundamentos de Análisis Forense Digital y Respuesta ante Incidentes

La forensia digital se define como la aplicación de la ciencia a la identificación, recolección, examinación y análisis de datos, preservando la integridad de la información y manteniendo una estricta cadena de custodia (Kent y cols., 2006). Constituye un componente fundamental de la respuesta ante incidentes, en tanto su aplicación permite a los investigadores comprender la cadena de eventos que condujo a una acción maliciosa (Johansen, 2020). La combinación de ambas disciplinas conforma lo que se conoce como DFIR, orientado tanto a la investigación técnica de los hechos como a la contención y mitigación del incidente.

Los objetivos de DFIR abarcan múltiples dimensiones según el contexto. En el ámbito operativo, busca contener la amenaza activa, erradicar la presencia del atacante y restaurar las operaciones normales. Desde la perspectiva investigativa, persigue determinar la causa raíz del incidente y reconstruir las acciones del atacante desde el compromiso inicial hasta la detección. En contextos legales, el proceso forense provee evidencia que debe cumplir con los requisitos de admisibilidad en procedimientos judiciales (Johansen, 2020).

2.3.1. Proceso de respuesta ante incidentes

El proceso de respuesta ante incidentes se articula en seis fases que se ejecutan de forma cíclica: cada incidente retroalimenta la preparación ante los siguientes, de modo que toda respuesta contribuye a fortalecer la postura organizacional ante futuros eventos. La Figura 2.2 ilustra este ciclo (Johansen, 2020).

Sin una adecuada *preparación*, la respuesta resultará desorganizada y puede agravar la situación. Esta fase comprende la creación del plan de respuesta, la incorporación de herramientas forenses y la capacitación continua del personal. La *detección* es el momento en que la organización toma conocimiento por primera vez de eventos que posiblemente indican actividad maliciosa, ya sea a través de alertas de controles de seguridad, de terceros externos o de los propios usuarios. Durante el *análisis*, el personal recolecta evidencia de sistemas tales como la memoria en ejecución, archivos de registro y conexiones de red, con el objetivo de determinar la causa raíz del incidente y reconstruir las acciones del atacante desde el compromiso inicial hasta la detección.

Una vez que se comprende el alcance del incidente, la *contención* limita la capacidad del atacante de comprometer otros recursos o exfiltrar datos. La *erradicación y recuperación* remueve al atacante mediante soluciones anti-malware, reimagen de sistemas comprometidos, eliminación de cuentas vulneradas y/o aplicación de parches. Finalmente, la *actividad post-incidente* comprende una revisión completa de las acciones tomadas, cuyas lecciones aprendidas retroalimentan la fase de preparación del siguiente ciclo (Johansen, 2020).

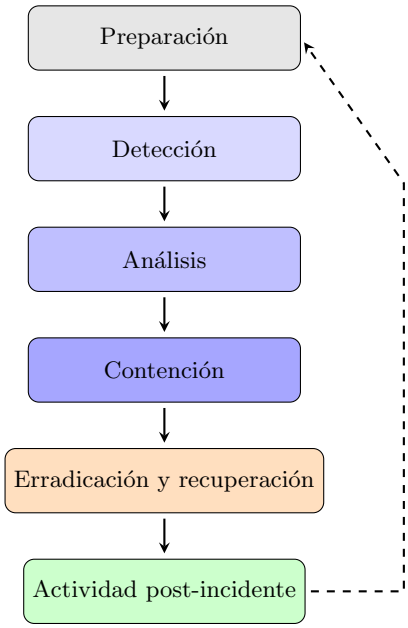


Figura 2.2: Proceso de respuesta ante incidentes. Adaptada de (Johansen, 2020)

2.3.2. Proceso forense digital

El proceso forense digital define el flujo de la evidencia digital desde su primera identificación hasta su presentación ante la dirección o ante una instancia judicial (Johansen, 2020). Para estructurarlo, se adopta el framework del *Digital Forensics Research Workshop* (DFRWS), que comprende seis elementos ilustrados en la Figura 2.3.

La *identificación* se apoya en el principio de intercambio de Locard, según el cual todo contacto entre dos objetos deja una traza en cada uno. Este principio subyace a la búsqueda forense: dado que ningún actor puede interactuar con un sistema sin dejar rastro, siempre existe evidencia potencial —tanto del atacante como de la propia actividad del analista sobre la escena. La *preservación* protege esa evidencia de cualquier modificación o eliminación, mediante controles físicos y lógicos sobre los sistemas sospechosos y la documentación de la cadena de custodia. La *recolección* inicia la adquisición de la evidencia teniendo en cuenta su naturaleza volátil: datos como la memoria RAM deben ser adquiridos antes que la evidencia persistente, pues se pierden al apagar el sistema.

La *examinación* aplica herramientas y técnicas forenses para descubrir y extraer datos de la evidencia adquirida, manteniendo en todo momento el cuidado necesario para evitar su contaminación. El *análisis* consiste en la interpretación y correlación de los datos extraídos durante la examinación, integrándolos con otras fuentes de evidencia para reconstruir los eventos ocurridos. La *presentación* comunica los hallazgos de forma clara, concisa e imparcial, tanto en un informe escrito como, cuando el caso lo requiere, en el testimonio del examinador ante

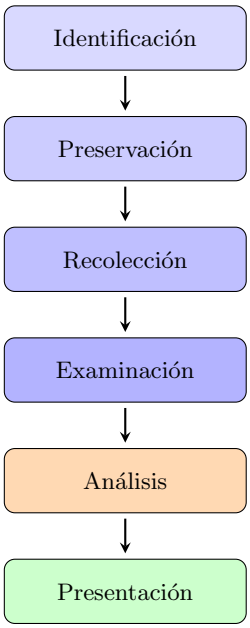


Figura 2.3: Proceso forense digital (framework DFRWS). Adaptada de (Johansen, 2020)

una instancia judicial (Johansen, 2020).

2.3.3. Dominios de evidencia forense

Johansen estructura el análisis forense a partir de tres dominios de evidencia. El análisis de red incorpora técnicas que permiten obtener información sobre la actividad de red asociada a posibles amenazas, dado que los adversarios están sujetos a los mismos protocolos de red que rigen el tráfico normal. El análisis de memoria explota la riqueza de datos contenidos en la RAM durante la ejecución del sistema, que incluyen procesos activos, librerías cargadas, controladores de dispositivos, claves de registro abiertas, conexiones de red e historial de comandos. El análisis de sistema de archivos complementa los anteriores: aunque la causa raíz de un incidente puede determinarse a partir de evidencia de red o de memoria, el almacenamiento del sistema contiene una cantidad masiva de material que los analistas pueden aprovechar para reconstruir el incidente (Johansen, 2020).

Este trabajo adopta dicha estructura como marco conceptual, organizando el análisis forense en tres ejes: forensia de red, forensia de memoria volátil y forensia de sistema de archivos. Desde la perspectiva de la formación, cada dominio exige competencias diferenciadas. El análisis de red demanda el dominio de protocolos y la capacidad de correlacionar eventos distribuidos en el tiempo. La forensia de memoria entrena la habilidad de razonar sobre el estado dinámico

de un sistema en ejecución, identificando artefactos que no dejan rastros persistentes. El análisis de almacenamiento desarrolla la capacidad de reconstruir líneas temporales a partir de metadatos y de distinguir actividad relevante de actividad de fondo. Transversal a los tres dominios, la documentación de hallazgos constituye una competencia en sí misma: el analista debe registrar cada artefacto identificado, las herramientas utilizadas y las conclusiones alcanzadas, de forma que otro investigador pueda reproducir y validar el análisis de forma independiente.

2.3.4. Artefacto Forense Digital

En el ámbito forense digital conviven tres conceptos relacionados pero conceptualmente distintos: artefacto, evidencia y prueba. Las actividades que un usuario realiza en un sistema generan datos remanentes que persisten en él con independencia de cualquier proceso investigativo (Casey, 2011). Estos rastros —registros de sistema, historiales de navegador, entradas de registro, capturas de tráfico de red, volcados de memoria— son lo que en este trabajo se denomina *artefacto forense digital*: trazas de información presentes en todo sistema en operación, resultado del uso normal del sistema o sus aplicaciones.

La *evidencia digital* es el nivel siguiente: Carrier distingue explícitamente entre los usos investigativos y legales de la evidencia, señalando que puede haber situaciones en que los datos recolectados no puedan ser admitidos en un tribunal (Carrier, 2005). En su uso investigativo, define la evidencia digital como un objeto digital que contiene información confiable que apoya o refuta una hipótesis. El artefacto no se convierte en evidencia por sus propiedades intrínsecas, sino por el tratamiento metodológico que recibe: su identificación, recolección y preservación dentro de un proceso investigativo formal, manteniendo su integridad mediante cadena de custodia, según las prácticas establecidas por el NIST (Kent y cols., 2006).

Finalmente, la *prueba forense* corresponde al nivel legal: la investigación forense digital desarrolla y somete a prueba teorías que pueden ser presentadas ante un tribunal (Carrier, 2005). Es la evidencia que ha sido admitida en un procedimiento judicial con valor legal para atribuir responsabilidad (Casey, 2011).

Esta distinción es relevante para el presente trabajo: la solución desarrollada genera *artefactos* forenses digitales sintéticos, no evidencia en sentido estricto, dado que su propósito es educativo y no implica cadena de custodia ni validez legal.

Artefacto forense digital curado

Si bien la literatura clásica se enfoca en el análisis de artefactos forenses existentes, en contextos como el entrenamiento, la experimentación o los cyber ranges surge la necesidad de trabajar con artefactos generados de forma controlada. En estos escenarios, los artefactos no son simplemente recolectados, sino diseñados, generados y preparados intencionalmente para su análisis.

Es entonces que se introduce el concepto de *artefacto forense digital curado*, para referirse a elementos que han sido aceptados para inclusión en una base de datos de artefactos forenses, con campos requeridos que encapsulan su significado y contexto investigativo.

A diferencia de los artefactos obtenidos directamente de incidentes reales, los artefactos forenses digitales curados son creados en entornos controlados, pero replican de forma fiel las características estructurales, temporales y contextuales de los artefactos de escenarios reales (Harichandran, Walnycky, Baggili, y Breitinger, 2016).

2.3.5. Forensia del Sistema de Archivos

El análisis forense de sistemas de archivos examina la información almacenada en medios de almacenamiento persistente con el objetivo de reconstruir eventos, identificar modificaciones y recuperar datos eliminados o alterados. (Carrier, 2005).

La forensia de sistemas de archivos se fundamenta en comprender que los artefactos digitales trascienden los archivos visibles para el usuario. Los sistemas de archivos mantienen múltiples capas de estructuras de datos, metadatos y espacios residuales que preservan información incluso después de operaciones de eliminación o modificación. Esta característica permite a los investigadores forenses reconstruir líneas temporales de actividad, recuperar archivos borrados y detectar intentos de ocultamiento o destrucción de evidencia.

El sistema de archivos puede conceptualizarse como una máquina de estados donde cada operación (creación, modificación, eliminación, cambio de permisos) transita entre estados discretos dejando rastros en múltiples estructuras de datos. Cuando un usuario elimina un archivo, el sistema operativo no borra inmediatamente su contenido del disco, sino que modifica metadatos específicos marcando los bloques de datos como disponibles para reutilización. El contenido original permanece intacto hasta que nuevos datos lo sobrescriban, creando oportunidades para recuperación forense. Esta propiedad fundamental explica por qué la eliminación lógica difiere sustancialmente de la eliminación física, concepto central en análisis forense (Carrier, 2005).

La distinción entre tipos de evidencia resulta de utilidad para planificar estrategias de análisis. La evidencia activa corresponde a archivos y directorios actualmente accesibles mediante el sistema operativo, visibles a través de herramientas convencionales de exploración. La evidencia residual, por contraste, existe en espacios no asignados del disco (bloques marcados como libres pero aún conteniendo datos antiguos), en espacio slack (fragmentos finales de bloques parcialmente utilizados) o en estructuras de metadatos sobrescritas parcialmente. Gran parte de la evidencia forense más valiosa, como archivos eliminados intencionalmente o fragmentos de actividad previa, reside en estas áreas residuales que requieren técnicas especializadas de recuperación.

Un principio metodológico fundamental consiste en evitar dependencia de herramientas proporcionadas por el sistema operativo bajo análisis. Estas herramientas pueden estar comprometidas por malware, manipuladas por el atacante

o simplemente proporcionar información filtrada que oculte evidencia relevante. El análisis forense confiable requiere inspección directa de las estructuras crudas del sistema de archivos, interpretando manualmente tablas de inodos, mapas de bits de asignación, entradas de directorio y estructuras de journal, sin intermediación del sistema operativo potencialmente comprometido.

En un entorno de entrenamiento, el análisis forense de sistema de archivos desarrolla tres competencias centrales. Primero, la capacidad de reconstrucción temporal: correlacionar timestamps de múltiples artefactos para establecer la secuencia de eventos del incidente. Segundo, la habilidad de filtrado: distinguir actividad benigna —el wear-and-tear del sistema— de actividad maliciosa. Tercero, el razonamiento sobre ausencias: interpretar la eliminación deliberada de artefactos como evidencia en sí misma, infiriendo acciones del atacante a partir de huecos en el registro de actividad (Carrier, 2005).

2.3.6. Forensia de la Memoria Volátil

La forensia de memoria analiza la memoria volátil (RAM) de sistemas comprometidos con el objetivo de reconstruir incidentes de ciberseguridad, identificar actividad maliciosa y recuperar artefactos críticos que no persisten en almacenamiento. A diferencia del análisis de sistemas de archivos, que examina evidencia histórica relativamente estable, la forensia de memoria captura el estado dinámico del sistema en ejecución, incluyendo procesos activos, conexiones de red establecidas, claves criptográficas en uso, módulos cargados dinámicamente y técnicas de ocultamiento empleadas por atacantes avanzados (Ligh, Case, Levy, y Walters, 2014; Casey, Richard, y James, 2014).

La relevancia de este dominio radica en que numerosos artefactos forenses críticos existen exclusivamente en memoria y desaparecen al apagar el sistema. Procesos maliciosos sin respaldo en disco, hooks inyectados en funciones del kernel, credenciales activas y conexiones de red efímeras constituyen evidencia que las técnicas tradicionales basadas en análisis de disco no pueden recuperar (Ligh y cols., 2014). La adquisición y el análisis de memoria pueden realizarse en paralelo con la adquisición del disco, lo que permite obtener contexto operativo temprano sobre la naturaleza del incidente sin necesidad de esperar el procesamiento completo de la imagen de disco (DFRWS, 2019).

El análisis de memoria presenta desafíos metodológicos específicos derivados de la naturaleza volátil y dinámica de la información. La RAM debe capturarse mediante volcados forenses que congelen su estado en un momento específico, evitando alteraciones que comprometan la integridad de la evidencia. Adicionalmente, las estructuras de datos en memoria varían significativamente entre sistemas operativos y versiones de kernel, requiriendo conocimiento profundo de la arquitectura interna para interpretarlas correctamente.

La metodología de análisis se fundamenta en la inspección directa de estructuras de datos mantenidas por el kernel del sistema operativo. En lugar de confiar en herramientas del sistema comprometido, que podrían estar manipuladas por el atacante, se examina la memoria cruda identificando listas enlazadas de procesos, tablas de descriptores de archivos, estructuras de memoria virtual

y objetos del subsistema de red. Este enfoque permite detectar inconsistencias provocadas por técnicas anti-forenses, como procesos ocultados mediante manipulación de listas del kernel o módulos cargados sin registro formal (Ligh y cols., 2014).

El proceso constituye la unidad fundamental de análisis en forensia de memoria, caracterizado por dos dimensiones complementarias. Por un lado, sus metadatos incluyen identificadores de proceso (PID) y proceso padre (PPID), información de usuario propietario y marcas temporales de creación y ejecución. Por otro lado, su contexto de ejecución comprende los comandos invocados, librerías dinámicas cargadas (DLLs en Windows, objetos compartidos en Linux), archivos abiertos, conexiones de red activas, variables de entorno y recursos asignados por el kernel. El análisis aislado de procesos individuales resulta insuficiente, la reconstrucción del contexto completo, incluyendo relaciones jerárquicas entre procesos padre-hijo, correlación temporal de eventos y uso compartido de recursos, permite establecer líneas temporales precisas y comprender la secuencia de acciones que condujeron al incidente (Ligh y cols., 2014).

2.3.7. Forensia de Red

La forensia de red se ocupa de la identificación, adquisición, análisis y correlación de evidencia digital derivada del tráfico de red, con el objetivo de reconstruir eventos, detectar actividad maliciosa y responder a incidentes de seguridad. A diferencia de otros dominios del análisis forense digital, la evidencia de red es inherentemente volátil, distribuida y dependiente del tiempo, lo que impone desafíos particulares para su recolección y preservación (Casey, 2011).

En este contexto, el análisis forense tradicional posterior a un incidente resulta limitado, ya que gran parte de la información relevante solo existe mientras las comunicaciones están ocurriendo. Por este motivo, la forensia de red moderna se apoya fuertemente en enfoques de monitoreo continuo que permitan observar, registrar y analizar el comportamiento de la red en tiempo real.

El concepto de Network Security Monitoring (NSM) constituye una disciplina operativa orientada a la detección temprana y respuesta ante incidentes. En lugar de enfocarse exclusivamente en el análisis forense posterior, este enfoque propone la implementación de sensores y sistemas de recolección que proporcionen visibilidad sostenida sobre la actividad de red (Bejtlich, 2013).

Un concepto central del enfoque NSM es la visibilidad, entendida como la capacidad de observar el comportamiento real de la red a distintos niveles de detalle. Sin una visibilidad adecuada, la detección de incidentes y la posterior reconstrucción de eventos se vuelven incompletas o directamente inviables. Esta visibilidad se logra mediante la recolección sistemática de múltiples tipos de datos de red, cada uno con distintos costos y beneficios analíticos. (Bejtlich, 2013)

Para que una captura de tráfico de red sintético sea considerada realista y útil en entornos virtualizados, debe cumplir con una serie de requerimientos técnicos, entre los cuales se incluyen la corrección de los paquetes, el cumplimiento de estándares de protocolo, la precisión temporal, la reproducibilidad y

la adaptabilidad a escenarios dinámicos (Spiekermann y Keller, 2022).

Las competencias desarrolladas mediante el análisis forense de red incluyen la interpretación de protocolos en distintas capas del modelo de red, la identificación de patrones de comportamiento anómalo en flujos de tráfico, y la correlación de eventos de red con artefactos de sistema de archivos y memoria para construir una narrativa unificada del incidente. La generación sintética de capturas de red en el cyber range permite que el estudiante practique estas habilidades en escenarios controlados donde la actividad maliciosa ha sido diseñada con características conocidas, facilitando la evaluación objetiva del análisis realizado.

Capítulo 3

Análisis de la Solución

El objetivo de este capítulo es analizar los artefactos forenses existentes, basado en literaturas previas sobre el tópico y los atributos que estos poseen. Además se establecen los requerimientos transversales que deben cumplir los tres ejes principales del proyecto: artefactos de red, sistema de archivos y memoria.

3.1. Antecedentes

Al analizar la generación de escenarios para el cyber range, se identificaron diversas problemáticas que han sido abordadas previamente en la literatura. A continuación se presentan dichas problemáticas junto con los enfoques propuestos por distintos autores.

La principal problemática identificada en investigaciones previas es la generación de escenarios forenses educativos realistas. El uso de datos reales se encuentra restringido por consideraciones éticas, legales y de privacidad, lo que obliga a recurrir a la generación de evidencia sintética (Du y cols., 2021). Sin embargo, este enfoque también presenta dificultades: la generación manual de evidencia sintética puede resultar costosa en términos de tiempo, recursos y veracidad. A esto se suma el riesgo asociado a la reutilización de los mismos casos forenses en múltiples instancias evaluativas, lo cual incrementa la probabilidad de deshonestidad académica (Korkosh, Samsonsen, y Moan, 2025).

Para abordar la problemática de la veracidad, la literatura ha adoptado distintos enfoques. En primer lugar, se propone la emulación de la generación de evidencia mediante scripting, de modo que la propia máquina virtual genera automáticamente los artefactos forenses correspondientes. En segundo lugar, se plantea la construcción de paquetes de evidencia inyectables a partir de una imagen base de sistema operativo (Scanlon y cols., 2017). Un tercer enfoque propone una arquitectura modular en la que componentes especializados generan los artefactos mediante un lenguaje declarativo.

Otra problemática relevante es la producción de actividad de fondo que otorgue realismo a los escenarios. Los sistemas reales contienen artefactos generados

por el uso cotidiano durante meses o años, mientras que la evidencia maliciosa representa una fracción mínima del total (Göbel y cols., 2024). La ausencia de esta actividad impide simular las condiciones reales de análisis, en las que el analista debe distinguir la actividad benigna de la evidencia relevante. Además, dicha actividad debe ser coherente tanto temporal como contextualmente (Antal y Miclea, 2024). La literatura aborda esta problemática mediante distintos enfoques: emulación de acciones mediante scripting, inserción de archivos mediante funciones matemáticas que distribuyen artefactos cotidianos entre los artefactos relevantes, control de interfaces gráficas mediante visión por computadora (Schmidt, Škrabal, y Franke, 2023), y generación de historiales de actividad coherentes mediante modelos de lenguaje de gran escala (Antal y Miclea, 2024).

La reproducibilidad constituye otra problemática significativa en los escenarios forenses educativos: los estudiantes deben recibir ejercicios de dificultad equivalente, y los instructores deben poder regenerar exactamente los mismos escenarios para validar soluciones o investigar inconsistencias. La literatura aborda este problema desde distintas perspectivas. Un enfoque consiste en generar los artefactos mediante archivos de configuración que especifican qué artefactos se producen, en qué momento y cómo se relacionan entre sí, de modo que el mismo archivo produce siempre el mismo conjunto de datos. Otro enfoque combina un motor de plantillas con variables generadas pseudoaleatoriamente y el uso de semillas deterministas, garantizando que una misma semilla reproduzca exactamente el mismo escenario.

Finalmente, la gestión de timestamps representa una problemática adicional en la generación de evidencia forense sintética. Los sistemas de archivos asocian múltiples marcas temporales a cada archivo (creación, modificación, acceso, entre otras), las cuales deben mantener relaciones lógicas coherentes entre sí (Carrier, 2005). La literatura aborda esta cuestión mediante la manipulación directa del reloj del sistema antes de la creación de cada artefacto, de forma que el sistema operativo asigna automáticamente la marca temporal correcta .

Todos los enfoques planteados en la literatura presentan ventajas y desventajas según el aspecto considerado, las cuales serán discutidas a lo largo del informe en la medida en que resulte pertinente.

3.2. Artefactos Forenses

Las problemáticas identificadas en los antecedentes —realismo de los escenarios, incorporación de actividad de fondo, reproducibilidad y coherencia temporal— convergen en un objeto técnico común: el artefacto forense (definido en la subsección 2.3.4). Comprender qué variantes existen, qué propiedades deben controlarse durante su generación y en qué contexto serán analizados resulta indispensable para derivar requerimientos concretos. Las subsecciones siguientes desarrollan esta caracterización, que constituye la base del análisis de requerimientos presentado al cierre del capítulo.

3.2.1. Diversidad y Alcance de los Artefactos

Cada rama dentro de la forensia comprende múltiples tipos de artefactos. Por ejemplo: dentro de red se encuentran paquetes individuales, sesiones, capturas completas de tráfico. Dentro de sistema de archivos se encuentran artefactos como archivos, directorios y sistemas de archivos completos, y dentro de la vertiente de memoria se encuentran los procesos, las tablas de procesos y los volcados de memoria.

La enumeración anterior no es exhaustiva: cada vertiente contiene una gran cantidad de artefactos posibles, lo que hace inviable modelarlos explícitamente en su totalidad. Sin embargo, puede observarse que muchos artefactos complejos son combinaciones o estructuras derivadas de artefactos más básicos —una captura de tráfico puede entenderse como un conjunto ordenado de paquetes, un sistema de archivos como una estructura jerárquica de archivos y directorios, y un árbol de procesos como una relación entre procesos individuales.

De esta observación se desprende que el estudio de artefactos básicos por cada vertiente podría resultar como fundamento para modelar estructuras de mayor complejidad. Este enfoque, sin embargo, no es suficiente por sí solo: dado que el objetivo es la generación de artefactos curados, no basta con producirlos de forma genérica. Es necesario controlar de forma explícita los atributos con relevancia forense —marcas temporales, nombres, ubicaciones, contenido— de modo que sea posible generar escenarios donde el estudiante pueda focalizar su análisis en un atributo particular.

3.2.2. Actividad de Fondo y No Trivialidad

Un aspecto central identificado es la necesidad de incorporar actividad de fondo. La mera existencia de un artefacto relevante no garantiza un escenario pedagógicamente adecuado si su identificación resulta inmediata.

Por lo tanto, los artefactos de interés deben coexistir con otros elementos del mismo dominio (archivos adicionales, tráfico legítimo, procesos benignos), de modo que la tarea de identificación requiera razonamiento y no mera inspección superficial.

Esta consideración introduce una dimensión adicional al problema: no se trata únicamente de generar artefactos puntuales, sino de encapsularlos en un entorno coherente.

3.2.3. Propiedades Transversales de los Artefactos

A pesar de la diversidad existente, los distintos tipos de artefactos comparten propiedades estructurales comunes que trascienden los dominios a los que pertenecen.

La más evidente es la temporalidad. Todo artefacto posee, de una u otra forma, una marca de tiempo: en red indica el momento de envío del paquete, en archivos se manifiesta en tiempos de creación, modificación y acceso, y en procesos representa el instante de inicio y eventualmente de finalización.

Otra propiedad transversal es el contenido asociado al artefacto. Aunque su naturaleza puede variar según el tipo de artefacto —por ejemplo, un archivo puede contener texto, imágenes o datos binarios— todos los artefactos poseen algún tipo de información que constituye su *payload* o contenido principal.

En conjunto, estas similitudes sugieren que los artefactos forenses, a pesar de pertenecer a dominios distintos, comparten una estructura conceptual común articulada en torno a tres ejes: identidad, contenido y temporalidad.

3.2.4. Modalidades de Artefactos

El análisis permite identificar dos dimensiones independientes que caracterizan la entrega y la generación de artefactos en contextos educativos orientados a DFIR.

La primera dimensión es la *modalidad de distribución*. El artefacto puede entregarse de forma estática —como un conjunto de artefactos previamente capturados, por ejemplo una imagen de disco, una captura de tráfico o un volcado de memoria— o de forma dinámica, brindando un entorno activo donde el estudiante debe capturar el artefacto antes de analizarlo.

Esta distinción resulta particularmente relevante en el contexto de DFIR. En un escenario centrado en análisis forense, el objeto de estudio suele presentarse como evidencia ya adquirida, delimitada y preservada, lo que corresponde a la modalidad estática. En cambio, en escenarios de respuesta a incidentes, el analista debe interactuar con sistemas en ejecución y realizar la adquisición de evidencia como parte del proceso de investigación, lo que se aproxima a la modalidad dinámica.

La segunda dimensión corresponde al *mecanismo de generación* de los artefactos. Estos pueden producirse de forma *online*, ejecutando acciones reales sobre un sistema en ejecución de modo que los metadatos surjan como consecuencia natural de dichas acciones, o de forma *offline*, generándolos mediante herramientas que operan directamente sobre estructuras de datos sin requerir un sistema activo.

La generación *online* favorece el realismo de los metadatos y la coherencia natural entre eventos, aunque introduce dependencias del entorno de ejecución y limita el control fino sobre atributos específicos. La generación *offline*, por su parte, permite especificar con precisión atributos como marcas temporales, estructuras internas o estados de eliminación, independientemente de la infraestructura subyacente.

3.2.5. Conclusión del Análisis sobre Artefactos Forenses

En síntesis, el análisis permite establecer cuatro conclusiones fundamentales:

1. Existe una gran diversidad de artefactos en cada vertiente forense, lo que hace inviable abordarlos de forma exhaustiva.

2. Muchos artefactos complejos pueden entenderse como combinaciones o extensiones de artefactos básicos.

3. A pesar de su diversidad, los artefactos comparten propiedades estructurales comunes, especialmente en términos de temporalidad, identidad y contexto.

4. La generación y la distribución de artefactos pueden abordarse mediante distintas modalidades, cuya elección impacta en el control sobre los atributos de los artefactos forenses y el tipo de actividad DFIR que se busca ejercitar.

3.3. Requerimientos del Sistema

Es posible derivar un conjunto de requerimientos que la solución propuesta debe satisfacer, basado en el análisis de artefactos previo, requerimientos que se derivan del enfoque educativo y del uso de Tectonic como plataforma base.

A continuación, se describen los requerimientos identificados. Cada requerimiento incluye un criterio de verificación que permite determinar objetivamente si la implementación lo cumple.

- **Generación sintética de evidencia**: Del análisis de antecedentes surge que el uso de datos reales no es viable por restricciones éticas y legales. En consecuencia, la solución debe permitir la generación sintética de evidencia de forma controlada, manteniendo las propiedades técnicas y forenses de artefactos reales. Esto implica crear archivos con contenido realista, tráfico de red con estructura de protocolos correcta, y metadatos coherentes que simulen actividad humana auténtica.

  Ejemplo: Creación de un archivo que en sus metadatos contiene algún tipo de rastro de actividad maliciosa.

  Criterio de verificación: Los artefactos generados deben ser parseables y analizables por herramientas forenses estándar sin errores de formato, y los metadatos de los artefactos deben ser consistentes con el sistema operativo y las aplicaciones simuladas.

  Prioridad: Alta, porque sin capacidad de generación sintética el sistema dependería de casos reales o de evidencia manualmente creada.

- **Generación de actividad de fondo**: Como se revisó en la literatura presentada, resulta pertinente generar actividad de fondo para simular escenarios realistas. En sistemas reales, los artefactos de evidencia se encuentran mezclados con actividad de fondo pre-existente.

  Ejemplo: En un escenario donde se encuentran artefactos de evidencia, además del ataque debe existir actividad de fondo de un usuario realizando tareas cotidianas (navegación web, edición de documentos), porque sin esta actividad descubrir la evidencia sería trivial y el ejercicio carecería de valor pedagógico.

Criterio de verificación: El sistema debe ser capaz de producir un volumen de actividad de fondo que simule un sistema en uso cotidiano durante un período de tiempo.

Prioridad: Alta. La capacidad de distinguir actividad maliciosa de actividad legítima es uno de los pilares del análisis forense real.

- **Control de marcas de tiempo**: Al desprenderse del análisis que las marcas de tiempo son atributos intrínsecos de cada uno de los artefactos, se debe poder gestionar la cronología, permitiendo modificar y adaptar las líneas temporales de los artefactos creados.

  Ejemplo: El atacante primero escanea la red, luego explota una vulnerabilidad, después escala privilegios, finalmente exfiltra datos. Los timestamps de los artefactos deben reflejar esta secuencia: no puede ocurrir que archivos exfiltrados tengan timestamps anteriores a la explotación de la vulnerabilidad que permitió el acceso.

  Criterio de verificación: En cualquier escenario generado, el orden cronológico de los timestamps de todos los artefactos debe ser consistente con la secuencia de eventos especificada en la configuración del escenario. La verificación se realiza mediante análisis de línea de tiempo, construyendo una vista unificada y ordenada cronológicamente de los artefactos generados y comprobando que ningún evento contradiga la secuencia causal definida.

  Prioridad: Alta, dado que inconsistencias temporales son detectables por cualquier herramienta forense estándar y comprometen directamente la credibilidad del escenario.

- **Transversalidad**: Los ataques reales dejan rastros en múltiples capas del sistema. Un análisis forense completo requiere correlacionar evidencia de red, sistema de archivos y memoria. El sistema debe poder entrelazar artefactos coherentes entre estas tres dimensiones.

  Ejemplo: Un ataque podría dejar credenciales en un volcado de memoria, una clave de cifrado en una captura de red y archivos cifrados en una imagen de disco, requiriendo que el estudiante correlacione las tres fuentes para recuperar el contenido de un archivo eliminado.

  Criterio de verificación: El sistema debe ser capaz de generar al menos un escenario que requiera correlacionar simultáneamente artefactos de red (captura de red), sistema de archivos (imagen de disco) y memoria (volcado de proceso), donde ninguna fuente individual sea suficiente para resolver el caso completo.

  Prioridad: Alta, porque el análisis forense real requiere correlación entre múltiples fuentes de evidencia.

- **Escalabilidad**: Los laboratorios prácticos de forensia digital requieren que múltiples estudiantes trabajen simultáneamente, cada uno con su propio entorno aislado. El sistema debe permitir desplegar múltiples instancias

de un escenario a partir de una única especificación, sin que el esfuerzo del instructor crezca proporcionalmente al número de estudiantes.

Ejemplo: un instructor define un escenario una sola vez y lo despliega simultáneamente para 30 estudiantes, cada uno con su propia instancia aislada, sin necesidad de repetir ni adaptar manualmente la configuración para cada uno.

Criterio de verificación: Deben poder generarse n instancias de un escenario a partir de una única especificación.

Prioridad: Alta, porque sin esta capacidad el sistema no puede utilizarse en contextos educativos reales, donde se trabaja con múltiples estudiantes en simultáneo.

- **Trazabilidad**: Para validar hallazgos y facilitar la corrección, es esencial rastrear qué artefactos fueron generados, y dónde se ubican. Esto permite que instructores verifiquen si un estudiante omitió evidencia o si identificó falsos positivos.

  Ejemplo: En un escenario con artefactos de evidencia, un estudiante reporta haber resuelto la práctica, habiendo encontrado un valor concluyente dentro de un archivo cifrado, pero el contenido descifrado es incorrecto. El instructor verifica mediante trazabilidad que el archivo se generó correctamente, fue cifrado con una clave específica, y posteriormente eliminado dejando el contenido recuperable en el sistema de archivos. Esto confirma que el error fue del estudiante al descifrar, no un defecto en la generación del artefacto.

  Criterio de verificación: El sistema debe generar un registro que documente cada artefacto producido, incluyendo tipo, ubicación, valor esperado y relaciones con otros artefactos. Un instructor debe poder determinar, consultando únicamente este registro, si cualquier artefacto específico del escenario fue correctamente generado.

  Prioridad: Alta, porque permite a los instructores validar respuestas de estudiantes y depurar errores en la generación de evidencia.

- **Reproducibilidad**: El sistema debe ser capaz de recrear exactamente un escenario previamente generado a partir de una misma especificación.

  Ejemplo: Un estudiante presenta un resultado inesperado durante la evaluación. El instructor regenera el escenario exacto y verifica si el problema fue de generación o de análisis.

  Criterio de verificación: Ejecutar el sistema dos veces con la misma configuración debe producir artefactos forenses idénticos.

  Prioridad: Alta, porque sin reproducibilidad es imposible verificar errores reportados o corroborar soluciones en evaluaciones.

- **Compatibilidad con Tectonic**: Tectonic define escenarios mediante archivos YAML que describen topologías de red, máquinas virtuales y confi-

guraciones de software. La solución de generación de evidencia debe integrarse nativamente en este flujo declarativo, permitiendo que instructores especifiquen artefactos forenses como parte de la definición del escenario sin requerir scripts externos o pasos manuales post-despliegue.

Ejemplo: un instructor define en el YAML del escenario que una máquina debe contener evidencia de ejecución de comandos sospechosos, y Tectonic genera los artefactos correspondientes al momento del despliegue.

Criterio de verificación: Se verifica confirmando que la solución se integra como una extensión nativa del mecanismo de despliegue de Tectonic, sin modificar su flujo principal. La verificación consiste en que, al ejecutar un despliegue estándar de Tectonic, las máquinas virtuales se aprovisionan correctamente con los artefactos forenses especificados en la definición del escenario, sin intervención manual posterior.

Prioridad: Alta, porque este proyecto busca extender las capacidades de Tectonic.

- **Generación de contenido realista**: Los artefactos generados deben contener contenido internamente coherente y verosímil. No basta con que un archivo tenga la estructura técnica correcta: su contenido debe simular actividad humana auténtica, con nombres, textos, historiales y datos que sean plausibles en el contexto del perfil de usuario simulado.

  Ejemplo: el historial de navegación de un usuario de perfil administrativo debería contener sitios coherentes con esa actividad (portales institucionales, herramientas de gestión), y no URLs generadas aleatoriamente que resulten inverosímiles al análisis.

  Criterio de verificación: Un analista externo, al examinar los artefactos generados sin conocer el sistema, no debe poder distinguirlos de artefactos producidos por actividad humana real.

  Prioridad: Media, porque los artefactos con estructura técnica correcta son suficientes para practicar las técnicas forenses fundamentales, aunque contenido inverosímil puede reducir el realismo del escenario.

Requerimiento	**Tipo**	**Origen**	**Prioridad**
Generación sintética de artefactos forenses	Funcional	Análisis de antecedentes	Alta
Generación de actividad de fondo	Funcional	Literatura sobre realismo	Alta
Control de marcas de tiempo	Funcional	Propiedades transversales	Alta
Transversalidad	Funcional	Naturaleza multidominio del análisis forense	Alta
Trazabilidad	Funcional	Necesidad pedagógica	Alta
Reproducibilidad	Funcional	Requisitos académicos	Alta
Escalabilidad	No funcional	Contexto educativo multiusuario	Alta
Compatibilidad con Tectonic	No funcional	Integración con plataforma base	Alta
Generación de contenido realista	Funcional	Realismo pedagógico	Media

Tabla 3.1: Requerimientos derivados del análisis, clasificados por tipo, origen y prioridad.

Capítulo 4

Diseño de la Solución

El presente capítulo expone el diseño de la arquitectura propuesta para abordar los tres ejes principales del proyecto: artefactos de red, sistema de archivos y memoria. Se define una solución técnica orientada a facilitar la generación de escenarios forenses realistas para los instructores, basándose en un esquema de abstracción por niveles que permite simplificar la creación de artefactos complejos y garantiza poder crear artefactos de forma tan granular como se desee.

4.1. Derivación de Principios de Diseño

El análisis presentado en el capítulo anterior permitió identificar una serie de características estructurales del problema que condicionan de manera directa la forma en que se diseña la solución.

En primer lugar, es necesario permitir la especificación detallada de los atributos deseados de cada artefacto curado, y por otro, debe posibilitarse la construcción de escenarios complejos sin que el nivel de detalle requerido vuelva inviable su definición. Esto implica la necesidad de distintos niveles de abstracción, para poder operar con precisión como también con menor detalle.

En segundo lugar, se observó que muchos artefactos complejos pueden entenderse como combinaciones o estructuras derivadas de artefactos más básicos. Esta propiedad composicional sugiere que la solución debe permitir construir estructuras de mayor complejidad a partir de unidades más elementales.

En tercer lugar, se identificó la existencia de propiedades transversales compartidas entre artefactos de distintas vertientes, como la temporalidad. Estas propiedades no corresponden exclusivamente a un tipo particular de artefacto, sino que atraviesan múltiples dominios. En consecuencia, la solución no puede tratar cada artefacto como una entidad completamente aislada, sino que debe contemplar mecanismos que garanticen consistencia estructural entre ellos.

A partir de estas observaciones, se desprende que la arquitectura de la solución debe estar orientada a la composición de artefactos, la reutilización de

lógica común, la preservación de coherencia transversal y la separación de responsabilidades.

En este contexto, una arquitectura en capas surge como una consecuencia natural del análisis realizado. La misma permite:

- Diferenciar entre artefactos elementales y estructuras compuestas.
- Centralizar propiedades transversales compartidas.
- Ofrecer distintos niveles de abstracción según el grado de control requerido.
- Facilitar la extensibilidad sin comprometer la coherencia global del sistema.

Bajo este marco conceptual, se propone a continuación una arquitectura organizada en niveles de abstracción claramente delimitados, cuyo diseño responde directamente a las conclusiones obtenidas en el análisis previo.

4.2. Arquitectura General

Cuando los instructores generen un escenario de tipo DFIR, necesitarán producir diversos artefactos forenses. Para ello, resulta necesario contar con un mecanismo que permita especificar de forma clara qué artefactos deben generarse y bajo qué condiciones. Es por esto que la solución propuesta puede entenderse conceptualmente como la interacción entre dos componentes principales: un lenguaje de especificación y un sistema de generación de artefactos.

El lenguaje de especificación constituye el mecanismo mediante el cual el instructor describe el escenario deseado. A través de una sintaxis sencilla, el instructor puede indicar qué artefactos forenses deben generarse y qué características deben poseer. La idea es que el instructor pueda expresar *qué* artefactos desea generar sin necesidad de detallar cómo deben construirse internamente.

Por su parte, el sistema de generación es el componente encargado de materializar los artefactos especificados. Este sistema expone un conjunto de primitivas que representan operaciones de generación de artefactos forenses. Dichas primitivas constituyen la interfaz que el lenguaje de especificación utiliza para invocar las capacidades del sistema. De esta forma, el lenguaje funciona como un mecanismo de orquestación de las primitivas disponibles.

Bajo este modelo, el instructor únicamente debe describir el escenario utilizando el lenguaje de especificación, mientras que el sistema se encarga de ejecutar las primitivas necesarias para producir los artefactos correspondientes.

Para el sistema de generación de los artefactos, se ha diseñado una arquitectura basada en un modelo de tres capas de abstracción (Capa 1 a Capa 3), complementado por una capa transversal (Capa 0) que gestiona la generación sintética de datos. Cada capa encapsula su propia complejidad interna y expone primitivas bien definidas que pueden ser utilizadas tanto por las capas superiores como directamente desde el lenguaje de especificación. De esta forma, cada primitiva disponible en el sistema no solo es una función de la jerarquización en

capas del sistema, sino que constituye también una operación accesible desde el lenguaje utilizado por el instructor.

Esta arquitectura permite generar artefactos complejos mediante una única llamada a primitivas con un nivel de abstracción adecuado, mientras mantiene la capacidad de controlar detalles técnicos cuando se requiera precisión específica. Se puede ver un diagrama de la arquitectura en la figura 4.1.

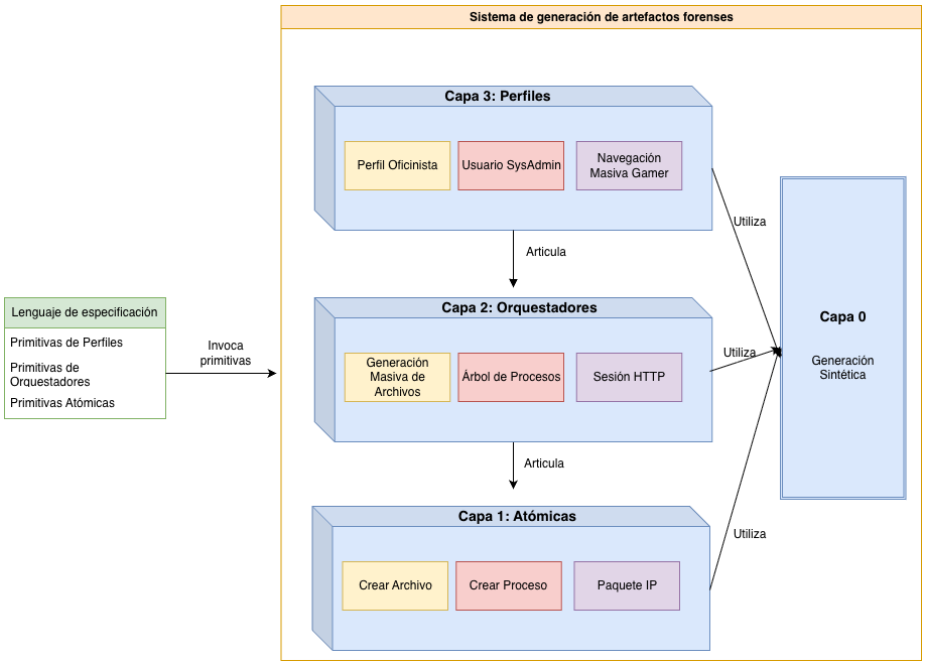


Figura 4.1: Diagrama de arquitectura. El color amarillo representa la vertical de sistema de archivos. El color rojo representa la vertical de memoria. El color violeta representa la vertical de red.

La decisión de estructurar la solución en tres capas responde a la identificación de tres niveles conceptualmente distintos de construcción de artefactos.

En primer lugar, el análisis identificó la existencia de artefactos atómicos, entendidos como unidades mínimas desde el punto de vista del modelado forense (un archivo individual, un paquete de red, un proceso en ejecución). Estos elementos poseen atributos propios y pueden ser definidos con precisión técnica. De esta observación se desprende la necesidad de una capa dedicada exclusivamente a la generación controlada de dichas unidades elementales que permita profundizar en un artefacto de forma tan detallada como se desee. En el caso del sistema de archivos, por ejemplo, esto implica poder especificar atributos como nombre, ubicación o fecha de creación, tal como ilustra la Figura 4.2.

En segundo lugar, se observó que el valor forense raramente reside en artefactos aislados, sino en conjuntos estructurados de artefactos cuya relación temporal y semántica construye significado. Una sesión de red, una descarga de

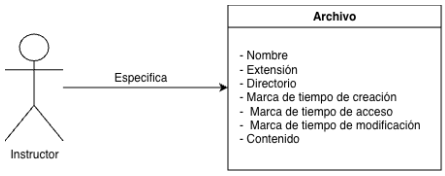


Figura 4.2: Diagrama de especificación de metadatos y contenido de artefacto forense en sistema de archivos

archivos o una actividad administrativa son ejemplos de estructuras compuestas por múltiples unidades atómicas. Esta propiedad composicional fundamenta la existencia de una capa intermedia responsable de la orquestación coherente de múltiples artefactos atómicos.

En tercer lugar, el análisis destacó la importancia del contexto operativo y de la actividad de fondo. Los escenarios realistas requieren no solo eventos puntuales, sino también actividad adicional que represente el uso normal del sistema. Esta actividad de fondo otorga verosimilitud al escenario y dificulta la identificación trivial del artefacto de interés. En este nivel no se busca modelar un evento aislado, sino el comportamiento del sistema a lo largo de períodos prolongados de tiempo. A partir de esta necesidad surge la capa superior del modelo, orientada a la generación de contextos completos donde múltiples actividades coexisten y evolucionan en el tiempo.

Finalmente, el análisis estableció una dimensión transversal, la cual no pertenece a un tipo particular de artefacto, sino que atraviesa todos los niveles del sistema. De aquí se desprende la incorporación de una capa común dedicada a la provisión de datos comunes a todos los enfoques. Si bien la arquitectura propuesta comprende cuatro capas, este último nivel no se enmarca dentro del modelo clásico de la arquitectura en capas, dado que puede ser consumida desde cualquiera de los niveles del sistema y no tiene como responsabilidad la generación de artefactos, función que sí es propia de las demás capas. Por este motivo, y a fin de distinguirla conceptualmente del resto, se la denomina Capa 0.

En consecuencia, las tres capas identificadas más la capa 0, representan la materialización estructural de los conceptos del análisis: *artefacto atómico, artefacto compuesto, contexto operativo e información transversal*.

4.2.1. Capa de Uso Común (Capa 0)

La Capa 0 proporciona primitivas para generar datos sintéticos realistas que pueblan automáticamente los artefactos forenses. Como se identificó en el análisis, no solamente la temporalidad es un atributo compartido entre los artefactos, sino también los identificadores, nombres y ubicaciones. Su función es producir contenido válido para los diferentes tipos de artefactos forenses sin requerir que el instructor especifique manualmente cada detalle. Las primitivas de Capa 0 generan:

- **Identidades y datos personales**: Nombres completos y nombres de em-

presas, direcciones de correo electrónico, números de teléfono, direcciones postales.

- **Direcciones de red**: Direcciones IP públicas y privadas, direcciones MAC, nombres de dominio, URLs.
- **Contenido textual**: Párrafos de texto pseudoaleatorio, oraciones, palabras, títulos.
- **Datos técnicos de red**: Agentes de usuario de navegadores reales, tipos MIME, puertos, números de secuencia.

Estas primitivas son invocadas por las capas superiores cuando necesitan poblar campos no especificados por el instructor. Por ejemplo, cuando la Capa 2 genera una sesión HTTP de navegación web, invoca primitivas de Capa 0 para generar automáticamente un agente de usuario y nombres de dominio sintéticos, permitiendo que el instructor especifique solo *generar navegación web a n sitios*. El instructor por lo tanto solo se dedica a especificar lo pedagógicamente relevante, mientras los artefactos generados pseudoaleatoriamente son técnicamente válidos.

Las primitivas de generación pseudoaleatoria introducidas en esta capa no se limitan exclusivamente a los escenarios DFIR. Al proveer la capacidad de generar datos sintéticos, esta capa puede ser reutilizada en otros tipos de laboratorios dentro de la plataforma Tectonic, evitando el uso repetitivo de valores estáticos y habilitando la incorporación de datos dinámicos en escenarios de otras áreas.

4.2.2. Capa Atómica (Capa 1)

La Capa 1 constituye el nivel fundamental de generación de artefactos dentro de la arquitectura. Su responsabilidad es proveer primitivas capaces de materializar artefactos forenses individuales, entendidos como unidades mínimas desde el punto de vista del modelado. Ejemplos de este tipo de artefactos incluyen un único archivo en el sistema de archivos, la generación de un paquete de red o la creación de un proceso en memoria.

Las primitivas de esta capa permiten controlar de forma explícita los atributos técnicos de cada artefacto. El objetivo de esta capa es ofrecer al sistema el máximo nivel de control técnico sobre los artefactos generados. Cada primitiva representa una operación elemental que puede ser invocada directamente desde el lenguaje de especificación o utilizada por capas superiores como bloque de construcción. De esta forma, la Capa 1 actúa como el conjunto de bloques básicos sobre los cuales se construyen las abstracciones de mayor nivel presentes en las capas superiores.

4.2.3. Capa de Orquestadores (Capa 2)

La Capa 2 introduce un nivel adicional de abstracción encargado de coordinar múltiples primitivas de la Capa 1 para generar secuencias de actividad

coherentes. Mientras que la capa atómica se enfoca en artefactos individuales, esta capa permite modelar interacciones o procesos compuestos que involucran varios artefactos relacionados.

Las primitivas de esta capa encapsulan patrones recurrentes de generación de evidencia. Por ejemplo, una sesión de navegación web puede requerir múltiples paquetes de red, resoluciones DNS previas y transferencias HTTP asociadas.

En lugar de requerir que el instructor describa manualmente cada uno de estos pasos, los orquestadores encapsulan dicha lógica en una única primitiva de mayor nivel. Internamente, estas primitivas invocan múltiples operaciones de la Capa 1, coordinando su ejecución y garantizando la coherencia temporal y semántica entre los artefactos producidos.

Este nivel de abstracción permite describir comportamientos más complejos con una menor cantidad de instrucciones en el lenguaje de especificación, reduciendo la carga operativa del instructor.

Al igual que en la capa anterior, cuando ciertos parámetros no son especificados explícitamente, los orquestadores pueden recurrir a las primitivas de la Capa 0 para completar información necesaria para la generación de los artefactos involucrados.

4.2.4. Capa de Perfiles (Capa 3)

La Capa 3 representa el nivel de abstracción más alto de la arquitectura. Su función principal es permitir al instructor definir y reutilizar comportamientos complejos de interés dentro de los escenarios, pudiendo invocarlos de forma individual o combinada. Por ejemplo, es posible invocar simultáneamente un perfil de *procesos de Administrador de Sistemas* y un perfil de *sistema de archivos de Administrador de Sistemas*, obteniendo como resultado un entorno donde el disco contiene documentos y estructuras clásicas de un administrador de sistemas, mientras la memoria posee un conjunto de artefactos de procesos asociados con dicha actividad. Actúa como un coordinador de alto nivel que puede invocar tanto orquestadores de la Capa 2 como primitivas atómicas de la Capa 1 para construir un caso de uso integral.

En la práctica, los escenarios definidos por los instructores suelen apoyarse principalmente en esta capa para generar grandes volúmenes de actividad de forma compacta. En lugar de especificar manualmente múltiples operaciones de bajo nivel, el instructor puede invocar perfiles que encapsulan dicha lógica, simplificando considerablemente la construcción del escenario. Por ejemplo, si se desea construir un escenario que contenga un archivo con una bandera dentro de un sistema de archivos correspondiente al entorno de trabajo de un desarrollador, el instructor podría invocar un perfil encargado de generar un sistema de archivos típico de este tipo de usuario. Dicho perfil podría crear automáticamente una colección de archivos representativos, como código fuente en Python o JavaScript, archivos de configuración y algunos documentos de uso cotidiano. Posteriormente, el instructor puede complementar este entorno invocando directamente una primitiva de la Capa 1 para crear el archivo específico que contiene la bandera dentro del sistema de archivos generado.

En aquellos casos donde un patrón de uso resulte recurrente, es posible encapsular completamente dicha lógica en un nuevo perfil. Siguiendo con el ejemplo anterior, podría definirse un perfil denominado *developer_with_flag*, el cual reciba como parámetro la ubicación de la bandera y opcionalmente su contenido. Si el contenido no es especificado, este puede generarse automáticamente utilizando las primitivas de la Capa 0. Internamente, este perfil primero generaría el conjunto de archivos característicos de un entorno de desarrollo y posteriormente invocaría la primitiva atómica correspondiente para crear el archivo que contiene la bandera en la ubicación indicada.

Este enfoque permite definir tantos perfiles como resulten necesarios según las necesidades pedagógicas. Algunos perfiles pueden dedicarse exclusivamente a generar actividad de fondo realista, mientras que otros pueden combinar dicha actividad con artefactos específicos relevantes para el ejercicio. Al encapsular estos comportamientos en perfiles reutilizables, se facilita la creación de laboratorios más complejos sin incrementar la complejidad del lenguaje de especificación utilizado por los instructores.

Desde el punto de vista de uso, los perfiles se comportan como cualquier otra primitiva del sistema: se invocan desde el lenguaje de especificación mediante su nombre y los parámetros correspondientes.

De esta manera, la Capa 3 se posiciona como el punto de interacción principal entre el instructor y el sistema de generación de artefactos, permitiendo describir escenarios complejos mediante la composición de comportamientos previamente definidos.

Cabe destacar que todas las primitivas de la Capa 3 reciben parámetros transversales para el manejo de la temporalidad, específicamente el *timestamp_base* y la función de distribución, tal como se detalla en la sección 4.3.1 de coherencia temporal.

4.3. Diseño de las Propiedades Transversales de los Artefactos

Como se identificó durante el análisis, los artefactos forenses comparten un conjunto de propiedades transversales que aparecen de forma consistente en distintos tipos de evidencia. Entre ellas se encuentran, por ejemplo, las marcas temporales, ciertos identificadores y distintos tipos de contenido asociados a los artefactos.

Con el objetivo de mantener un diseño modular y favorecer la reutilización, estas propiedades se modelan de forma independiente de las primitivas específicas de cada subsistema. De esta manera, las distintas capas del sistema pueden utilizar los mismos mecanismos para gestionar estos atributos sin duplicar lógica de implementación.

4.3.1. Coherencia Temporal

En el análisis se observó que todos los artefactos forenses poseen marcas de tiempo, además dichas marcas deben mantener coherencia lógica (por ejemplo, un archivo no puede modificarse antes de haber sido creado, ni una respuesta DNS puede preceder a su consulta).

Un primer enfoque considerado fue permitir que todas las primitivas recibieran un único parámetro *timestamp*. Sin embargo, esta solución resulta insuficiente para primitivas de capas superiores (Capa 2 y Capa 3), que internamente orquestan múltiples llamadas a primitivas de nivel inferior. En esos casos surge la pregunta: ¿con qué valor de tiempo deberían invocarse cada uno de los artefactos internos?

Para resolver esta limitación, se diseñó un esquema basado en dos parámetros. Un primer parámetro basado en un timestamp base, que representa el instante inicial, y un segundo parámetro siendo una función de distribución temporal, que define cómo se espacian cronológicamente los eventos posteriores. De esta forma, las primitivas de mayor nivel utilizan estos parámetros para calcular automáticamente los instantes de cada evento interno, garantizando coherencia temporal sin que el instructor deba gestionar manualmente cada marca de tiempo. Mientras que cuando el instructor invoca una primitiva de Capa 1, debe especificar un timestamp exacto. En este nivel, la coherencia temporal queda bajo su responsabilidad.

De esta manera, el lenguaje permite un control preciso cuando se requiere granularidad máxima (Capa 1) y orquestación temporal automática en capas superiores.

En caso de que el instructor no especifique parámetros temporales, el sistema utiliza valores por defecto. Para las primitivas de Capa 1, el *timestamp* se establece automáticamente en el momento de ejecución. Para las capas superiores, se utiliza el momento de ejecución como valor de `timestamp_base`, una distribución uniforme como función de distribución temporal, y un `timestamp_final` generado pseudoaleatoriamente mediante la Capa 0.

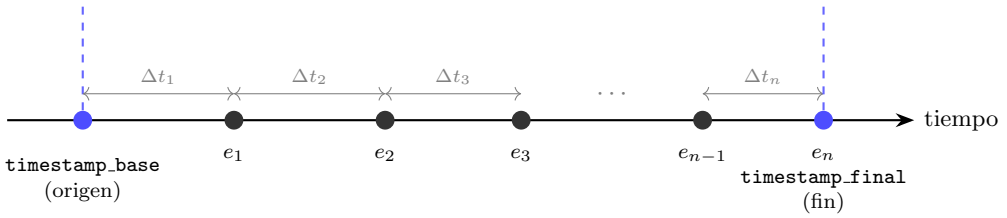


Figura 4.3: Esquema de coherencia temporal: `timestamp_base` y `timestamp_final` delimitan el intervalo de generación, y la función de distribución determina el espaciado Δt_i entre eventos sucesivos.

4.3.2. Reproducibilidad mediante Semillas

Para garantizar la reproducibilidad de los escenarios, se adoptó el uso de semillas (*seeds*) para controlar los procesos pseudoaleatorios, enfoque predominante en la literatura. Cada primitiva acepta un parámetro de semilla opcional, donde si se especifica, la ejecución es determinista.

Sin embargo, cuando una primitiva realiza múltiples llamadas pseudoaleatorias con la misma semilla (sucede con las primitivas de Capa 2 y Capa 3), los resultados generados serían idénticos entre sí. Para evitarlo, el sistema implementa un mecanismo de derivación de sub-semillas basado en índices, donde cada llamada interna utiliza el valor:

$$seed + index \tag{4.1}$$

garantizando así que cada elemento obtenga valores distintos dentro de una misma ejecución, preservando al mismo tiempo la reproducibilidad.

4.3.3. Contenidos

Los artefactos forenses no solo poseen atributos estructurales, sino también contenido que puede ser analizado durante una investigación forense.

La generación de estos contenidos introduce desafíos adicionales respecto a otros atributos de los artefactos. Dependiendo del tipo de artefacto, el contenido puede variar considerablemente en complejidad. En algunos casos puede tratarse simplemente de texto contenido en un archivo. En otros, el contenido debe respetar formatos y estructuras bien definidas. Por ejemplo, un archivo de tipo *syslog* debe seguir convenciones establecidas en los RFC correspondientes, mientras que en el ámbito de red los contenidos pueden variar desde comunicaciones en texto plano hasta documentos HTML que deben respetar la sintaxis y protocolos asociados.

Modelar de forma completa los posibles tipos de contenido escapa al alcance de este proyecto. Sin embargo, con el objetivo de demostrar la viabilidad del diseño de contenido, se diseñaron dos modelos representativos de contenido en el eje de sistema de archivos: correos electrónicos y registros de autenticación (*auth.log*).

Para ello se diseñó una arquitectura específica para la generación de contenidos, organizada en tres capas de abstracción de forma análoga al resto del sistema. Este esquema permite describir contenidos complejos mediante la composición de primitivas de distinto nivel. El diseño detallado de estas primitivas y su organización se presenta en el documento anexo `Diseño de Primitivas`.

4.4. Artefactos de Sistema de Archivos

En el eje de sistema de archivos, la unidad mínima de modelado es el archivo individual, compuesto por contenido y metadatos. Sin embargo, el valor forense rara vez reside en un único archivo aislado, sino en la relación temporal

y estructural entre múltiples artefactos dentro de una jerarquía de directorios. Por lo tanto, el diseño distingue entre operaciones elementales sobre archivos y mecanismos de generación masiva que permiten reproducir entornos completos con coherencia estructural.

Cabe destacar que por cuestiones de verbosidad no se detallan todas las primitivas. Sin embargo, la totalidad de las mismas con sus explicaciones se encuentran en el documento anexo `Diseño de Primitivas`.

4.4.1. Capa 1: Primitivas Atómicas de Archivo

Estas primitivas constituyen las operaciones fundamentales y atómicas sobre archivos individuales. Garantizan la precisión técnica de cada artefacto y sirven como base para las capas superiores.

La Tabla 4.1 describe los parámetros de la primitiva `generate_file`, que constituye la operación elemental de generación de artefactos de archivo con contenido y metadatos controlados.

Parámetro	Tipo	Descripción	Requerido
`path`	string	Ruta absoluta donde se crea el archivo. Debe incluir el nombre del archivo y su extensión.	Sí
`extension`	string	Tipo de archivo a generar. Valores soportados: `txt`, `pdf`, `jpg`, `png`, `docx`, `xlsx`, `zip`, `tar`, `tar.gz`, `sh`, `py`.	Sí
`content`	string	Contenido explícito del archivo. Por defecto se genera con Capa 0. El contenido se genera en el formato acorde al formato implícito del archivo.	No
`size`	string	Tamaño objetivo del archivo (e.g. `10KB`, `2MB`). Aplicable a archivos de texto y PDF. Por defecto: determinado por el contenido generado.	No
`creation_date`	datetime	Marca de tiempo de creación del archivo. Por defecto: momento de ejecución.	No
`access_date`	datetime	Marca de tiempo de acceso del archivo. Por defecto: momento de ejecución.	No
`modification_date`	datetime	Marca de tiempo de modificación del archivo. Por defecto: momento de ejecución.	No
`permissions`	string	Permisos Unix en formato octal (e.g. `0644`, `0755`). Por defecto: los establecidos en la creación.	No
`seed`	int	Semilla para garantizar reproducibilidad del contenido generado. Sin valor por defecto.	No

Tabla 4.1: Primitiva `generate_file`: generación de un archivo individual con contenido sintético y metadatos controlados.

Además de `generate_file`, se contemplan cuatro primitivas complementa-

rias para la manipulación de artefactos existentes. La primitiva `delete_file` elimina un artefacto del sistema de archivos. La primitiva `modify_permissions` altera los permisos de un artefacto, recibiendo la ruta, los permisos deseados y una marca de tiempo. La primitiva `encrypt_file` simula el cifrado de un artefacto a partir de la ruta, el algoritmo de cifrado y la extensión que identifica el tipo de cifrado aplicado, si corresponde se proporciona también la clave de encriptado y opcionalmente una semilla para garantizar reproducibilidad. El hecho de contar con la primitiva `encrypt_file` motiva la existencia de `decrypt_file`, que permite recuperar el artefacto con sus metadatos y contenido cargados al momento de creación.

4.4.2. Capa 2: Orquestadores de Sistema de Archivos

La Tabla 4.2 describe los parámetros del orquestador `generate_files_bulk`, que coordina la generación masiva de archivos.

Parámetro	Tipo	Descripción	Requerido
`count`	int	Cantidad total de archivos a generar.	Sí
`base_directory`	string	Directorio raíz donde se generan los archivos.	Sí
`distribution`	dict	Distribución de tipos de archivo con porcentajes que deben sumar 100. Por defecto: {`pdf: 30, docx: 40, txt: 30`}.	No
`structure`	string	Tipo de estructura de directorios a generar. Valores: `flat` o `tree`. Por defecto: `flat`.	No
`name_pattern`	string	Plantilla para los nombres de archivo. Soporta strings y variables. Por defecto: `document_{n}`.	No
`tree_depth`	int	Profundidad máxima del árbol de directorios. Aplicable únicamente con `structure=tree`. Por defecto: Cantidad total de archivos.	No
`subdirs_per_level`	int	Cantidad de subdirectorios por nivel en estructura jerárquica. Por defecto: 1.	No
`seed`	int	Semilla para garantizar reproducibilidad de nombres y contenido generado. Sin valor por defecto.	No

Tabla 4.2: Orquestador `generate_files_bulk`: generación masiva de archivos con distribución de tipos y estructura de directorios configurable.

El parámetro `distribution` recibe un diccionario donde cada clave es una extensión de archivo y su valor es el porcentaje deseado. Los porcentajes se normalizan automáticamente si no suman exactamente 100, y el remanente de la división entera se asigna al último tipo declarado, garantizando que el total de archivos generados sea exactamente `count`.

El parámetro `structure` controla la organización del sistema de directorios. Con `flat`, todos los archivos se generan en `base_directory` sin subdirectorios.

Con `tree`, se construye una jerarquía de directorios cuya profundidad máxima se controla mediante `tree_depth` y cuya ramificación por nivel se define con `subdirs_per_level`; por ejemplo, con `tree_depth=3` y `subdirs_per_level=2` se genera un árbol de hasta $2^3 = 8$ hojas, y los archivos se distribuyen entre todos los nodos del árbol aleatoriamente.

El parámetro `name_pattern` acepta una plantilla con variables que se expanden por archivo. Las variables básicas son {`n`} (índice del archivo), {`date`}, {`time`} y {`file_type`}. Cada archivo recibe valores únicos derivados de la semilla y el índice del archivo, garantizando que nombres distintos sean reproducibles. Por ejemplo, el patrón `document_{n}` aplicado sobre tres archivos produce `document_1.pdf`, `document_2.pdf` y `document_3.pdf`.

Además de `generate_files_bulk`, la Capa 2 contempla cuatro orquestadores complementarios para la manipulación masiva de artefactos existentes. El orquestador `shared_features_bulk` aplica características comunes a un conjunto de archivos filtrado por patrón de nombre, permitiendo modificar permisos, propietarios, tamaños y eliminar un porcentaje o cantidad exacta de archivos de forma simultánea. El orquestador `apply_temporal_distribution` modifica las marcas de tiempo de acceso y modificación de un conjunto de archivos según una distribución estadística configurable, soportando los modelos uniforme, gaussiano y exponencial, en función de las acciones temporales que se busquen simular. Por último, `encrypt_files` y `decrypt_files` coordinan el cifrado y descifrado masivo de archivos, recibiendo una lista de rutas, la clave de cifrado si es necesario, y la extensión que identifica los archivos cifrados.

4.4.3. Capa 3: Perfiles de Sistema de Archivos

Los perfiles de Capa 3 representan patrones de uso completo del sistema de archivos por parte de un tipo de usuario del sistema, ya sea legítimo o malicioso. Cada perfil orquesta operaciones de Capa 2 y Capa 1 para reproducir el entorno de archivos característico del rol que modela.

Por ejemplo, si el instructor desea que el artefacto forense de interés sea un correo de *phishing* que permitió la exfiltración de credenciales y comprometió la infraestructura de una organización, es necesario brindar un contexto que sitúe dicho artefacto dentro de un entorno realista. En este caso se podría crear un perfil de *Oficinista* que genere actividad de fondo característica de este tipo de entorno, como documentos Word, planillas de cálculo y otros archivos de uso cotidiano.

Este perfil puede especificar opcionalmente la cantidad de archivos generados mediante el parámetro `file_count`, que por defecto produce una cantidad pseudoaleatoria de archivos entre 100 y 500 utilizando la primitiva `generate_files_bulk`. Es posible definir tantos perfiles de actividad de fondo como se desee utilizando las primitivas mencionadas previamente.

Otro ejemplo es el Perfil de Ransomware, que coordina la generación de artefactos víctima, su cifrado y la eliminación de los originales, dejando tras de sí rastros de cifrado masivo y notas de rescate características del propio ataque.

4.5. Artefactos de Red

El análisis forense de tráfico de red rara vez se limita a la observación de paquetes individuales de forma aislada, sino que generalmente requiere la reconstrucción de flujos completos de comunicación y la correlación temporal entre múltiples intercambios. Interacciones como sesiones HTTP, autenticaciones remotas o transferencias de archivos se materializan como secuencias de numerosos paquetes que deben mantener coherencia protocolar y temporal.

No obstante, en muchos escenarios el evento relevante puede concentrarse en un único paquete específico, como aquel que descarga un artefacto malicioso o aquel que transporta una instrucción maliciosa. Por lo tanto, el diseño de este eje distingue entre primitivas capaces de generar paquetes individuales con control detallado de sus campos y mecanismos de nivel superior que permiten orquestar secuencias protocolarias completas.

La presente sección detalla las principales primitivas por cada capa en el eje de red. La totalidad de las mismas con sus explicaciones se encuentran en el documento anexo `Diseño de Primitivas`.

4.5.1. Capa 1: Primitivas Atómicas de Red

Estas primitivas constituyen las operaciones fundamentales y atómicas sobre artefactos de red individuales. Garantizan la validez técnica de cada paquete generado y sirven como base para las capas superiores, donde múltiples intercambios se combinan para formar sesiones o comportamientos de red más complejos.

La Tabla 4.3 describe los parámetros de la primitiva `create_packet`, que constituye la operación elemental para la construcción de un paquete individual basado en Ethernet e IP.

Esta primitiva ofrece control completo sobre los campos de cada paquete y resulta útil para generar tráfico específico o modelar situaciones atípicas. Sin embargo, la mayoría de los artefactos de red relevantes siguen protocolos que imponen reglas adicionales sobre la secuencia de paquetes, como los números de secuencia o los identificadores de transacción.

Si únicamente se dispusiera de esta primitiva, la lógica necesaria para mantener dichas reglas debería implementarse en las capas superiores o directamente por el instructor. Por ejemplo, simular una comunicación TCP válida requeriría gestionar manualmente el establecimiento de la conexión, la actualización de los números de secuencia y reconocimiento, y el cierre correcto del flujo.

Para simplificar esta tarea se diseñaron módulos especializados para algunos protocolos comunes. Un módulo no representa un nuevo tipo de operación, sino simplemente un agrupamiento conceptual de primitivas relacionadas entre sí según el protocolo o funcionalidad que implementan. Todas las primitivas continúan siendo independientes entre sí y pueden ser invocadas directamente tanto por los orquestadores de capas superiores como por el instructor. El uso correcto de estas primitivas depende de la lógica que las invoca; por ejemplo, nada impide que un instructor invoque dos veces consecutivas una primitiva de

Parámetro	Tipo	Descripción	Requerido
`MAC_src`	string	Dirección MAC de origen del paquete Ethernet. Por defecto se genera con Capa 0.	No
`MAC_dst`	string	Dirección MAC de destino del paquete Ethernet. Por defecto se genera con Capa 0.	No
`ip_src`	string	Dirección IP de origen del paquete. Por defecto se genera con Capa 0.	No
`ip_dst`	string	Dirección IP de destino del paquete. Por defecto se genera con Capa 0.	No
`port_src`	int	Puerto de origen. Por defecto se genera con Capa 0.	No
`port_dst`	int	Puerto de destino. Por defecto 80	No
`protocol`	string	Protocolo de transporte utilizado (`TCP` o `UDP`). Por defecto `TCP`.	No
`flags`	string	Banderas del protocolo de transporte (por ejemplo `SYN`, `ACK`, `FIN`, `RST`). Por defecto vacío.	No
`seq`	int	Número de secuencia del protocolo TCP cuando corresponda.	No
`ack`	int	Número de reconocimiento del protocolo TCP cuando corresponda.	No
`payload`	string	Carga útil del paquete. Por defecto se genera con Capa 0.	No
`ttl`	int	Valor de *time-to-live* del paquete IP. Por defecto: 64	No
`timestamp`	datetime	Marca de tiempo asociada al envío del paquete. Por defect: momento de ejecución	No
`seed`	int	Semilla para garantizar reproducibilidad en la generación de campos pseudoaleatorios. Sin valor por defecto.	No

Tabla 4.3: Primitiva `create_packet`: generación de un paquete individual con control sobre los campos de Ethernet, IP y transporte.

cierre de conexión, aún cuando ello no represente un comportamiento del protocolo válido. Estos módulos encapsulan la lógica necesaria para generar flujos válidos, utilizando internamente la primitiva `create_packet` para la creación de los paquetes individuales.

Uno de los casos más relevantes es la generación de comunicaciones TCP válidas. En estos escenarios, los paquetes deben mantener coherencia entre los números de secuencia (*seq*) y reconocimiento (*ack*) a lo largo de todo el flujo.

Para resolver esta complejidad se diseñó un motor de estados capaz de generar flujos TCP válidos siguiendo el comportamiento definido en el RFC 793 (Postel, 1981). Este módulo expone primitivas que permiten abrir una conexión, insertar paquetes dentro del flujo y finalmente cerrarla.

La Tabla 4.4 describe la primitiva `start_conn`, encargada de generar el es-

tablecimiento de una conexión TCP válida mediante el proceso de *three-way handshake*.

Parámetro	Tipo	Descripción	Requerido
`ip_src`	string	Dirección IP del cliente que inicia la conexión. Por defecto: se genera con Capa 0.	No
`ip_dst`	string	Dirección IP del servidor destino. Por defecto: se genera con Capa 0.	No
`port_dst`	int	Puerto del servicio al que se conecta el cliente. Por defecto: puerto 80.	No
`port_src`	int	Puerto origen del cliente. Por defecto: se genera con Capa 0.	No
`initial_seq_cli`	int	Número inicial de secuencia del cliente. Por defecto: se genera con Capa 0.	No
`initial_ack_srvr`	int	Número inicial de *acknowledge* del cliente. Por defecto: se genera con Capa 0.	No
`timestamp`	datetime	Marca temporal del inicio de la conexión. Por defecto: momento de ejecución.	No
`seed`	int	Semilla para reproducibilidad de valores generados automáticamente. Sin valor por defecto.	No

Tabla 4.4: Primitiva `start_conn`: establecimiento de una conexión TCP mediante el *three-way handshake*.

La primitiva `add_to_stream` permite insertar nuevos paquetes dentro del flujo manteniendo la continuidad de los números de secuencia y reconocimiento. Para ello recibe como parámetro el último paquete de la conexión, la dirección del mensaje (cliente a servidor o servidor a cliente) y el contenido de la carga útil.

Finalmente, la primitiva `close_conn` se encarga de finalizar la conexión respetando el protocolo TCP mediante secuencias `FIN/ACK`. Esta operación recibe el estado del último paquete transmitido como parámetro, para garantizar que el cierre del flujo sea coherente con la comunicación previamente generada.

En la Figura 4.4 se ilustra el uso típico de estas primitivas. En un escenario común, el instructor o los orquestadores de capas superiores invocan primero la primitiva de apertura de conexión, posteriormente agregan uno o más paquetes dentro del flujo, y finalmente ejecutan la primitiva de cierre. Esta secuencia garantiza la generación de una comunicación TCP técnicamente válida.

Además del módulo de gestión de conexiones TCP, se diseñó un módulo específico para modelar transacciones DNS respetando la lógica definida en el RFC 1035 (Mockapetris, 1987).

La primitiva `dns_query` permite generar paquetes de consulta DNS hacia un servidor específico. Entre sus parámetros principales se encuentran el dominio a resolver, el tipo de consulta, la dirección IP del cliente que realiza la consulta y la dirección del servidor DNS. El módulo genera automáticamente los campos internos del protocolo como el identificador de transacción cuando este no es provisto por elinstructor.

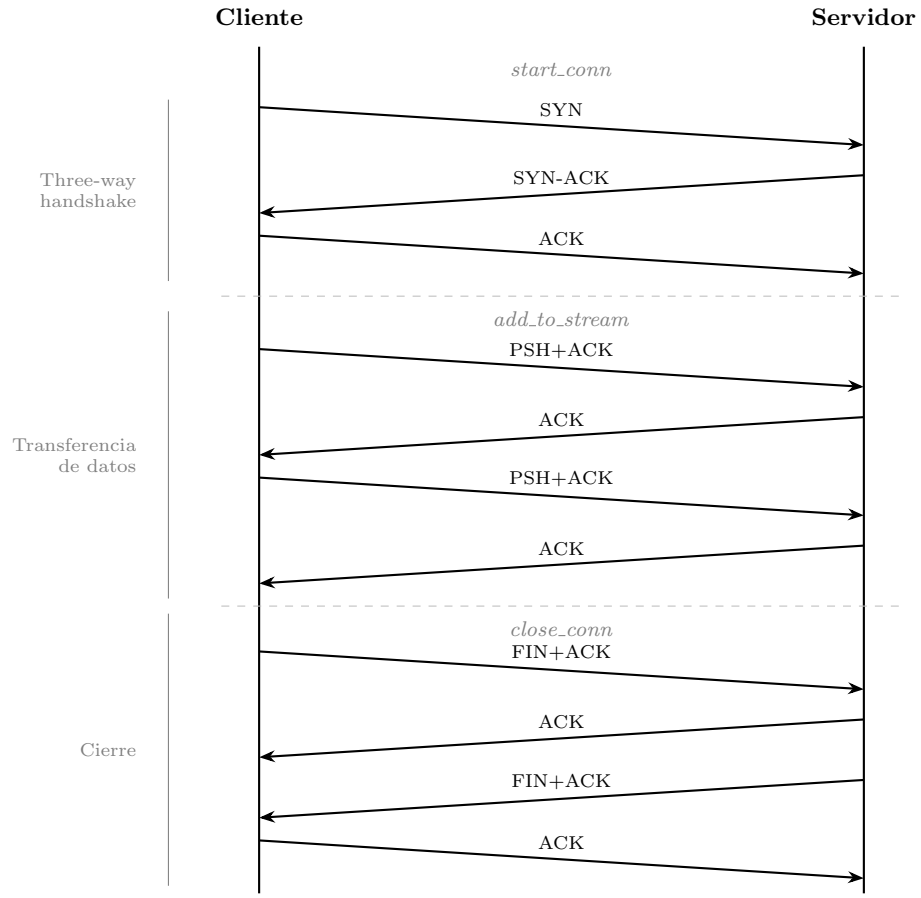


Figura 4.4: Diagrama de secuencia del módulo de gestión TCP: las primitivas `start_conn`, `add_to_stream` y `close_conn`.

Complementariamente, la primitiva `dns_resp` permite construir una respuesta coherente a una consulta previa. Esta primitiva recibe como parámetro el paquete de consulta original, lo que permite reutilizar automáticamente el identificador de transacción y garantizar la coherencia del intercambio. Entre los parámetros relevantes se encuentran la dirección IP de respuesta y el estado de la consulta (`NOERROR`, `NXDOMAIN` o `SERVFAIL`).

Estas primitivas permiten modelar consultas DNS individuales manteniendo la validez estructural de los paquetes generados, y sirven como base para los orquestadores de red de capas superiores que construyen conversaciones completas entre clientes y servidores.

4.5.2. Capa 2: Orquestadores

La Capa 2 introduce primitivas de mayor nivel que combinan múltiples operaciones atómicas para construir comportamientos de red completos. Estas primitivas abstraen la complejidad protocolar subyacente y permiten al instructor especificar actividades típicas de red mediante parámetros de alto nivel.

En lugar de generar manualmente cada paquete, el instructor puede definir una acción lógica —como una resolución DNS o una navegación web— y el sistema se encarga de generar la secuencia completa de paquetes necesaria para representar dicho comportamiento.

La primitiva `tcp_raw_stream` permite simular un flujo arbitrario de datos sobre una conexión TCP existente.

Parámetro	**Tipo**	**Descripción**	**Requerido**
`src_ip`	string	Dirección IP de origen del flujo TCP. Por defecto se genera con Capa 0.	No
`dst_ip`	string	Dirección IP de destino. Por defecto se genera con Capa 0.	No
`dst_port`	int	Puerto del servicio al que se conecta el cliente. Por defecto se genera con Capa 0.	No
`src_port`	int	Puerto origen del cliente.Por defecto se genera con Capa 0.	No
`payload_list`	list	Lista de cargas útiles que se transmitirán en el flujo TCP.	Sí
`timestamp_base`	datetime	Instante inicial del flujo. Por defecto: momento de ejecución	No
`distribution_function`	object	Distribución temporal aplicada entre los paquetes generados. Por defecto: Uniforme	No
`seed`	int	Semilla de generación pseudoaleatoria. Sin valor por defecto.	No

Tabla 4.5: Primitiva `tcp_raw_stream`: generación de un flujo de datos arbitrario sobre TCP.

Esta primitiva utiliza internamente el módulo de gestión TCP de la capa inferior. En primer lugar se genera el establecimiento de la conexión mediante `start_conn`. Posteriormente se insertan paquetes de datos utilizando `add_to_stream` para cada par de elementos de la lista `payload_list`, cabe destacar que la lista `payload_list` se diseño como una lista de diccionarios de tipo `consulta: respuesta`. Finalmente, la conexión se cierra mediante la primitiva `close_conn`. De esta manera se obtiene un flujo TCP completo con coherencia en los números de secuencia y reconocimiento.

La primitiva `http_web_navigation` representa la navegación web hacia un servidor específico.

Esta primitiva coordina múltiples primitivas para generar una secuencia realista de tráfico web. Inicialmente se ejecuta una resolución DNS utilizando la primitiva de capa 2 `dns_conversation` para obtener la dirección IP del servidor asociado al dominio indicado en `host`.

Parámetro	Tipo	Descripción	Requerido
`host`	string	Nombre del servidor web al que se accede.	Sí
`src_ip`	string	Dirección IP del cliente que navega. Por defecto: se genera con Capa 0.	No
`dns_server`	string	Servidor DNS utilizado para resolver el dominio. Por defecto: se genera con Capa 0.	No
`resources`	list	Lista de recursos solicitados (tipo y tamaño aproximado). Por defecto: vacío.	No
`dst_port`	int	Puerto del servicio al que se conecta el cliente. Por defecto: se genera con Capa 0.	No
`src_port`	int	Puerto origen del cliente. Por defecto: se genera con Capa 0.	No
`timestamp_base`	datetime	Instante inicial de la navegación. Por defecto: momento de ejecución.	No
`distribution_function`	object	Distribución temporal utilizada entre solicitudes HTTP. Por defecto: Uniforme.	No
`seed`	int	Semilla para generación determinística. Sin valor por defecto.	No

Tabla 4.6: Primitiva `http_web_navigation`: simulación de una sesión de navegación web.

Una vez resuelta la dirección, se establece una conexión TCP hacia el servidor mediante `start_conn`. Sobre esta conexión se generan las solicitudes HTTP correspondientes a los recursos indicados en el parámetro `resources`. Cada recurso puede representar, por ejemplo, un documento HTML, una hoja de estilos, un *script* o un archivo *JSON*, generando múltiples intercambios de paquetes TCP mediante llamadas sucesivas a `add_to_stream`. Finalmente, una vez completada la transferencia de recursos, la conexión se finaliza utilizando la primitiva `close_conn`.

Este mecanismo permite generar capturas de tráfico que representan de forma realista la carga de una página web completa, incluyendo la resolución DNS previa y la descarga de múltiples recursos asociados en la misma conexión como permite el protocolo `HTTP 1.1`.

Además de las primitivas descritas anteriormente, se implementaron otros orquestadores de red. Entre ellos se incluyen primitivas para generar una consulta con su respectiva resolución DNS *dns_conversation*, simular descargas completas de archivos (`file_download`), mantener intercambios arbitrarios de datos sobre una conexión TCP (`tcp_raw_stream`) y para generar patrones de tráfico asociados a ataques de denegación de servicio, como inundaciones SYN (`syn_flood_stream`).

4.5.3. Capa 3: Perfiles de Red (Perfiles)

Al igual que en los otros ejes, los perfiles de red permiten desplegar escenarios masivos. Por ejemplo, si el instructor desea que el artefacto forense de interés sea una exfiltración de datos realizada a través de la red, resulta poco realista

presentar una captura donde únicamente aparezca dicho flujo malicioso. En entornos reales, el tráfico relevante suele encontrarse inmerso dentro de grandes volúmenes de comunicaciones legítimas.

Para generar este contexto, puede utilizarse un perfil de *Navegación Web* que produzca actividad de fondo representativa del uso cotidiano de una red corporativa, generando múltiples sesiones web hacia dominios variados. De esta manera, el tráfico malicioso queda oculto dentro de un conjunto mayor de comunicaciones legítimas, obligando al estudiante a realizar un análisis más exhaustivo de la captura.

En el ámbito ofensivo, también es posible definir perfiles que modelen patrones de ataque específicos. Por ejemplo, un perfil de inundación SYN puede generar grandes volúmenes de intentos de establecimiento de conexión TCP hacia un mismo servicio, reproduciendo el comportamiento característico de un ataque de denegación de servicio.

4.6. Artefactos de Procesos en Memoria

En el eje de memoria, la unidad mínima de modelado es el proceso en ejecución o la estructura interna asociada a este, tales como regiones de memoria o módulos cargados. No obstante, el valor forense en este dominio rara vez se encuentra en una estructura aislada, sino en la relación entre procesos, sus estados, sus interacciones y la huella que dejan en el espacio de memoria. La presencia de inyecciones de código, procesos huérfanos o escaladas de privilegios solo adquiere significado cuando se analiza el contexto completo de ejecución. Por lo tanto, el diseño distingue entre la generación de entidades individuales y mecanismos que permiten construir estados de sistema más amplios, donde múltiples elementos coexisten de forma coherente y verificable.

Cabe destacar que por cuestiones de verbosidad no se detallan todas las primitivas. Sin embargo, la totalidad de las mismas con sus explicaciones se encuentran en el documento anexo `Diseño de Primitivas`.

4.6.1. Capa 1: Primitivas Atómicas de Proceso

Estas primitivas definen las propiedades fundamentales de la existencia y el estado de un proceso individual en el sistema. Su objetivo es garantizar que la información reflejada en la memoria RAM y en las estructuras de control del núcleo sea pericialmente útil y sirva como base para las capas superiores.

La Tabla 4.7 describe los parámetros de la primitiva `execute_proc`, que constituye la operación elemental de simulación de un proceso individual en ejecución.

Además de `execute_proc`, se contemplan dos primitivas complementarias para la manipulación de procesos existentes. La primitiva `set_proc_state` modifica el estado de un proceso, recibiendo el identificador del proceso, el estado deseado (`running`, `sleeping`, `zombie`, entre otros), una marca de tiempo y opcionalmente la prioridad de planificación del proceso, que de no indicarse queda

Parámetro	Tipo	Descripción	Requerido
`name`	string	Nombre del binario o comando a simular.	Sí
`user`	string	Usuario bajo cuyo contexto se ejecuta el proceso.	Sí
`timestamp`	datetime	Momento de inicio del proceso. Por defecto: momento de la ejecución.	No
`args`	list	Argumentos de línea de comando. Por defecto: Sin valor por defecto.	No
`pid`	int	identificador del proceso. Por defecto: generado por Capa 0.	No
`ppid`	int	Identificador del proceso padre. Por defecto: generado por Capa 0.	No
`environment_vars`	dict	Variables de entorno asociadas al proceso. Por defecto: vacío.	No
`seed`	int	Semilla para garantizar reproducibilidad de campos auxiliares generados. Sin valor por defecto.	No

Tabla 4.7: Primitiva `execute_proc` —simulación de un proceso individual con atributos de ejecución controlados.

con la prioridad por defecto del sistema. La primitiva `map_resource` asocia recursos a un proceso en ejecución, recibiendo el identificador del proceso y una marca de tiempo, opcionalmente acepta una lista de archivos abiertos y una lista de conexiones activas, ambas vacías por defecto.

4.6.2. Capa 2: Orquestadores de Procesos

Las primitivas de este nivel permiten gestionar conjuntos de ejecuciones para simular comportamientos del sistema o ataques coordinados que involucran múltiples procesos relacionados.

La Tabla 4.8 describe los parámetros del orquestador `process_batch`, que coordina la generación de un conjunto de procesos con distribución de usuarios y roles configurable.

4.6.3. Capa 3: Perfiles de Procesos

Los perfiles en este eje permiten modelar patrones completos de ejecución de procesos asociados a distintos roles operativos dentro de un sistema. Cada perfil orquesta primitivas de las capas inferiores para generar un conjunto de procesos coherente con el tipo de actividad que se desea representar.

Por ejemplo, si el instructor desea construir un escenario donde el análisis forense de memoria sea relevante, resulta útil contar con un conjunto de procesos que refleje la actividad habitual del sistema. Un perfil de *Administrador de Sistemas* puede generar múltiples procesos de monitoreo, herramientas de administración y ejecuciones de comandos con permisos privilegiados, reproduciendo el tipo de actividad que este rol suele realizar sobre el sistema.

Parámetro	Tipo	Descripción	Requerido
`count`	int	Cantidad de procesos a generar.	Sí
`user_distribution`	dict	Distribución de usuarios propietarios de los procesos generados. Por defecto: {`user: 100`}.	No
`role`	dict	Distribución de tipos de procesos con porcentajes que deben sumar 100. Por defecto: {`daemon: 10, system: 30, user: 60`}.	No
`timestamp_base`	datetime	Instante inicial a partir del cual se distribuyen los procesos generados.	Sí
`distribution_function`	function	Función utilizada para espaciar temporalmente los procesos generados. Por defecto: distribución uniforme.	No
`seed`	int	Semilla para garantizar reproducibilidad. Sin valor por defecto.	No

Tabla 4.8: Orquestador `process_batch`: generación de un conjunto de procesos relacionados con distribución de usuarios y roles configurable.

La presencia de estos procesos permite construir capturas de memoria más realistas, donde los artefactos de interés se encuentran inmersos dentro de un conjunto mayor de actividad legítima, obligando al estudiante a distinguir entre procesos habituales del sistema y aquellos potencialmente maliciosos.

Capítulo 5

Implementación

El presente capítulo detalla la implementación de la arquitectura diseñada, describiendo la organización modular del sistema de generación de artefactos, las herramientas seleccionadas y los desafíos técnicos resueltos durante el desarrollo.

5.1. Alcance de la Implementación

Tras el análisis de requerimientos y la evaluación del diseño, se definió un alcance de implementación que prioriza demostrar la viabilidad del enfoque propuesto. El alcance acordado comprende la implementación de dos de las tres vertientes principales: artefactos de red y artefactos de sistema de archivos.

La implementación de artefactos de memoria quedó fuera del alcance inicial, aunque se diseñó la arquitectura para facilitar su incorporación futura. Los análisis y decisiones de diseño presentados en capítulos anteriores quedan como maqueta para su próxima implementación.

Adicionalmente, en lo que respecta al contenido de los artefactos, se adopta un enfoque de tipo *best effort*. Esto implica que el sistema genera contenido válido, pero sin modelar en profundidad las estructuras y relaciones semánticas que podrían existir en sistemas reales.

5.2. Integración con la Infraestructura de Tectonic

El proyecto es una extensión de Tectonic, y uno de los requerimientos es que la solución se integre de forma natural con la infraestructura ya existente. Dado que Tectonic se apoya en Ansible como motor de automatización para el despliegue de laboratorios, se decidió implementar la solución y particularmente el lenguaje de especificación aprovechando directamente sus mecanismos nativos: archivos YAML (Ben Kiki, Evans, y Ingerson, 2021) combinados con el motor de plantillas Jinja2 (Ronacher, 2025).

Esta decisión permitió preservar la coherencia tecnológica del ecosistema. Además, tanto YAML como Jinja2 poseen una naturaleza declarativa y parametrizable, características alineadas con las cualidades del lenguaje diseñado.

Durante la implementación de las primitivas forenses se identificó una limitación estructural: Tectonic utilizaba Ansible únicamente con sus módulos estándar, ejecutando tareas declarativas desde playbooks YAML. Sin embargo, la mayoría de las primitivas definidas en este proyecto —como la generación de flujos TCP o la manipulación precisa de metadatos de archivos— no existen como funcionalidades nativas de Ansible.

Por esta razón es necesario permitir la ejecución de código Python personalizado desde los playbooks. Para soportar estas funcionalidades se utilizó el mecanismo estándar de extensión de Ansible. Este mecanismo permite implementar módulos personalizados de Ansible escritos en Python, los cuales deben estar ubicados en el directorio `library/` y son invocables directamente desde los playbooks como cualquier módulo nativo.

Implementar cada primitiva directamente como un módulo de Ansible implicaría repetir en cada módulo la lógica de construcción de paquetes, generación de direcciones IP/MAC, manejo de timestamps y escritura en archivos PCAP (archivo que contiene tráfico de red). Por esta razón, para evitar la duplicación de código y promover una estructura modular, se utilizó otro mecanismo provisto por Ansible. El mismo permite ubicar código Python reutilizable en el directorio `module_utils/` que puede ser importado por los módulos definidos en `library/`, pero no es invocable directamente desde los playbooks. De esta forma, todas las primitivas forenses y su organización jerárquica residen en `module_utils/`, mientras que los módulos de `library/` actúan como adaptadores que traducen parámetros YAML en llamadas a dichas primitivas.

Siguiendo la arquitectura conceptual del sistema, dentro de `module_utils/` se creó el espacio `dfir/`, separado en: `filesystem/` y `network/`. Y dentro de cada eje en: `first_layer/`, `second_layer/` y `third_layer/`.

De esta forma, los módulos de Ansible definidos en `library/` actúan como interfaz de invocación desde los playbooks, mientras que la lógica de generación de artefactos se implementa en primitivas reutilizables ubicadas en `module_utils/`.

Adicionalmente, se incorporó el directorio `filter_plugins/`, que permite definir filtros personalizados de Jinja2. Estos filtros exponen funciones auxiliares —que implementan la capa 0— que pueden utilizarse directamente en expresiones dentro de los playbooks.

Para soportar esta estructura fue necesario realizar una modificación puntual en el componente `tectonic/ansible.py`. Agregando las variables de entorno `tectonic_library_path`, `tectonic_filter_plugins_path` y `tectonic_module_utils_path` se generó un mecanismo para que se detecten automáticamente los directorios: `library/`, `module_utils/` y `filter_plugins/`. Con esta modificación, cualquier módulo o filtro agregado en dichas carpetas es descubierto automáticamente sin necesidad de alterar el núcleo de Tectonic.

De esta manera, se logró integrar una arquitectura en capas dentro del modelo de Ansible, manteniendo reutilización de código, modularidad y capacidad de extensión futura.

5.3. Propiedades Transversales de los Artefactos

Las propiedades transversales definidas en el diseño se reflejan directamente en la implementación de las primitivas.

En lo que respecta a la temporalidad, la gestión del tiempo se basa en el estándar *Unix Epoch*, es decir, la cantidad de segundos transcurridos desde el 1 de enero de 1970. En las primitivas de la primera capa, el tiempo se especifica directamente mediante un único parámetro de tipo `float` que representa el timestamp exacto asociado al artefacto generado.

En cambio, las primitivas de capas superiores no reciben un tiempo fijo, sino un *Timestamp Base* junto con un diccionario de configuración que define la distribución temporal utilizada para generar eventos derivados. Este diccionario requiere un campo `type`, cuyos valores posibles son `uniform`, `normal` o `exponential`, y un conjunto de parámetros específicos para la distribución seleccionada.

Por ejemplo, en el caso de una distribución de tipo `normal`, el instructor debe proporcionar los parámetros `mu` (media) y `sigma` (desviación estándar). A partir de esta información, las primitivas generan timestamps derivados que mantienen coherencia temporal dentro del escenario.

Las funciones de distribución utilizadas en la implementación corresponden a los siguientes métodos de la biblioteca estándar `random` de Python:

- **Uniforme**: `random.uniform(a, b)`.
- **Normal**: `random.gauss(mu, sigma)`.
- **Exponencial**: `random.expovariate(1/mu)`.

En lo que respecta a la gestión de semillas, la misma resulta directa de implementar debido a que la biblioteca `Faker` (biblioteca utilizada para la implementación de la Capa 0, se brinda más detalle en la sección siguiente) soporta nativamente la inicialización mediante semillas. En consecuencia, cada primitiva de la capa 0 recibe un parámetro de semilla que es utilizado para inicializar la instancia de `Faker`. De esta forma, para una misma semilla y los mismos parámetros de entrada, la generación de datos sintéticos produce siempre los mismos resultados, garantizando la reproducibilidad de los escenarios.

5.4. Implementación de la Capa 0

La Capa 0 es la capa transversal encargada de la generación de datos pseudoaleatorios. Su implementación se basa en la integración de la biblioteca Faker (Faraglia y Faker Contributors, 2025), una herramienta de Python diseñada para generar datos ficticios pero con formato realista y contenido pseudoaleatorio (nombres, direcciones, perfiles de usuario, IPs, etc.).

Debido a que la plataforma Tectonic utiliza Ansible para la configuración de los laboratorios, se diseñaron dos formas de acceder a las capacidades de Faker: una orientada al uso directo del instructor en archivos de configuración (Filtros) y otra orientada a la lógica interna de los generadores de evidencia (Funciones Utilitarias).

El soporte para estos mecanismos se integra con la modificación arquitectónica descrita en la Sección 5.2, donde se incorporó el descubrimiento automático de los directorios `filter_plugins/` y `module_utils/`. Gracias a esta configuración, los filtros y utilidades de la Capa 0 son detectados dinámicamente por Ansible sin requerir configuración adicional por parte del usuario final.

5.4.1. Filtros personalizados de Jinja2

Ansible incorpora nativamente el concepto de filtros de Jinja2, que permiten extender el lenguaje declarativo sin modificar su sintaxis. Un filtro es una función Python invocable dentro de expresiones YAML mediante la sintaxis de tubería (*pipe*). Por ejemplo, {{ '' | `faker_name` }} genera un nombre pseudoaleatorio en tiempo de ejecución.

La implementación consiste en un complemento (*plugin*) que define una clase `FilterModule` registrando cada función de `Faker` como filtro disponible para Ansible. De esta forma, el instructor puede generar datos sintéticos pseudoaleatorios en la configuración del escenario sin especificarlos de antemano:

```
1 - name: Crear carpeta personal del usuario
2   file:
3     path: "/home/{{ '' | faker_name(seed=123) }}/Documentos"
4     state: directory
```

A continuación en la Tabla 5.1 se presentan los filtros disponibles en Capa 0 junto con ejemplos de retorno.

5.4.2. Funciones Utilitarias

Este módulo actúa como una biblioteca interna que provee funciones utilitarias que pueden ser invocadas directamente desde las primitivas de generación de evidencia. Entre ellas se incluyen funciones como `get_rnd_port()`, `get_private_ip()` o `get_text_content()`, que permiten generar valores pseudoaleatorios que se utilizan en la creación de artefactos.

Este mecanismo cumple dos objetivos principales. Por un lado, permite que ciertas primitivas generen pseudoaleatoriamente valores, cuando estos no son especificados en la configuración del escenario. De esta manera, el instructor no necesita definir todos los parámetros de cada primitiva, lo que simplifica la generación de escenarios.

Por otro lado, estas funciones permiten implementar lógicas de generación más complejas directamente en Python. Por ejemplo, es posible generar programáticamente una lista de cientos de direcciones IP únicas para simular un

Categoría	Filtro	Ejemplo de retorno

Identidades	`faker_name`
	`faker_email`
	`faker_phone`
	`faker_job`
		`Albano Llopis Hierro`
		`emiliocalatayud@example.net`
		`+34823 96 00 13`
		`Chef`

Empresas	`faker_company` `faker_cif`	`Manufacturas Españolas S.A.` `B1234567J`

Geografía	`faker_city`
	`faker_country`
	`faker_address`
		`Guipúzcoa`
		`Guinea`
		`Paseo Simón Giménez 59, Valladolid`

Red	`faker_ipv4`
	`faker_mac`
	`faker_url`
	`faker_user_agent`
		`166.107.43.197`
		`a6:89:21:6c:a1:6c`
		`https://hotel.com/`
		`Opera/8.81.(Windows NT 6.1; ...)`

Contenido	`faker_sentence`
	
	`faker_paragraph`
	`faker_lorem_text`
		`Te luis pueda considera tengo será.`
		`Construcción centro madrid.`
		`Existen área tan libertad. Sí lado clase.`

Fechas y números	`faker_date`
	`faker_number`
	`faker_color`
		`2018-01-20`
		`655`
		`Coral`

Listas	`faker_name_list` `faker_company_list`	`[Dorotea del Pujol, Constanza...]` `[Distribuciones Roca S.Coop., ...]`
Documentos	`faker_dni` `faker_iban`	`12345678Z` `ES4951462704828148932528`

Tabla 5.1: Filtros de Capa 0 expuestos hacia el usuario final en Tectonic.

ataque de denegación de servicio, una tarea que resultaría ineficiente de expresar manualmente mediante filtros en YAML.

5.5. Implementación de Artefactos de Sistema de Archivos

Esta sección describe el proceso de implementación de artefactos de sistema de archivos. Se comienza por justificar el enfoque adoptado en cuanto a generación y distribución de evidencia, seguido de la descripción de las bibliotecas y primitivas utilizadas para la creación de artefactos. Posteriormente se detallan los perfiles de usuario implementados, los desafíos técnicos encontrados y las estrategias adoptadas para resolverlos, y finalmente el proceso de validación que confirma la corrección de los artefactos generados.

5.5.1. Estrategia de generación y distribución

Siguiendo el análisis presentado en la Sección 3.2.4, la implementación adopta un enfoque de generación *online* y distribución dinámica de los artefactos.

La generación *online* se implementa mediante módulos de Ansible que ejecutan acciones reales sobre el sistema objetivo. De esta manera, los artefactos producidos —archivos, directorios y sus metadatos asociados— surgen como consecuencia directa de las operaciones realizadas por el sistema operativo, garantizando coherencia natural entre eventos y marcas temporales.

En cuanto a la distribución, los artefactos se presentan dentro de un sistema en ejecución. Este enfoque permite que el estudiante interactúe con el entorno, realice tareas de adquisición y posteriormente lleve a cabo el análisis de la evidencia. Este flujo reproduce un escenario DFIR, donde el analista no solo examina artefactos, sino que también participa en su recolección.

Sin embargo, existen ciertos artefactos que no pueden generarse de forma controlada únicamente mediante llamadas al sistema operativo. Un ejemplo es el caso de archivos eliminados que deben permanecer recuperables. En un sistema en ejecución, la posible reasignación de los bloques liberados al eliminar un archivo depende del comportamiento interno del sistema de archivos y del sistema operativo. Como consecuencia, no es posible garantizar que los metadatos y los bloques de datos del archivo eliminado permanezcan intactos para su posterior recuperación. Para estos casos particulares se adopta una estrategia complementaria: los artefactos del sistema de archivos se generan sobre una imagen desmontada. Permitiendo así garantizar la recuperabilidad de los archivos, sin comprometer la integridad del sistema en ejecución. De esta forma, el escenario continúa siendo dinámico desde la perspectiva del estudiante, aunque el objeto de estudio ya se encuentra delimitado.

5.5.2. Creación de los artefactos

Para crear los artefactos atómicos en sistema de archivos se implementó la primitiva `generate_file`, que genera archivos del tipo especificado en el parámetro `extension`. El módulo se invoca desde Ansible de la siguiente manera:

```
- name: "Generate file"
  generate_file:
    path: "/tmp/document.pdf"
    extension: pdf
```

Listado 5.1: Invocación de la primitiva `generate_file`

Se emplearon múltiples bibliotecas especializadas según el tipo de archivo:

- **Pillow (PIL)**: Generación de imágenes JPEG/PNG con contenido visual aleatorio reproducible((Alex) y Pillow Contributors, 2025).

- **python-docx**: Creación de documentos Word con estructura y metadatos sintéticos(Canny y python-docx Contributors, 2013).

- **openpyxl**: Generación de hojas de cálculo Excel con datos tabulares sintéticos(Gazoni y Clark, 2024).

- **Pandoc y pdflatex**: Conversión de Markdown a PDF con formato profesional(MacFarlane, 2025).

- **zipfile/tarfile**: Creación de archivos comprimidos con contenido interno generado(Python Software Foundation, 2025b)(Python Software Foundation, 2025a).

- **Cryptography**: Cifrado AES-256-CBC real para simulación de ransomware, con vector de inicialización (IV) pseudoaleatorio y padding PKCS7. El IV es un bloque de 16 bytes colocado al inicio del contenido, generado pseudoaleatoriamente que se combina con la clave de cifrado para garantizar que dos cifrados del mismo contenido produzcan resultados distintos. El padding PKCS7 es un esquema que completa el último bloque de datos hasta el tamaño de bloque requerido por el cifrado (16 bytes en AES), rellenando los bytes faltantes con un valor igual a la cantidad de bytes añadidos, lo que permite al receptor identificar y descartar el relleno al descifrar(Python Cryptographic Authority, 2025).

A modo de ejemplo, el Listado 5.2 presenta la primitiva de Capa 1 para generación de archivos PDF. La función inicializa Faker con la semilla recibida como parámetro y construye el contenido en formato Markdown con título, autor y cuerpo generados sintéticamente (líneas 2–9). Este contenido se escribe en un archivo temporal (líneas 11–15) que es posteriormente procesado por `pandoc` con motor `pdflatex` para producir el PDF final en la ruta especificada (líneas 17–21).

Posteriormente, en la capa 2, para crear un conjunto de artefactos en sistema de archivos se utiliza el orquestador `generate_files_bulk`. Este recibe la cantidad total de archivos a generar (`count`), un diccionario de distribución por tipo (`distribution`) que es normalizado y traducido a conteos exactos mediante `distribute_by_percentages`, el directorio base (`base_directory`), el tipo de estructura de directorios (`structure`), que determina si los archivos se distribuyen en un único nivel o en un árbol configurado por `tree_depth` y `subdirs_per_level`, una plantilla de nombres (`name_pattern`) expandida por `expand_filename_pattern` con variables básicas y de datos sintéticos, y una semilla (`faker_seed`) que garantiza reproducibilidad tanto en nombres como en contenido. Por cada archivo, el orquestador deriva una semilla única como `faker_seed + file_index`, como se explica en 4.3.2, y delega la creación al tipo correspondiente mediante `_create_file_by_type`, que internamente invoca la primitiva de Capa 1 adecuada. El Listado 5.3 muestra un ejemplo de invocación desde Ansible, y el Listado 5.4 presenta la implementación del orquestador.

```
def create_pdf_file(filepath, content=None, seed=None):
    Faker.seed(seed)
    fake = Faker()
    markdown_content = f"""---
title: "{fake.sentence()}"
author: "{fake.name()}"
---
# {fake.sentence()}
{fake.paragraph(nb_sentences=5)}"""

    temp_md = tempfile.NamedTemporaryFile(
        mode='w', suffix='.md'
    )
    temp_md.write(markdown_content)
    temp_md.close()

    subprocess.run(
        ['pandoc', temp_md.name, '-o', filepath,
         '--pdf-engine=pdflatex'],
        timeout=30, check=True
    )
```

Listado 5.2: Primitiva de generación de archivos PDF.

Se puede observar que esta invocación genera 100 archivos distribuidos en 30 PDFs, 40 documentos Word, 20 archivos de texto y 10 imágenes JPEG, organizados en un árbol de directorios de profundidad 3 con 2 directorios por nivel bajo `/tmp/documents`. Los nombres de archivo siguen el patrón {`faker_company`}_{`n`}, por lo que cada archivo recibe un nombre de empresa sintético único derivado de la semilla 42 combinada con su índice, resultando en nombres como `Manufacturas_Españolas_0.pdf` o `Distribuciones_Roca_1.docx`.

Finalmente, se expone la capa 3, los perfiles implementados para sistema de archivos son los siguientes:

- **Oficinista**: Simula el entorno de trabajo de un empleado corporativo, con documentos Word organizados en carpetas de proyectos y reportes.
- **Desarrollador**: Reproduce el espacio de trabajo de un programador, con código fuente Python, scripts de configuración y archivos de texto distribuidos en estructuras de directorios propias de proyectos de software.
- **Diseñador**: Modela el directorio de un diseñador gráfico, poblado principalmente con archivos SVG organizados por proyectos y entregas.
- **Usuario personal**: Refleja el uso doméstico y ocasional de un equipo, con fotos distribuidas sin una estructura organizativa definida.
- **Estudiante**: Representa el entorno de un estudiante universitario, con PDFs distribuidos en el sistema.

```yaml
- name: "Generate files bulk"
  generate_files_bulk:
    count: 100
    base_directory: "/tmp/documents"
    distribution:
      pdf: 30
      docx: 40
      txt: 20
      jpg: 10
    structure: tree
    tree_depth: 3
    subdirs_per_level: 2
    name_pattern: "{faker_company}_{n}"
    faker_seed: 42
```

Listado 5.3: Invocación de `generate_files_bulk` desde Ansible

- **Servidor**: Simula un servidor de archivos con zip distribuidos en el mismo.
- **Ransomware**: Perfil malicioso implementado como caso de estudio integrador, detallado en el Capítulo 6.

A continuación en el Listado 5.5 se brinda un ejemplo de invocación desde Ansible para generar un perfil de actividad de fondo.

La invocación anterior genera, bajo el directorio `/srv/users/employee1`, una estructura de carpetas compuesta por Documentos, Reportes y Presentaciones (creadas de acuerdo con la configuración del perfil). Estas carpetas se pueblan con 200 documentos (`file_count`) de tipo Word, cuyos nombres y contenidos se generan de forma pseudoaleatoria pero determinística a partir de la semilla `faker_seed`.

La implementación del orquestador de Capa 3 sigue el mismo principio que el de Capa 2. El método `generate_fs_user_profile` recibe el tipo de perfil y lo resuelve contra el registro estático `AVAILABLE_PROFILES`, un diccionario definido en el mismo módulo que asocia cada tipo de perfil con su configuración: extensiones de archivo habilitadas, subdirectorios a crear, patrones de nombres y nivel de actividad. A partir de esa configuración, el orquestador crea la estructura de directorios, distribuye los archivos de forma uniforme entre ellos invocando `generate_files_bulk` de Capa 2 con una semilla derivada por directorio, aplica una distribución temporal sobre las marcas de tiempo mediante `apply_temporal_distribution` según el nivel de actividad del perfil, y finalmente ajusta los permisos de cada archivo con `change_file_permissions` de Capa 1 según el modo correspondiente al tipo de usuario simulado.

5.5.3. Desafíos de Implementación

A continuación se presentan los desafíos encontrados durante la implentación de artefactos forenses en sistemas de archivos, y cómo los mismos fueron

```python
def generate_files_bulk(count, distribution, base_directory,
                        structure='flat', name_pattern='
                            document_{n}',
                        tree_depth=3, subdirs_per_level=3,
                            faker_seed=1):
    type_counts = distribute_by_percentages(count,
        distribution)
    if structure == 'flat':
            directories = _generate_flat_structure(
                base_directory, count)
        elif structure == 'tree':
            directories = _generate_tree_structure(
                base_directory, count, tree_depth,
                    subdirs_per_level, faker_seed
            )
    created_files, file_index = [], 0
    for file_type, file_count in type_counts.items():
        for i in range(file_count):
            filename = expand_filename_pattern(
                name_pattern, file_index, file_type, seed=
                    faker_seed)
            filepath = os.path.join(
                directories[file_index % len(directories)],
                    filename)
            _create_file_by_type(filepath, file_type,
                                 seed=faker_seed +
                                     file_index)
            file_index += 1
```

Listado 5.4: Implementación del orquestador `generate_files_bulk`

abordados.

Eliminación de archivos

La eliminación de archivos mediante la primitiva `os.remove()` presentó dificultades dado que su comportamiento depende de la infraestructura subyacente del sistema de archivos, al tratarse de una llamada de alto nivel del sistema operativo. En sistemas ext4 sobre almacenamiento SSD, el sistema operativo puede sobrescribir o limpiar los bloques inmediatamente tras la operación a nivel de firmware, imposibilitando la recuperación mediante herramientas forenses tradicionales. En sistemas más antiguos con HDD y ext3, los bloques pueden permanecer intactos pero marcados como libres, permitiendo su recuperación con herramientas como The Sleuth Kit. Esta inconsistencia hace que una primitiva basada en `os.remove()` no sea suficiente para garantizar la recuperabilidad del artefacto con independencia del entorno de despliegue, lo cual es importante tener en un entorno educativo.

```
- name: "Generate user profile"
  generate_fs_user_profile:
    profile_type: office
    base_directory: /srv/users/employee1
    faker_seed: 42
    file_count: 200
```

Listado 5.5: Invocación de `generate_fs_user_profile`

Para resolver esta limitación, la primitiva de eliminación fue dividida en dos variantes según el nivel de abstracción requerido. La variante de alto nivel conserva `os.remove()` para casos donde la recuperabilidad no es relevante. La variante de bajo nivel requiere que se opere directamente sobre la partición desmontada mediante `debugfs`. Una partición es una división lógica de un dispositivo de almacenamiento que contiene un sistema de archivos, montarla es el proceso mediante el cual el sistema operativo la integra en el árbol de directorios, habilitando el acceso a sus contenidos. Para operar a bajo nivel sin comprometer la integridad de los artefactos, la partición debe estar desmontada: al encontrarse fuera de línea, el sistema operativo no genera actividad de escritura que pueda sobrescribir los bloques de datos subyacentes. De este modo, la eliminación queda registrada a nivel de metadatos del sistema de archivos pero el contenido permanece recuperable mediante herramientas estándar como Autopsy o The Sleuth Kit(Basis Technology, 2025; Carrier, 2005).

Marca de tiempo de creación de archivos

En sistemas de archivos Linux es posible modificar las marcas de tiempo de modificación y acceso, pero no la de creación mediante llamadas estándar del sistema operativo. La modificación de esta marca requiere operar directamente sobre la partición desmontada mediante `debugfs`, de forma análoga a la variante de bajo nivel descrita para la eliminación de archivos. Esta funcionalidad quedó fuera del alcance del presente trabajo y se propone como línea de trabajo futuro.

5.5.4. Validación de Artefactos de Sistema de Archivos

Para validar que el sistema de archivos generado se corresponde con el esperado se utilizaron distintas herramientas forenses.

Se utilizó *Autopsy* (Basis Technology, 2025) para verificar que la imagen de disco generada contenga los artefactos esperados en la estructura de directorios correcta.

Una segunda forma de verificación consistió en montar el sistema de archivos y navegar su contenido mediante `ls`, abriendo archivos de distintos tipos (PDF, SVG, Word, entre otros) para confirmar que fueron generados con contenido pseudoaleatorio y sus nombres siguen el patrón definido en el momento de su creación.

Se utilizó *CyberChef* (GCHQ, 2025) para validar que el cifrado de los archivos sea el correcto, extrayendo el vector de inicialización embebido en los primeros 16 bytes de cada archivo cifrado y aplicando la clave de cifrado conocida para verificar que el descifrado produce el contenido original.

Finalmente, para verificar el correcto funcionamiento del borrado forense, se utilizó *The Sleuth Kit* (Carrier, 2005), identificando las entradas de directorio marcadas como eliminadas en el sistema de archivos y realizando el proceso de recuperación a partir de la referencia interna al bloque de datos correspondiente, confirmando que el contenido permanece intacto y es recuperable.

5.6. Implementación de Artefactos de Red

Esta sección describe la implementación del módulo encargado de la generación de artefactos de red. En primer lugar se presenta la estrategia adoptada para la generación y distribución del tráfico.. Posteriormente se presenta el proceso de construcción de paquetes utilizando Scapy (Biondi y Scapy Community, 2025). Finalmente, se describen los perfiles de actividad desarrollados y el proceso de validación aplicado para verificar la integridad y el comportamiento del tráfico generado.

5.6.1. Estrategia de generación y distribución

Siguiendo el análisis presentado en la Sección 3.2.4, la implementación adopta un enfoque de generación *offline* y distribución estática de los artefactos de red.

La generación *offline* se realiza mediante la construcción de paquetes utilizando la biblioteca Scapy (Biondi y Scapy Community, 2025). Este enfoque permite definir explícitamente cada campo de los paquetes generados y construir capturas completas de tráfico en formato PCAP.

En cuanto a la distribución, el tráfico generado se entrega directamente como artefacto forense estático. De esta forma, los estudiantes disponen de una captura completa con la totalidad de artefactos, que puede ser analizada con herramientas forenses de red, sin depender del momento de captura.

Este enfoque permite controlar con precisión el contenido de la captura y definir explícitamente los artefactos que componen el escenario. Al mismo tiempo, evita la necesidad de desplegar y mantener múltiples máquinas virtuales generando tráfico en tiempo real y elimina la dependencia de que el estudiante capture el tráfico en el instante exacto en que ocurren los eventos relevantes. Finalmente, permite construir capturas que incluyen actividad de fondo distribuida a lo largo del tiempo sin requerir ejecuciones prolongadas ni capturas de larga duración.

5.6.2. Creación de los artefactos

Como se mencionó anteriormente, Scapy permite la construcción de paquetes con control total sobre:

- Capas Ethernet, IP y TCP/UDP con todos sus campos configurables.
- Cálculo automático de *checksums* válidos.
- Asignación precisa de *timestamps*.
- Exportación a formato PCAP estándar.

Estas características permiten implementar las primitivas de generación de tráfico propuestas en el diseño presentado en la sección 4.5.

Para simplificar la construcción de paquetes, la primitiva de diseño `create_packet` fue dividida en dos métodos concretos de implementación: `create_tcp_packet` y `create_udp_packet`. Esta decisión responde a que los protocolos de transporte TCP y UDP presentan diferencias en los campos que deben configurarse y en las validaciones necesarias durante la construcción del paquete. En particular, TCP incluye parámetros adicionales como *flags*, números de secuencia (`seq`) y reconocimiento (`ack`), que no están presentes en UDP. Separar ambas implementaciones permite simplificar la lógica de construcción de paquetes y aprovechar de forma más directa las estructuras provistas por la biblioteca Scapy.

Estos métodos encapsulan la lógica necesaria para construir la estructura base de un paquete de red. A partir de los parámetros recibidos generan las capas Ethernet, IP y de transporte correspondientes, junto con su *payload*.

El Listado 5.6 muestra la implementación de la primitiva `create_tcp_packet`, que construye la estructura de un paquete TCP que contiene:

- **Capa de enlace** (`Ether`): define las direcciones MAC de origen y destino.
- **Capa de red** (`IP`): especifica las direcciones IP involucradas.
- **Capa de transporte** (`TCP`): establece puertos y permite configurar campos como `flags`, `seq` y `ack`.

El operador `/` representa la encapsulación entre distintos encabezados de protocolo (por ejemplo Ethernet, IP y TCP). La expresión `Ether() / IP() / TCP()` indica que el segmento TCP se encapsula en un paquete IP, que a su vez se encapsula en una trama Ethernet. Finalmente, la capa `Raw(load=payload)` representa los datos de aplicación transportados por el segmento TCP, permitiendo incluir una carga útil arbitraria dentro del paquete. El resultado es un objeto paquete que puede serializarse en un archivo PCAP.

El uso de `**kwargs` permite parametrizar dinámicamente los campos del encabezado TCP, habilitando la generación flexible de distintos tipos de segmentos (por ejemplo SYN, ACK, FIN o PSH) así como la configuración de números de secuencia y reconocimiento.

```python
def create_tcp_packet(
    self, source_ip: str, dest_ip: str, source_port: str,
        dest_port: str,
    src_mac: Optional[str] = None, dst_mac:Optional[str] =
        None, payload, **kwargs
):
    client_mac = src_mac or Layer0.get_mac_address(seed)
    server_mac = dst_mac or Layer0.get_mac_address(seed)

    return Ether(src=client_mac, dst=server_mac) / \
           IP(src=source_ip, dst=dest_ip) / \
           TCP(sport=source_port, dport=dest_port, **kwargs)
               / \
           Raw(load=payload)
```

Listado 5.6: Implementación del método `create_tcp_packet` utilizando Scapy

En el Listado 5.7 se presenta un ejemplo simplificado —se omiten las validaciones y asignaciones de parámetros— de la primitiva de capa 1 correspondiente al inicio de una conexión TCP (three-way handshake), que constituye la interacción más simple del módulo.

La implementación muestra cómo la primitiva encapsula la lógica completa del establecimiento de una conexión TCP a partir de parámetros opcionales. En caso de no especificarse, valores como direcciones IP, números de puertos o números de secuencia son generados mediante la Capa 0. A partir de estos parámetros, se construyen los tres segmentos del *three-way handshake* utilizando la primitiva `create_tcp_packet`, definiendo explícitamente las banderas TCP correspondientes a cada etapa: un segmento inicial con flag `SYN`, la respuesta del servidor con `SYN-ACK`, y la confirmación final del cliente con `ACK`, junto con sus respectivos números de secuencia y confirmación. Finalmente, la primitiva retorna una lista ordenada de paquetes que representan el inicio de conexión.

```python
def initialize_connection(self, src_ip: Optional[str] = None
    , dst_ip: Optional[str] = None, src_port: Optional[int] =
     None, dst_port: Optional[int] = None, ini_seq_cli:
    Optional[int] = None, ini_ack_srvr: Optional[int] = None,
     ts: Optional[str] = None, seed:int):
    if src_ip is None:
        src_ip = Layer0.get_private_ip(seed)
    if dst_ip is None:
        dst_ip = Layer0.get_public_ip(seed)
    # ...
    if ini_ack_srvr is None:
        ini_ack_srvr = Layer0.get_seq(seed)

    syn = self.create_tcp_packet(
        src_ip, dst_ip, src_port, dst_port,
```

```
        flags="S", seq=ini_seq, seed
    )
    syn.time = ts
    syn_ack = self.create_tcp_packet(
        src_ip, dst_ip, src_port, dst_port,
        flags="SA", seq=ini_ack, ack=ini_seq + 1
    )
    syn_ack.time = ts + random.uniform(0.001, 0.1)
    ack = self.create_tcp_packet(
        src_ip, dst_ip, src_port, dst_port,
        flags="A", seq= ini_seq + 1, ack= ini_ack + 1
    )
    ack.time = syn_ack.time + random.uniform(0.001, 0.1)
    return [syn, syn_ack, ack]
```

Listado 5.7: Primitiva de inicio de conexión TCP

La Capa 2 se encarga de orquestar las listas de paquetes que la Capa 1 va retornando. En términos prácticos, el orquestador mantiene una lista acumulativa de paquetes y va agregando a dicha lista las secuencias generadas por cada primitiva invocada.

El ejemplo del Listado 5.8 muestra un fragmento simplificado —se eliminó para evitar demasiada verbosidad los chequeos de los parámetros, manejo de excepciones y la asignación de variables usando la Capa 0 en caso de que los parámetros sean `None`– de la primitiva `tcp_raw_stream`. Esta función modela una interacción TCP de intercambios arbitrarios de strings entre dos extremos (por ejemplo una comunicación telnet). La función recibe un diccionario de mensajes intercambiados entre atacante y víctima y genera los paquetes correspondientes a la conversación.

```

def tcp_raw_stream(
    self,
    messages: List[Dict[str, str]],
    src_ip: str,
    dst_ip: str,
    src_port: Optional[int] = None,
    dst_port: Optional[int] = None,
    base_timestamp: Optional[float] = None,
    dist_function: Optional[Dict[str,Any]] = None,
    seed: int = 0
) -> List[Packet]:

    timestamp = base_timestamp if base_timestamp is not None
        else now_ts()
    all_packets: List[Packet] = []

    generator = Layer1Generator()

    tcp_handshake = generator.start_complete_connection(
```

```
20          src_ip=src_ip,
21          dest_ip=dst_ip,
22          source_port=src_port,
23          dest_port=dst_port,
24          ts=timestamp
25      )
26
27      all_packets.extend(tcp_handshake)
28
29      new_time = Layer0.get_random_time(
30          initial_time=tcp_handshake[-1].time,
31          distribution_function_and_params=dist_function,
32          seed=seed
33      )
34
35      last_dst_packet = tcp_handshake[-1]
36
37      for message_pair in messages:
38
39          src_packet = generator.add_packet_in_progress(
40              previous_packet=last_dst_packet,
41              payload=message_pair["src"].encode("utf-8"),
42              flags="PA",
43              ts=new_time
44          )
45
46          all_packets.append(src_packet)
47          last_src_packet = src_packet
48
49          new_time = Layer0.get_random_time(
50              initial_time=new_time,
51              distribution_function_and_params=dist_function,
52              seed=seed
53          )
54
55          dst_packet = generator.add_packet_in_progress(
56              previous_packet=last_src_packet,
57              payload=message_pair["dst"].encode("utf-8"),
58              flags="PA",
59              ts=new_time
60          )
61
62          all_packets.append(dst_packet)
63          last_dst_packet = dst_packet
64
65          new_time = Layer0.get_random_time(
66              initial_time=new_time,
67              distribution_function_and_params=dist_function,
68              seed=seed
69          )
```

```
70
71      return all_packets
```

Listado 5.8: Orquestación de primitivas de capa 1 en una interacción TCP

En primer lugar se genera el *three-way handshake* mediante una primitiva de capa 1 que devuelve la lista de paquetes correspondientes al establecimiento de la conexión. Estos paquetes se agregan a la lista acumulativa mediante `extend`. Posteriormente, cada par de mensajes se transforma en dos segmentos TCP: uno enviado y otro como respuesta.

Para mantener la coherencia de la sesión TCP, el orquestador conserva referencias al último paquete enviado por cada extremo. Estas referencias se utilizan como entrada de la primitiva `add_packet_in_progress`, que calcula automáticamente los valores de secuencia y de reconocimiento correspondientes.

De esta manera, las primitivas de capa 2 permiten describir interacciones de red completas utilizando un nivel de abstracción mayor, delegando los detalles de construcción de paquetes a las primitivas atómicas de capa inferior.

La implementación de la Capa 3 mantiene el mismo principio de funcionamiento que la Capa 2. Al igual que en el nivel anterior, las primitivas de niveles inferiores retornan listas de paquetes y la lógica principal consiste en gestionar dichas listas y mantener la coherencia temporal de los paquetes generados.

La implementación de la Capa 3 desarrollada define dos tipos de perfiles: uno orientado a la generación de actividad legítima de fondo y otro destinado a simular comportamiento malicioso.

El perfil de actividad de fondo genera tráfico web benigno simulando patrones realistas de navegación. Para ello realiza múltiples invocaciones a la primitiva de capa 2 `http_web_navigation`, que modela sesiones completas de navegación incluyendo resolución DNS y conexiones HTTP. Los parámetros de este perfil permiten configurar, entre otros aspectos, la cantidad de usuarios simulados (cada usuario se corresponde a una diferente dirección IP de origen), el número de hosts visitados por cada usuario, las direcciones IP involucradas, el servidor DNS utilizado, los puertos destino y los parámetros asociados al modelo temporal y a la semilla de generación pseudoaleatoria.

A continuación se presenta un ejemplo de invocación de este perfil desde un *playbook* de Ansible. En la Figura 5.1 se ilustra el inicio de la captura de tráfico generada a partir de dicha configuración.

```
1  - name: Generate multiple users browsing (3 users, 5
       websites each)
2    generate_user_profile:
3      users_amount: 3
4      hosts_amount_per_user: 5
5      dns_srvr_ip: "8.8.8.8"
6      dest_port: 80
7      faker_seed: 42
8      output_path: "/tmp/multiple_users_browsing.pcap"
9    register: multiple_users
```

Listado 5.9: Invocación del User Browsing Profile desde Ansible

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.100	8.8.8.8	DNS	69	Standard query 0xa42d A acero.com
2	0.015000	8.8.8.8	192.168.1.100	DNS	94	Standard query response 0xa42d A acero.com A 40.86.188.107
3	0.017465	192.168.1.100	40.86.188.107	TCP	54	5276 → 80 [SYN] Seq=0 Win=8192 Len=0
4	0.017644	40.86.188.107	192.168.1.100	TCP	54	80 → 5276 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
5	0.017731	192.168.1.100	40.86.188.107	TCP	54	5276 → 80 [ACK] Seq=1 Ack=1 Win=8192 Len=0
6	0.018736	192.168.1.100	40.86.188.107	HTTP	271	GET / HTTP/1.1
7	0.019157	40.86.188.107	192.168.1.100	HTTP	4864	HTTP/1.1 200 OK (text/html)
8	0.019551	192.168.1.100	40.86.188.107	HTTP	293	GET /css/style1.css HTTP/1.1
9	0.019774	40.86.188.107	192.168.1.100	HTTP	667	HTTP/1.1 200 OK (text/html)
10	0.019903	192.168.1.100	40.86.188.107	HTTP	319	GET /js/script2.js HTTP/1.1
11	0.022178	40.86.188.107	192.168.1.100	HTTP	1152	HTTP/1.1 200 OK (text/html)
12	0.023387	192.168.1.100	40.86.188.107	HTTP	278	GET /js/script3.js HTTP/1.1
13	0.024319	40.86.188.107	192.168.1.100	HTTP	620	HTTP/1.1 200 OK (text/html)
14	0.025333	192.168.1.100	40.86.188.107	HTTP	279	GET /js/script4.js HTTP/1.1
15	0.025492	40.86.188.107	192.168.1.100	HTTP	1005	HTTP/1.1 200 OK (text/html)
16	0.025874	192.168.1.100	40.86.188.107	HTTP	316	GET /api/data5.json HTTP/1.1
17	0.026476	40.86.188.107	192.168.1.100	HTTP	452	HTTP/1.1 200 OK (text/html)
18	0.028457	192.168.1.100	40.86.188.107	TCP	54	5276 → 80 [FIN, ACK] Seq=1433 Ack=8039 Win=8192 Len=0
19	0.029088	40.86.188.107	192.168.1.100	TCP	54	80 → 5276 [ACK] Seq=8437 Ack=1434 Win=8192 Len=0
20	0.030773	40.86.188.107	192.168.1.100	TCP	54	80 → 5276 [FIN, ACK] Seq=8437 Ack=1434 Win=8192 Len=0
21	0.031510	192.168.1.100	40.86.188.107	TCP	54	5276 → 80 [ACK] Seq=1434 Ack=8438 Win=8192 Len=0
22	14.942301	192.168.1.100	8.8.8.8	DNS	74	Standard query 0xcf0d A promociones.es
23	14.957301	8.8.8.8	192.168.1.100	DNS	104	Standard query response 0xcf0d A promociones.es A 20.47.117.236
24	14.957634	192.168.1.100	20.47.117.236	TCP	54	33432 → 80 [SYN] Seq=0 Win=8192 Len=0
25	14.958353	20.47.117.236	192.168.1.100	TCP	54	80 → 33432 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
26	14.958495	192.168.1.100	20.47.117.236	TCP	54	33432 → 80 [ACK] Seq=1 Ack=1 Win=8192 Len=0
27	14.960173	192.168.1.100	20.47.117.236	HTTP	236	GET / HTTP/1.1
28	14.962340	20.47.117.236	192.168.1.100	HTTP	2981	HTTP/1.1 200 OK (text/html)
29	14.964268	192.168.1.100	20.47.117.236	HTTP	321	GET /css/style1.css HTTP/1.1

Figura 5.1: Inicio de captura de tráfico tras especificar `User Browsing Profile`

En primer lugar, se observa una consulta DNS dirigida al servidor cuya IP fue especificada por parámetro. La consulta solicita la resolución de un nombre de dominio *acero.com*, y obtiene una respuesta satisfactoria con la IP 40.86.188.107. Tanto el dominio consultado como la dirección IP resultante son generados por el nivel atómico del sistema (Capa 0) de manera pseudoaleatoria.

Tras la resolución exitosa del dominio, se establece la conexión TCP correspondiente hacia la dirección IP obtenida. A continuación se produce el intercambio HTTP completo, que incluye múltiples solicitudes GET destinadas a la obtención de distintos recursos del servidor, tales como el documento HTML principal (*GET /*), hojas de estilo (*GET /css/styles1.css*), archivos JavaScript (*GET /js/script2.js*, *GET /js/script3.js*, *GET /js/script4.js*)y un archivo JSON (*GET /api/data5.json*) con datos adicionales. Este comportamiento reproduce de manera realista la carga progresiva de un sitio web moderno.

Finalmente, la sesión HTTP se cierra correctamente. Posteriormente, se inicia una nueva consulta DNS correspondiente al siguiente host dentro de los cinco especificados en la invocación del perfil. Una vez que finalicen los 5 hosts para el primer usuario, comienzan los siguientes 5 hosts para el siguiente usuario y así hasta completar los 3 usuarios especificados en el parámetro `users_amount`. Este patrón se repite de acuerdo con los parámetros configurados, modelando una actividad de navegación web secuencial y coherente en el tiempo.

El segundo perfil implementado corresponde a la simulación de un ataque de denegación de servicio basado en el mecanismo *SYN Flood*. Este ataque explota el proceso de establecimiento de conexión del protocolo TCP enviando un gran número de segmentos con la bandera `SYN` activada hacia un servidor objetivo sin completar posteriormente el *three-way handshake*.

En un sistema real, el servidor responde con segmentos `SYN-ACK` y reserva recursos internos asociados a cada conexión parcial. Si el volumen de solicitudes es suficientemente elevado, la tabla de conexiones del servidor puede saturarse, impidiendo el establecimiento de nuevas conexiones legítimas.

El perfil implementado se enfoca en generar tráfico que reproduce el patrón

observable del ataque: un gran volumen de solicitudes SYN provenientes de múltiples direcciones IP sin completar el establecimiento de conexión.

Es posible combinar este perfil malicioso con otros perfiles o primitivas de capas más granulares. Por ejemplo, puede generarse simultáneamente tráfico HTTP legítimo mediante las primitivas de Capa 2 para simular clientes reales interactuando con el servidor mientras ocurre el ataque.

A continuación se presenta un ejemplo de invocación de este perfil desde un *playbook* de Ansible, el cual genera tráfico de tipo inundación SYN, hacia el puerto 443 de la IP 10.0.0.5, desde 500 IPs distintas.

```
- name: Generate Syn Flood Attack with 500 attackers
  generate_syn_flood:
    victim_ip: "10.0.0.50"
    victim_port: 443
    attackers_amount: 500
    faker_seed: 42
    output_path: "/tmp/syn_flood_500_attackers.pcap"
  register: multiple_users
```

Listado 5.10: Invocación del Syn Flood Attack desde Ansible

La implementación de este perfil consiste en realizar múltiples invocaciones a la primitiva de capa 2 `not_ended_tcp_connection`, que genera conexiones TCP incompletas (SYN + SYN/ACK). Esta primitiva se construye a su vez a partir de la primitiva de capa 1 `initialize_connection`, eliminando el último segmento del establecimiento de conexión.

El perfil permite parametrizar distintos aspectos del ataque, tales como la dirección IP y el puerto de la víctima, la cantidad de direcciones IP atacantes simuladas y los parámetros asociados al modelo temporal y a la semilla de generación.

Creación de los artefactos en el host objetivo

Cuando un usuario desea crear un artefacto de red, el mismo se verá reflejado en forma de archivo PCAP. Por lo que es importante que el usuario pueda especificar la ubicación final del mismo. Cabe aclarar que esta materialización del archivo PCAP no es realizada por las primitivas en si misma, dado que las mismas trabajan exclusivamente con listas de paquetes en memoria; en ningún momento escriben archivos al disco.

Cuando el instructor define un escenario en un playbook YAML, es el módulo ubicado en *library/* el que recibe los parámetros especificados. Este módulo actúa como adaptador: interpreta los parámetros, invoca la primitiva o perfil correspondiente de *module_utils/*, y recibe como resultado una lista de paquetes Scapy. Una vez obtenida dicha lista, es dicho módulo quién materializa la creación del PCAP. La herramienta que brinda Scapy para serializar listas de paquetes en un archivo PCAP se llama `wrpcap`. Sin embargo, para escribir el archivo, delega esa responsabilidad a un módulo utilitario implementado específicamente

para este propósito, que utiliza la herramienta de línea de comandos `mergecap` del proyecto Wireshark(The Wireshark Team, 2024).

Un mismo escenario puede generar tráfico de red a partir de múltiples primitivas o perfiles, cada uno responsable de producir distintos eventos o patrones de actividad. En consecuencia, todos los módulos deben poder escribir paquetes sobre una misma captura que represente el tráfico total del escenario.

Si en la ruta de destino ya existiera un archivo PCAP previo, una escritura directa con `wrpcap` lo sobreescribiría, destruyendo el tráfico ya almacenado. Para evitar esta situación, el módulo utilitario verifica si existe un archivo en la ruta indicada. En caso de que no exista, los paquetes se escriben directamente. Si el archivo ya está presente, los nuevos paquetes se escriben primero en un archivo temporal y posteriormente se invoca la herramienta `mergecap` para combinar ambos archivos en uno solo en la ruta objetivo, ordenando los paquetes por *timestamp*. De esta manera, múltiples módulos pueden escribir tráfico sobre la misma ruta de salida sin sobrescribir información previamente generada.

Dado que los módulos ubicados en el directorio *library/* son ejecutados por Ansible en el host al que se aplica el playbook, el archivo PCAP resultante queda almacenado directamente en dicho sistema.

5.6.3. Validación de Artefactos de Red

La validación de la evidencia de red se realizó en dos niveles: validación estructural de los paquetes generados y validación de patrones de ataque.

Para la validación estructural se utilizó Wireshark, que permite inspeccionar automáticamente la consistencia de los protocolos mediante su sistema de *Expert Information*. Este mecanismo reporta anomalías tales como paquetes malformados, inconsistencias de protocolo o errores de checksum. En particular, se verificaron correctamente los siguientes aspectos:

- **Integridad de paquetes**: Verificación de que los paquetes tienen checksums IP y TCP/UDP correctos.
- **Coherencia de conexiones TCP**: Análisis de los números de secuencia y acknowledgement para verificar que siguen una progresión lógica coherente. Se utilizó la funcionalidad `Follow TCP Stream` de Wireshark para confirmar que las conexiones TCP tienen un handshake válido (SYN, SYN-ACK, ACK) y que los datos transmitidos son reconstruibles.
- **Validez de protocolos de aplicación**: Los paquetes fueron diseccionados por Wireshark sin errores de parseo, confirmando que los payloads generados cumplen con los formatos especificados en sus respectivos RFCs.
- **Coherencia temporal**: Verificación de que los timestamps de los paquetes siguen una secuencia temporal realista, con intervalos apropiados entre paquetes que simulan latencia de red real.

Mientras que para validar que los patrones de ataque generados son detectables por sistemas de detección de intrusos, se utilizó Snort como IDS de

referencia (Cisco Systems, 2026). Snort es una herramienta estándar de la industria para detectar amenazas de red. La validación del ataque de denegación de servicio (DoS) se realizó de la siguiente manera:

1. **Configuración de reglas**: Snort incluye por defecto un conjunto extenso de reglas para detección de ataques, muchas de las cuales están desactivadas para optimizar el rendimiento en entornos de producción. Para esta validación se habilitaron específicamente las reglas asociadas a la detección de ataques de tipo DoS. En particular, la regla presentada en el Listado 5.11 identifica patrones compatibles con un ataque de *TCP SYN Flood*:

```
tcp any any -> any any (msg:"DOS TCP SYN Flood Attempt";
flags:S; detection_filter: track by_dst, count 500,
    seconds 3;
sid:254; rev:1;)
```

Listado 5.11: Regla de Snort utilizada para la detección de ataques de tipo TCP SYN Flood.

   Esta regla genera una alerta cuando se detectan más de 500 paquetes TCP con la bandera SYN dirigidos al mismo destino dentro de una ventana temporal de 3 segundos, lo cual es característico de un ataque de tipo SYN Flood.

2. **Análisis del PCAP**: posteriormente se ejecutó Snort en modo offline sobre los archivos PCAP generados por el perfil `generate_syn_flood` con 800 atacantes. De esta forma se verificó que el tráfico producido efectivamente coincide con el patrón esperado de un ataque de denegación de servicio.

   Cuando Snort detecta que una regla se cumple, genera una alerta por cada paquete que continúa satisfaciendo dicha condición. En el Listado 5.12 se muestra un fragmento de la salida obtenida durante el análisis del PCAP generado:

```
03/10-16:00:16.771312 [**] [1:254:1] "DOS TCP SYN Flood
    Attempt" [**]
[Priority: 0] {TCP} 194.210.37.157:2521 ->
    192.168.100.1:80

03/10-16:00:16.771912 [**] [1:254:1] "DOS TCP SYN Flood
    Attempt" [**]
[Priority: 0] {TCP} 69.15.142.91:56781 ->
    192.168.100.1:80

03/10-16:00:16.772512 [**] [1:254:1] "DOS TCP SYN Flood
    Attempt" [**]
[Priority: 0] {TCP} 94.42.58.135:24878 ->
    192.168.100.1:80
```

Listado 5.12: Fragmento de la salida de Snort al analizar el PCAP generado, mostrando alertas correspondientes al patrón de ataque SYN Flood.

Estas alertas evidencian que el tráfico generado por el perfil malicioso es correctamente identificado por el sistema de detección de intrusos, validando así que el patrón de ataque sintetizado reproduce las características observables de un ataque real.

Esta validación con Snort es particularmente relevante desde el punto de vista pedagógico, ya que los estudiantes pueden utilizar las mismas herramientas para detectar y analizar los ataques en sus ejercicios prácticos.

5.7. Generación de Escenarios

Para desplegar los escenarios siguiendo el paradigma de Tectonic, se deben definir dos componentes: un `base_config` con la configuración base de las máquinas virtuales y un `after_clone` encargado de materializar la lógica específica del escenario.

5.7.1. Archivo `base_config.yml`

La generación de evidencia DFIR introduce un conjunto de dependencias técnicas que deben estar presentes antes de ejecutar cualquier escenario. Estas dependencias incluyen bibliotecas Python para generación de artefactos digitales, herramientas de red y paquetes del sistema.

Por este motivo, todas las dependencias requeridas por el framework DFIR deben instalarse obligatoriamente durante la fase de construcción de la imagen base, es decir, dentro de `base_config.yml`. De esta manera, cualquier escenario que se despliegue posteriormente podrá ejecutarse sin requerir instalación adicional ni conectividad externa.

Entre las principales dependencias se encuentran: `Faker`, `Pillow`, `python-docx`, `openpyxl`, `cryptography`, `scapy`, `gcc`, `build-essential` y `wireshark-common`, entre otras.

Con el objetivo de simplificar la configuración y evitar duplicación de tareas, se creó el playbook centralizado `install_evidence_generation_deps.yml`, el cual encapsula toda la lógica de instalación, detección del sistema operativo y verificación posterior.

De este modo, el archivo `base_config.yml` únicamente debe incluir dicho playbook antes de levantar cualquier escenario DFIR:

```
1 - name: "Install evidence generation dependencies"
2   ansible.builtin.include_tasks: ../../../tectonic/ansible/
        playbooks/install_evidence_generation_deps.yml
```

Listado 5.13: Inclusión del playbook de dependencias en base_config.yml

Esta abstracción reduce la complejidad del archivo principal y garantiza que todas las dependencias necesarias se instalen de manera consistente.

El playbook implementa:

- Actualización del caché de paquetes según la familia del sistema operativo (Debian o RedHat/Fedora).
- Instalación de dependencias de compilación y herramientas del sistema.
- Actualización del gestor de paquetes de Python *pip*.
- Instalación de las bibliotecas Python requeridas.

En consecuencia, la responsabilidad de gestión de dependencias queda completamente desacoplada de la lógica del escenario. Una vez ejecutado `base_config.yml`, la máquina virtual generada queda preparada para ejecutar cualquier módulo DFIR durante la fase `after_clone`.

5.7.2. Archivo `after_clone.yml`

El archivo `after_clone.yml` se ejecuta automáticamente luego de que Tectonic clona la imagen base para cada grupo de estudiantes. En esta etapa no se instalan dependencias, sino que se invocan directamente los módulos de generación de evidencia integrados como extensiones de Ansible. Gracias a la integración realizada, los perfiles DFIR pueden ejecutarse como tareas nativas de Ansible. En la Sección 6.3 se expone un archivo `after_clone` concreto para el caso de estudio.

De esta forma, el `base_config` garantiza la preparación del entorno técnico, mientras que el `after_clone` implementa la lógica concreta del escenario forense.

Capítulo 6

Experimentación

El presente capítulo valida la solución desarrollada mediante un caso de estudio integrador que ejercita las capacidades de generación de artefactos implementadas a lo largo del trabajo. El capítulo se estructura en cuatro secciones: la definición y justificación del caso de estudio seleccionado, la descripción de la arquitectura del escenario desplegado, el detalle del proceso de generación de artefactos organizado en fases secuenciales, y finalmente la resolución paso a paso del ejercicio forense, que permite verificar que los artefactos producidos son técnicamente correctos y analizables con herramientas estándar de la industria.

6.1. Caso de Estudio Integrador

Para la selección del escenario a desarrollar para el caso de estudio se establecieron tres criterios principales: que tenga un fundamento en un incidente real, que permita integrar múltiples dimensiones del análisis forense (red y sistema de archivos), y que requiera la utilización de diversas primitivas desarrolladas de la solución propuesta.

Adicionalmente, se definió como objetivo pedagógico la recuperación de una bandera oculta en un artefacto específico, de modo que el estudiante debe reconstruir la secuencia del incidente para resolver el ejercicio.

A partir de estos criterios, se diseñó un escenario en el cual la bandera se encuentra contenida en un archivo cifrado y posteriormente eliminado del sistema. Esta decisión introduce dos desafíos técnicos complementarios: por un lado, la necesidad de identificar y recuperar un archivo borrado, por otro, la obligación de descifrar su contenido para acceder a la información final.

Con el fin de integrar también la dimensión de red, se planteó que la clave de cifrado no estuviera almacenada localmente, sino que pudiera inferirse a partir del análisis de una captura de tráfico. De este modo, el estudiante debe correlacionar evidencia de red con artefactos del sistema de archivos, poniendo en práctica el principio de transversalidad definido en el capítulo anterior.

Como marco conceptual del escenario se tomó como referencia el ataque

de ransomware WannaCry. WannaCry fue una campaña global de ransomware ocurrida en mayo de 2017, que afectó a más de 230.000 computadoras en aproximadamente 150 países (Mohurle y Patil, 2017; Akbanov, Vassilakis, y Logothetis, 2019). Este malware cifraba los archivos de las víctimas y exigía el pago de un rescate para proveer la clave de descifrado (Kharraz, Kirda, Robertson, Balzarotti, y Francillon, 2015).

En consecuencia, el escenario diseñado entrega al estudiante una captura de tráfico de red que contiene la descarga de binarios maliciosos y una simulación de ejecución remota, y una imagen de disco donde los archivos han sido cifrados, incluyendo un archivo específico que contiene la bandera, el cual se encuentra cifrado y eliminado en la misma imagen.

El objetivo final consiste en reconstruir la secuencia del ataque, identificar la clave de cifrado a partir del análisis del tráfico de red, recuperar el archivo eliminado y descifrar su contenido para obtener la bandera.

De esta manera, el caso de estudio integra análisis de red, análisis de sistema de archivos y correlación de evidencia, permitiendo evaluar de forma práctica las capacidades de generación de artefactos desarrolladas en este trabajo.

6.2. Arquitectura del escenario

El escenario implementado despliega una infraestructura de dos máquinas virtuales diseñada para proporcionar un entorno realista de investigación forense.

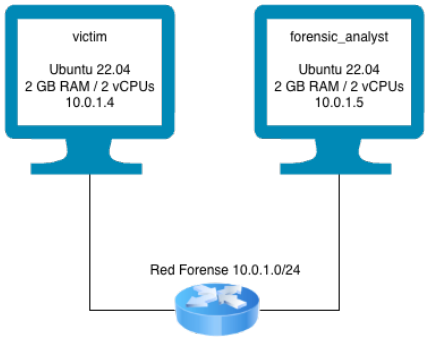


Figura 6.1: Topología de red del laboratorio forense

- **Máquina víctima**: Sistema donde se ejecuta la simulación del ataque. Durante el despliegue, esta máquina genera todos los artefactos forenses (archivo de captura de tráfico e imagen de disco) que luego serán analizados.
- **Estación de análisis forense**: Máquina de trabajo donde los estudiantes realizan la investigación. Los artefactos generados en la víctima se transfieren automáticamente a esta estación. La cual posee herramientas

de análisis forense como Wireshark, Autopsy, y The Sleuth Kit, que los estudiantes pueden utilizar.

6.3. Proceso de Generación de Artefactos

El playbook `after_clone.yml` orquesta la generación completa de evidencia durante la fase post-clonado de Tectonic. El proceso se ejecuta en la máquina víctima y se divide en seis fases secuenciales que se presentan a continuación.

6.3.1. Fase 1: Generación de Tráfico de Red

Nuevo Perfil de Red

Se generó un nuevo perfil en la vertiente de red, con el fin de generar los artefactos para el caso de experimentación. El perfil se llama `generate_wannacry_attack` y puede recibir los siguientes parámetros: `victim_ip` el cuál permite especificar la dirección IP de la víctima del ataque, `attacker_ip` (opcional, en caso de que no se especifique lo genera la Capa 0) que permite especificar la dirección IP de la máquina atacante e `ip_malicious_server` (opcional, en caso de que no se especifique lo genera la Capa 0) que permite especificar la dirección IP del servidor de donde se descargan los binarios de ataque.

Generación

La primera fase utiliza el módulo de capa 3 `generate_wannacry_attack` para generar un archivo PCAP que simula las distintas etapas del ataque. En el código 6.1 se puede ver la declaración del perfil en el *playbook* Ansible.

```
- name: "[NETWORK] Generate WannaCry attack traffic"
  generate_wannacry_attack:
    victim_ip: "172.16.0.100"
    output_path: "{{ pcap_output }}"
    seed: "{{ faker_seed }}"
  register: network_attack
```

Listado 6.1: Tarea Ansible de generación de tráfico de red

El módulo genera tráfico estructurado en tres etapas que replican el comportamiento documentado de varias infecciones por ransomware (Akbanov y cols., 2019; Kharraz y cols., 2015):

1. **Fase de reconocimiento**: Se simula una comunicación TCP en texto plano entre el atacante y la víctima, donde el atacante identifica el sistema objetivo y exfiltra un archivo `creds.txt` que contiene credenciales de acceso al servidor Samba. Samba es una implementación de código abierto del protocolo SMB/CIFS (*Server Message Block / Common Internet File*

System) para sistemas Unix y Linux, que permite compartir archivos e impresoras entre sistemas heterogéneos (Carter, 2003; The Samba Team, 2024). SMB es el protocolo nativo para compartir recursos en entornos Windows.

Esta fase replica el comportamiento de reconocimiento previo a la infección. En la Figura 6.2 se puede observar un paquete de ejemplo de que realiza un `cat` a un archivo llamado *credentials.txt*, tanto los paquetes previos como los posteriores al mencionado tienen el fin de identificar al atacante en el sistema y de hurtar credenciales.

2. **Fase de descarga**: Con las credenciales obtenidas, el atacante hace descargar a la víctima los binarios maliciosos desde un servidor web externo. El tráfico generado incluye la resolución DNS del dominio del servidor *wanna.srvr* (el cual se encuentra hardcodeado y no existe verídicamente), seguido de las peticiones HTTP de descarga de los componentes del ransomware, replicando el mecanismo de distribución de WannaCry mediante servidores de distribución de malware (Mohurle y Patil, 2017).

3. **Fase de Ejecución de los Binarios**: Se genera tráfico de comunicación entre el atacante y la víctima para la ejecución remota de los binarios. Como desafío forense, la clave de cifrado AES utilizada en la fase posterior de sistema de archivos se encuentra embebida en los comandos visibles en la captura de tráfico. Por lo que el estudiante debe analizar la captura para obtener la clave y posteriormente descifrar el archivo que está en la imagen de disco y recuperar la bandera oculta.

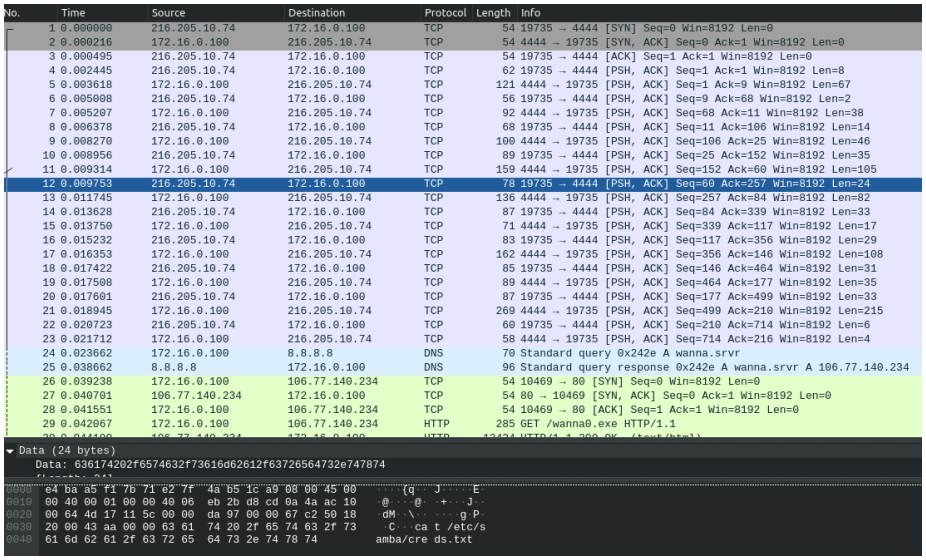


Figura 6.2: Inicio de captura de tráfico generada para el escenario de experimentación

6.3.2. Fase 2: Preparación del sistema de archivos forense

La evidencia de sistema de archivos se genera sobre una imagen de disco independiente del sistema operativo de la máquina virtual, en lugar de modificar directamente su sistema de archivos en ejecución. Esta decisión está motivada por la limitación técnica descripta en la sección 5.5.3: la primitiva estándar de eliminación no garantiza la recuperabilidad del artefacto en la infraestructura subyacente, mientras que operar con la primitiva de bajo nivel sobre un sistema de archivos montado y activo puede dejarlo en un estado inconsistente. La imagen de disco separada resuelve ambos problemas y aporta además beneficios pedagógicos concretos: la evidencia queda contenida en un archivo portable sin afectar el funcionamiento del sistema operativo subyacente; la imagen puede copiarse y analizarse en cualquier sistema con soporte ext4, facilitando su distribución. El proceso de preparación, orquestado por Ansible, ejecuta los siguientes pasos en orden: elimina cualquier imagen previa en la ruta configurada, crea una nueva imagen inicializada en cero, la formatea con ext4, libera cualquier loop device que haya quedado asociado a esa ruta en ejecuciones anteriores, asocia la nueva imagen a un loop device disponible, y finalmente monta el sistema de archivos en el punto de montaje configurado. Un loop device es un dispositivo de bloque virtual —abstracción de software que emula un disco físico— que permite tratar un archivo regular como si fuera un disco, habilitando operaciones de sistema de archivos estándar sobre él.

6.3.3. Fase 3: Creación de la estructura de directorios

Una vez montada la imagen, se crea la estructura de directorios que simulará el servidor de archivos de la víctima:

```
- name: "[FS] Create target directory /srv/samba/share
    inside disk image"
  file:
    path: "{{ mount_point }}/srv/samba/share"
    state: directory
    mode: "0755"
```

Listado 6.2: Creación del directorio objetivo dentro de la imagen

La ruta `/srv/samba/share` replica la estructura típica de un servidor de archivos compartidos en entornos corporativos Windows/Linux.

6.3.4. Fase 4: Simulación del ransomware

En esta fase se describe el módulo `execute_ransomware_profile`, implementado para simular un ataque de ransomware sobre un sistema de archivos objetivo, y su integración en el escenario de experimentación.

Nuevo perfil de Sistema de Archivos

El perfil `execute_ransomware_profile` acepta los siguientes parámetros: `profile`, que indica el perfil de ransomware a simular (`wannacry` o `custom`); `target_directory`, el directorio destino donde se ejecutará la simulación; y los opcionales, que son: `profile_config`, que permite sobrescribir parámetros del perfil base cuando se usa el modo `custom`; `file_count`, para definir la cantidad de archivos a generar; `faker_seed`, semilla para reproducibilidad de resultados; `force_rerun`, que fuerza la re-ejecución ignorando el control de idempotencia; `encryption_key`, clave AES-256 específica en formato hexadecimal; `enable_encryption`, para habilitar o deshabilitar el cifrado efectivo de archivos; `create_ransom_note`, que controla la creación de la nota de rescate; y `deletion_ratio`, proporción de archivos a eliminar aleatoriamente, simulando el comportamiento de WannaCry sobre respaldos y copias de seguridad.

Generación

El corazón del escenario es la ejecución del perfil de ransomware WannaCry mediante el módulo de capa 3:

```
- name: "[FILESYSTEM] Execute ransomware profile"
  execute_ransomware_profile:
    profile: wannacry
    target_directory: "{{ mount_point }}/srv/samba/share"
    file_count: "{{ file_count }}"
    faker_seed: "{{ faker_seed }}"
    deletion_ratio: "{{ deletion_ratio }}"
    force_rerun: "{{ force_rerun }}"
  register: ransomware_result
```

Listado 6.3: Ejecución del perfil de ransomware WannaCry

Dado que la infraestructura opera sobre Linux y WannaCry fue diseñado para atacar sistemas Windows mediante SMB, la simulación utiliza Samba como puente de compatibilidad. Samba implementa el protocolo SMB/CIFS permitiendo que el directorio {{`mount_point`}}`/srv/samba/share` emule un recurso compartido de red Windows, replicando el vector de propagación lateral característico del ataque original (The Samba Team, 2024; Carter, 2003). Cabe destacar que las vulnerabilidades específicas de SMBv1 que WannaCry explotaba no están presentes en Samba (The Samba Team, 2017), por lo tanto, la simulación genera el tráfico de red característico de la explotación de vulnerabilidad, y los artefactos de sistema de archivos resultantes del cifrado, pero no replica la explotación real de la vulnerabilidad.

El módulo orquesta automáticamente las siguientes operaciones sobre el directorio Samba:

1. **Generación de archivos víctima**: Crea archivos con una distribución

realista de tipos de documentos corporativos, utilizando la semilla configurada.

2. **Cifrado de archivos**: Cada archivo es cifrado con un algoritmo similar al utilizado por el ransomware.

3. **Eliminación de originales**: Los archivos sin cifrar son eliminados, simulando el comportamiento destructivo del ransomware real.

4. **Notas de rescate**: Se generan archivos `@Please_Read_Me@.txt` en múltiples directorios con el mensaje característico de WannaCry solicitando pago en Bitcoin.

5. **Simulación de limpieza**: El porcentaje indicado en `deletion_ratio` de los archivos cifrados es eliminado, dejando rastros recuperables en el sistema de archivos para replicar el comportamiento de WannaCry sobre archivos en el servidor compartido.

6. **Generación de ejecutables maliciosos**: Se generan archivos PE (*Portable Executable*) con firmas características de WannaCry. Un archivo PE es el formato binario estándar de ejecutables en sistemas Windows, cuya estructura incluye cabeceras que describen el tipo de imagen, secciones de código y datos, y metadatos de importación y exportación. La generación de estos archivos responde a la necesidad de que el escenario sea identificable como malicioso por herramientas de análisis reales: los patrones de bytes embebidos en la sección `.text` del ejecutable fueron extraídos de las reglas YARA publicadas por ReversingLabs para la detección de WannaCry (ReversingLabs, 2017). La validez forense de este enfoque fue verificada enviando los archivos generados a VirusTotal, donde fueron efectivamente identificados como ransomware por múltiples motores de detección(Google LLC, 2025).

6.3.5. Fase 5: Generación del archivo bandera

Esta fase implementa el desafío forense central del escenario: un archivo con un mensaje secreto que el estudiante debe recuperar y descifrar. El proceso consta de cinco etapas: creación del archivo, cifrado con AES-256-CBC, sincronización del sistema de archivos, desmontaje de la imagen y eliminación forense mediante `debugfs` para que el borrado sea recuperable:

Antes del desmontaje se ejecuta el comando `sync`, que fuerza la escritura de todos los datos pendientes en el buffer del kernel hacia el sistema de archivos subyacente. Linux mantiene un caché de escritura en memoria por razones de rendimiento, por lo que sin esta instrucción el archivo `flag.txt.WNCRY` podría no estar físicamente escrito en la imagen de disco al momento del desmontaje, comprometiendo su recuperabilidad posterior. Solo una vez garantizada la persistencia del archivo en el sistema de archivos se procede a desmontarlo y a ejecutar la eliminación forense.

```yaml
- name: "[FLAG] Create flag file with secret message"
  generate_file:
    path: "{{ mount_point }}/srv/samba/share/flag.txt"
    extension: txt
    content: "Encontraste la flag"

- name: "[FLAG] Encrypt flag file with AES-256-CBC (.WNCRY
    extension)"
  encrypt_files_aes:
    files:
      - "{{ mount_point }}/srv/samba/share/flag.txt"
    key: "0123456789
        ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
        "
    encrypted_extension: "WNCRY"
    keep_original: false

- name: "[DISK] Sync filesystem before unmounting"
  command: sync
  changed_when: false

- name: "[DISK] Unmount filesystem and remove fstab entry"
  mount:
    path: "{{ mount_point }}"
    state: absent

- name: "[FLAG] Delete encrypted flag file with debugfs"
  delete_file_debugfs:
    filename: "flag.txt.WNCRY"
    device: "{{ loop_device }}"
    directory: "/srv/samba/share"
```

Listado 6.4: Generación y eliminación forense del archivo flag

Esta eliminación utiliza `debugfs` para marcar el inodo del archivo como eliminado directamente sobre la partición desmontada. Este accionar está justificado en la Sección 5.5.3.

6.3.6. Fase 6: Limpieza del entorno

Una vez generada toda la evidencia, se desvincula el dispositivo loop para liberar la imagen de disco. Esta fase garantiza que la imagen de disco quede correctamente cerrada y lista para ser distribuida a los estudiantes como evidencia forense.

6.4. Solución del Escenario

Al finalizar el despliegue del escenario, el estudiante dispone de dos artefactos forenses: la captura de tráfico generada en la Fase 6.3.1 y la imagen de disco forense generada en la Fase 6.3.5. El análisis comienza por la captura de red, cuyo objetivo en este contexto además de reconstruir la totalidad del incidente, es extraer la clave de cifrado AES embebida en el paquete que ordena la ejecución remota de los binarios. Como se observa en la Figura 6.3, dicha clave es visible en texto plano en la capa de aplicación del paquete correspondiente.

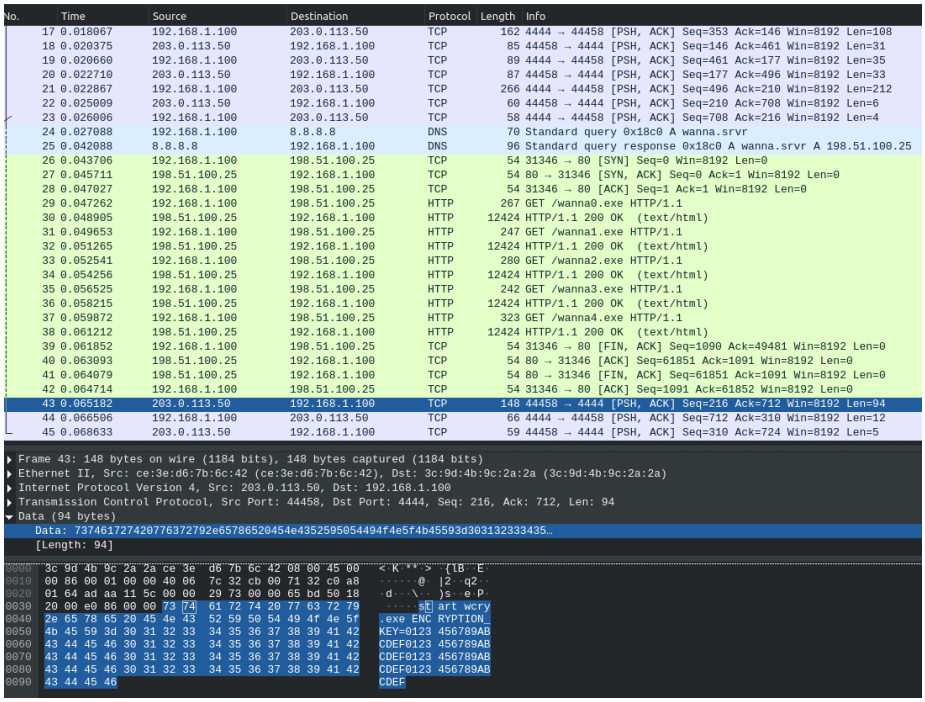


Figura 6.3: Final de la captura de tráfico generada para la experimentación

Con la clave obtenida, el análisis continúa sobre la imagen de disco mediante Autopsy (Basis Technology, 2025). Tras importar la imagen forense, Autopsy detecta automáticamente la partición ext4 y habilita la navegación del sistema de archivos, incluyendo los archivos eliminados. Navegando a `/srv/samba/share` desde el panel *File Analysis*, se observa el estado del sistema comprometido: archivos cifrados con extensión `.WNCRY`, ejecutables con nombres característicos de WannaCry (`mssecsvc.exe`, `tasksche.exe`, `wannacry.exe`) y la nota de rescate `@Please_Read_Me@.txt`. Entre estos artefactos, el archivo `flag.txt.WNCRY` aparece resaltado en rojo, indicando que fue eliminado pero sus bloques de datos permanecen recuperables en el espacio no asignado. Seleccionando el archivo y ejecutando *Export* desde el panel inferior, se descarga su contenido binario para su posterior análisis (Figura 6.4).

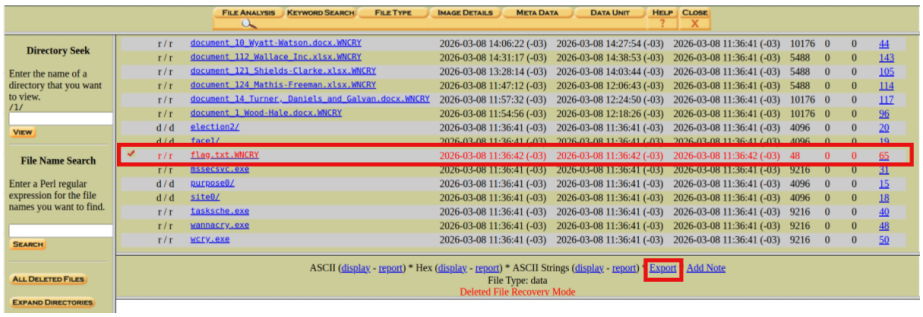


Figura 6.4: Archivo eliminado recuperado en `/srv/samba/share`

El archivo exportado (el que contiene la bandera) está cifrado con AES-256-CBC, cuyo formato antepone al texto cifrado un vector de inicialización (IV) de 16 bytes aleatorios. Para extraerlo, se carga el archivo en CyberChef (GCHQ, 2025) con la operación *To Hex*: los primeros 32 caracteres hexadecimales del output corresponden al vector de inicialización. Con este valor y la clave obtenida del tráfico de red, se construye la receta de descifrado: la operación *Drop bytes* (Start: 0, Length: 16) elimina el vector de inicialización del input, y *AES Decrypt* en modo CBC con la llave del cifrado y el vector de inicialización revela el mensaje oculto, dando por resuelto el escenario (Figura 6.5).

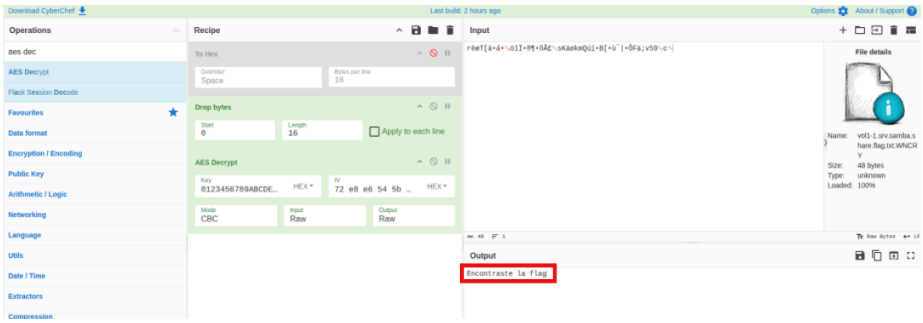


Figura 6.5: Descifrado exitoso de la bandera

Capítulo 7

Conclusiones y Trabajo Futuro

El presente capítulo expone una síntesis de los resultados alcanzados durante el desarrollo del proyecto, evaluando el cumplimiento de los objetivos planteados inicialmente. Se describen las contribuciones conceptuales y técnicas realizadas. Finalmente, se analizan las dificultades técnicas superadas y se propone una hoja de ruta para el trabajo futuro, orientada a expandir las capacidades forenses del sistema.

7.1. Resultados alcanzados

Este proyecto logró diseñar una arquitectura capaz de soportar las tres vertientes fundamentales del análisis forense digital: red, sistema de archivos y memoria RAM. La solución permite que un instructor defina escenarios con un nivel de granularidad arbitrario: desde la especificación campo por campo de un artefacto atómico, hasta la creación de entornos complejos mediante una única tarea de alto nivel.

7.1.1. Aportes conceptuales

El principal aporte conceptual es la definición del modelo de abstracción de tres capas. Este marco teórico permite desacoplar la complejidad técnica de la generación de evidencia de la lógica del escenario. Al estandarizar la forma en que se declaran los perfiles de comportamiento, se ha facilitado que docentes puedan construir laboratorios forenses técnicamente correctos sin necesidad de dominar las herramientas de bajo nivel, logrando que puedan extender los perfiles en los distintos ejes usando las primitivas brindadas por capas inferiores.

7.1.2. Aportes técnicos y estructurales

Se reestructuro la arquitectura de Ansible dentro de Tectonic. Se transformó un sistema de configuración estático en un ecosistema modular y extensible.

Mediante la implementación de un sistema de descubrimiento dinámico de componentes, se habilitó el uso de módulos personalizados, utilidades compartidas y filtros de Jinja2.

Se implementaron exitosamente dos ejes principales: evidencia de red y evidencia de sistema de archivos. La validación técnica confirmó que la evidencia generada es técnicamente correcta, reproducible, y analizable con herramientas forenses estándar de la industria.

7.1.3. Cumplimiento de Objetivos

Se ha verificado el cumplimiento de los objetivos específicos definidos al inicio del trabajo:

- **Investigación del estado del arte**: Se realizo un estudio de la disciplina de forensia digital y respuesta a incidentes, y además se analizó documentación académica complementando con la documentación comercial disponible en la web. Esto concluyó exitosamente, resultando en un análisis de técnicas de generación de artefactos forenses y su aplicación en cyber ranges.

- **Diseño de solución**: Se diseñó una arquitectura de tres capas que cumple con los requerimientos identificados, proporcionando flexibilidad y escalabilidad.

- **Implementación de prototipos**: Se implementó parcialmente, cubriendo evidencia de red y sistema de archivos con funcionalidad suficiente para demostrar viabilidad y validar el enfoque.

- **Validación mediante caso de estudio**: Se desarrolló y validó un escenario de ataque WannaCry que integra ambas vertientes implementadas.

7.1.4. Cumplimiento de Requerimientos

Lo detallado a lo largo del informe, acompañado por el caso de estudio presentado permite validar de forma práctica los requerimientos establecidos en la Sección 3.3.

Generación sintética de evidencia. Se considera cumplido el criterio de evaluación establecido. No sólo los artefactos utilizados en el escenario de experimentación fueron verificados mediante herramientas forenses estándar, sino que se comprobó la correcta generación de todos los tipos de artefactos que la solución permite producir. En todos los casos, los resultados fueron estructuralmente válidos.

Generación de actividad de fondo. El requerimiento se considera satisfecho. En el escenario de experimentación, además del artefacto objetivo (la bandera), se generaron múltiples archivos adicionales que simulan actividad legítima. En el ámbito de red, se implementaron primitivas específicas destinadas a la generación de tráfico de fondo. Para que un escenario contenga mayor cantidad de actividad de fondo que actividad maliciosa, el instructor únicamente debe invocar las primitivas correspondientes desde la definición en YAML.

Control de marcas de tiempo. Este requerimiento puede considerarse cumplido parcialmente. En los artefactos de red, el *timestamp* base y los valores temporales individuales pueden especificarse explícitamente, generándose los paquetes de acuerdo con dicha configuración. En el caso del sistema de archivos, por restricciones propias del sistema operativo y de las herramientas utilizadas, no fue posible modificar el tiempo de creación de los archivos. No obstante, se propusieron alternativas técnicas para abordar esta limitación, y sí se logró controlar los tiempos de acceso y modificación.

Transversalidad. El requerimiento se encuentra satisfecho y quedó evidenciado en el caso de estudio. El escenario integra artefactos de red y de sistema de archivos, requiriendo su correlación para la resolución completa del ejercicio.

Escalabilidad. Se considera cumplido. El uso del motor de plantillas Jinja y sus filtros permite instanciar múltiples variantes del mismo escenario mediante la parametrización de variables, evitando la repetición manual de configuraciones. En cuanto a desempeño, los tiempos de generación se mantienen alineados con otros escenarios de Tectonic. La creación de la imagen base mediante Docker, en un entorno de prueba con un equipo de cinco núcleos y 8 GB de memoria RAM, requiere aproximadamente 1 minuto y 30 segundos para una instancia y 1 minuto y 45 segundos para cinco instancias. El despliegue posterior demanda aproximadamente 40 segundos para una instancia y 1 minuto y 20 segundos para cinco instancias.

Compatibilidad con Tectonic. El requerimiento se considera cumplido. El escenario desarrollado puede desplegarse directamente utilizando el flujo estándar de Tectonic, sin requerir pasos manuales adicionales ni modificaciones externas.

Generación de contenido. Este requerimiento se encuentra parcialmente cumplido. El contenido de un archivo puede especificarse explícitamente por el instructor o generarse de forma pseudoaleatoria mediante Faker cuando no se requiere control directo sobre el mismo. Lo que quedó fuera del alcance fue la generación de contenido con mayor grado de realismo, que se puede por ejemplo mediante modelos de lenguaje (Antal y Miclea, 2024), visión por computadora (Schmidt y cols., 2023) u otros métodos que no comtemplamos.

7.2. Dificultades encontradas

Durante el desarrollo del proyecto se enfrentaron varias dificultades técnicas y de diseño:

Abstracción del Modelo de Capas: Hallar un equilibrio entre el control

granular y la simplicidad de uso representó una dificultad conceptual mayor. Se requirieron múltiples iteraciones para diseñar una arquitectura de tres capas que permitiera a un instructor especificar detalles mínimos (Capa 1) sin sacrificar la capacidad de desplegar escenarios masivos con configuraciones sencillas (Capa 3). El ejemplo implementado ilustra esta flexibilidad: el mismo YAML especifica tanto el comportamiento macro del ransomware (cifrado masivo de 150 archivos) como detalles forenses precisos (un archivo particular eliminado que los estudiantes deben recuperar), integrados ambos en la misma imagen de disco. Esta capacidad de combinación también habilita otros escenarios híbridos, como insertar paquetes maliciosos (Capa 1) específicos en capturas de tráfico de red legítima (Capa 3), o mezclar navegación web normal con comunicaciones de comando y control.

Lógica de Temporalidad Global: Diseñar un sistema de *timestamps* que fuera coherente a través de todas las capas de abstracción fue una tarea compleja. El desafío consistió en permitir que el tiempo fluyera de forma natural en escenarios dinámicos sin obligar al usuario a especificar manualmente cada marca de tiempo, manteniendo la causalidad entre eventos.

Determinismo y Reproducción: Lograr que una generación dinámica fuera reproducible requirió el diseño de un esquema de semillas. Se debió resolver el problema de los valores duplicados en generaciones masivas mediante la implementación de una lógica de índices ($seed + index$), asegurando variabilidad interna pero determinismo absoluto entre ejecuciones del mismo escenario.

Generación de tráfico de red realista: Generar tráfico de red técnicamente correcto con Scapy requiere conocimiento detallado de protocolos y gestión cuidadosa del estado de conexiones TCP. Se invirtió tiempo considerable en implementar el manejo de números de secuencia, ventanas, y flags TCP para garantizar que las capturas generadas sean válidas. Se implementó un sistema de persistencia de estado para que los flujos de red fueran técnicamente correctos y no generaran errores en herramientas de análisis como Wireshark.

Integración con Tectonic: Aunque Tectonic proporcionaba mecanismos de extensión bien documentados, la plataforma no estaba diseñada para soportar módulos o filtros personalizados de Ansible que era lo que se precisaba. Esto obligó a realizar una modificación en el núcleo (`ansible.py`) para habilitar el descubrimiento de componentes.

Validez Forense del Borrado: Implementar la eliminación de archivos presentó dificultades debido a la dependencia del sistema operativo y del sistema de archivos subyacente. Se decidió implementar el uso de llamadas estándar del sistema (`os.remove`), que borran metadatos en sistemas modernos, y herramientas de bajo nivel como `debugfs` para manipular inodos directamente y garantizar que la evidencia fuera recuperable mediante técnicas de *data carving*.

7.3. Trabajo futuro

El trabajo realizado establece una base sólida para extensiones y mejoras futuras. A continuación se detallan algunas áreas de trabajo futuro:

7.3.1. Completar la vertiente de memoria, emails y logs de autenticación

La implementación de evidencia de memoria representa la extensión más directa del trabajo actual. La arquitectura de tres capas ya contempla esta vertiente, por lo que la implementación consistiría en desarrollar primitivas de capa 1 para generar procesos con características específicas, orquestadores de capa 2 para generar conjuntos de procesos, y perfiles de capa 3 para escenarios completos.

También de forma análoga es valioso pensar las vertientes de emails y logs de autenticación que por asunto del tiempo para implementación quedó por fuera del alcance del trabajo actual.

7.3.2. Mejoras en metadatos y realismo

- **Metadatos avanzados**: Implementar generación de metadatos más sofisticados en documentos Office (historial de revisiones, tiempo total de edición), imágenes (coordenadas GPS, información de cámara), y archivos ejecutables (firmas digitales, información de compilación).

- **Control temporal avanzado**: Desarrollar mecanismos para manipular timestamps de archivos de forma más granular, posiblemente mediante manipulación directa de inodos cuando el sistema de archivos no está montado, o mediante técnicas de virtualización del tiempo del sistema.

- **Profundidad semántica en protocolos de aplicación**: Las bibliotecas de datos sintéticos actuales (como Faker) no ofrecen soporte nativo para generar *payloads* complejos. Como resultado, el ruido generado en HTML, CSS y JavaScript es válido sintácticamente pero limitado en contenido. Una línea de trabajo futuro consiste en integrar motores de plantillas más sofisticados.

7.3.3. Soporte para múltiples plataformas

La implementación actual se centra en sistemas Linux (específicamente Ubuntu) porque Tectonic no soporta fácilmente Windows y Mac. Extender el soporte a estos dos sistemas operativos requeriría:

- Adaptar las primitivas de sistema de archivos para trabajar con NTFS (Windows) y APFS/HFS+ (macOS).

- Implementar manipulación de metadatos específicos de cada plataforma (por ejemplo, timestamps de NTFS, streams alternativos, journaling de macOS).

- Desarrollar primitivas para generar artefactos específicos de cada plataforma (registro de Windows, plist files de macOS).

7.3.4. Optimización y rendimiento

Para escenarios que generen grandes volúmenes de evidencia (miles de archivos o millones de paquetes), se pueden implementar optimizaciones:

- Paralelización de la generación utilizando multiprocessing de Python.
- Generación incremental que permita reanudar la generación si se interrumpe.
- Compresión inteligente de artefactos generados para reducir tiempos de distribución.

7.3.5. Validación y testing automatizado

Desarrollar una suite de tests automatizados que valide la corrección técnica de la evidencia generada:

- Tests unitarios para cada primitiva que verifiquen que los artefactos generados cumplen con especificaciones técnicas (checksums correctos, formatos válidos, metadatos coherentes).
- Tests de integración que validen escenarios completos end-to-end.
- Tests de regresión que garanticen que cambios futuros no rompan funcionalidad existente.

7.4. Reflexiones finales

Este proyecto demostró que es técnicamente viable integrar capacidades de generación automatizada de evidencia forense en un cyber range existente, proporcionando una solución que apunta a reducir la carga operativa de los instructores, generando artefactos técnicamente correctos y analizables con herramientas estándar.

La arquitectura de tres capas propuesta busca un equilibrio entre flexibilidad y simplicidad de uso, permitiendo que instructores con diferentes niveles de expertise técnico creen escenarios apropiados para sus necesidades educativas.

El trabajo realizado establece una base para el desarrollo futuro de capacidades más avanzadas, y demuestra el valor de extender cyber ranges más allá de actividades de ataque y defensa hacia el dominio de análisis forense digital y respuesta a incidentes.

La validación realizada con el escenario WannaCry confirma que la solución es apropiada para su uso en contextos educativos, proporcionando a los estudiantes experiencia práctica con herramientas y técnicas estándar de la industria en un entorno seguro y controlado.

Referencias

Akbanov, M., Vassilakis, V. G., y Logothetis, M. D. (2019). Wannacry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms. *Journal of Telecommunications and Information Technology*(1), 113–124. doi: 10.26636/jtit.2019.130218

(Alex), J. A. C., y Pillow Contributors. (2025). *Pillow: Python imaging library*. https://pillow.readthedocs.io/. (Librería para procesamiento y manipulación de imágenes en Python. Visitado 2026)

Antal, M., y Miclea, L. (2024). Re-imagen: Generating coherent background activity in synthetic scenario-based forensic datasets using large language models. En *Dfrws 2024*. Descargado de https://dfrws.org/wp-content/uploads/2024/10/Re-imagen-Generating-Coherent-Background-Activity-in-Synthetic-Scenario-Based-Forensic-Datasets-Using-Large-Language-Models-1.pdf

Basis Technology. (2025). *Autopsy: Digital forensics platform*. https://www.autopsy.com/. (Plataforma de código abierto para análisis forense digital basada en The Sleuth Kit. Visitado 2026)

Bejtlich, R. (2013). *The practice of network security monitoring: Understanding incident detection and response*. No Starch Press.

Ben Kiki, O., Evans, C., y Ingerson, B. (2021, October). Yaml ain't markup language (yaml) version 1.2.2 [Manual de software informático]. Descargado de https://yaml.org/spec/1.2.2/ (Revision 1.2.2)

Biondi, P., y Scapy Community. (2025). *Scapy: Packet crafting for python2 and python3*. https://scapy.readthedocs.io/. (Librería para manipulación y análisis de paquetes de red a bajo nivel. Visitado 2026)

Canny, S., y python-docx Contributors. (2013). *python-docx documentation*. https://python-docx.readthedocs.io/. (Librería para crear y modificar archivos Microsoft Word (.docx). Visitado 2026)

Carrier, B. (2005). *File system forensic analysis*. Addison-Wesley.

Carter, G. (2003). *Ldap system administration*. O'Reilly Media. (Incluye capítulos sobre integración de Samba con servicios de directorio y compartición de archivos en redes heterogéneas)

Casey, E. (2011). *Digital evidence and computer crime* (3rd ed.). Academic Press.

Casey, E., Richard, A., y James, J. M. (2014). *Handbook of digital forensics and investigation*. Waltham, MA: Academic Press.

Cisco Systems. (2026). *Snort – network intrusion detection & prevention system.* Descargado de https://www.snort.org/ (Visitado 2026)

Cybersecurity and Infrastructure Security Agency. (2024). *Cybersecurity training & exercises.* https://www.cisa.gov/cybersecurity-training-exercises. (Programas de entrenamiento gubernamental en respuesta a incidentes mediante cyber ranges. Visitado 2026)

DFRWS. (2019). *Memory forensics as triage analysis.* https://dfrws.org/presentation/memory-forensics-as-triage-analysis-2/.

Du, X., Hargreaves, C., Sheppard, J., y Scanlon, M. (2021). Tracegen: User activity emulation for digital forensic test image generation. *Forensic Science International: Digital Investigation*, *38*, 301133. (DFRWS 2021 APAC - Proceedings of the First Annual DFRWS APAC) doi: 10.1016/j.fsidi.2021.301133

Elastic N.V. (2025). *Elastic security documentation.* https://www.elastic.co/security. (Plataforma de análisis y monitoreo de seguridad con capacidades SIEM y EDR. Visitado 2026)

Faraglia, D., y Faker Contributors. (2025). *Faker: Python package that generates fake data.* https://faker.readthedocs.io/. (Librería para generación de datos sintéticos (nombres, emails, direcciones, etc.). Visitado 2026)

Fortinet. (2024). *2024 cybersecurity skills gap global research report.* https://www.fortinet.com/content/dam/fortinet/assets/reports/2024-cybersecurity-skills-gap-report.pdf. (Estudio global sobre la brecha de habilidades en ciberseguridad y su impacto en las organizaciones. Visitado 2026)

Gazoni, E., y Clark, C. (2024). *openpyxl documentation.* https://openpyxl.readthedocs.io/. (Librería para leer y escribir archivos Excel (.xlsx, .xlsm). Visitado 2026)

GCHQ. (2025). *Cyberchef: The cyber swiss army knife.* https://gchq.github.io/CyberChef/. (Aplicación web para análisis y decodificación de datos en investigaciones forenses. Visitado 2026)

Google LLC. (2025). *Virustotal: Free online virus, malware and url scanner.* https://www.virustotal.com/. (Servicio en línea para el análisis de archivos y URLs mediante más de 70 motores antivirus. Subsidiaria de Google desde 2012. Visitado 2026)

Grupo de Seguridad Informática (GSI), FING, UDELAR. (2025). *Tectonic: Cyber range.* https://www.fing.edu.uy/inco/grupos/gsi/project/tectonic/. (Visitado 2026)

Göbel, T., Breitinger, F., y Baier, H. (2024). Data for digital forensics: Why a discussion on "how realistic is synthetic dataïs dispensable. *Forensic Science International: Digital Investigation*, *52*, 301882. (Análisis exhaustivo sobre propiedades de conjuntos de datos forenses, destacando la ausencia de wear-and-tear y background noise en datos sintéticos) doi: 10.1016/j.fsidi.2024.301882

Göbel, T., Schäfer, T., Hachenberger, J., Türr, J., y Baier, H. (2020). A novel approach for generating synthetic datasets for digital forensics. En *Advances in digital forensics xvi* (pp. 73–93). Springer. (Presenta el framework

hystck para generación automática de tráfico de red y artefactos de sistema operativo sintéticos con ground truth) doi: 10.1007/978-3-030-56223-6_5

Harichandran, V. S., Walnycky, D., Baggili, I., y Breitinger, F. (2016). Cufa: A more formal definition for digital forensic artifacts. *Digital Investigation*, *18*, S125-S137. Descargado de https://www.sciencedirect.com/science/article/pii/S1742287616300366 doi: https://doi.org/10.1016/j.diin.2016.04.005

HashiCorp. (2025a). *Packer documentation.* https://developer.hashicorp.com/packer. (Herramienta para la creación automatizada de imágenes de máquinas virtuales y contenedores para múltiples plataformas. Visitado 2026)

HashiCorp. (2025b). *Terraform documentation.* https://developer.hashicorp.com/terraform. (Herramienta de Infraestructura como Código para definir y aprovisionar recursos de infraestructura de forma declarativa. Visitado 2026)

ISACA. (2024, October). *State of cybersecurity 2024* (Inf. Téc.). ISACA. Descargado de https://www.isaca.org/resources/reports/state-of-cybersecurity-2024 (Dato citado en el resumen público: https://www.isaca.org/resources/news-and-trends/newsletters/atisaca/2024/volume-19/stress-levels-on-the-rise-for-cybersecurity-professionals)

Johansen, G. (2020). *Digital forensics and incident response: An intelligent way to respond to cyber attacks*. Birmingham, UK: Packt Publishing.

Kent, K., Chevalier, S., Grance, T., y Dang, H. (2006). *Guide to integrating forensic techniques into incident response.* NIST Special Publication 800-86, National Institute of Standards and Technology. (https://doi.org/10.6028/NIST.SP.800-86)

Kharraz, A., Kirda, E., Robertson, W., Balzarotti, D., y Francillon, A. (2015). Cutting the gordian knot: A look under the hood of ransomware attacks. En *Detection of intrusions and malware, and vulnerability assessment (dimva 2015)* (Vol. 9148, pp. 3–24). Cham: Springer. (Estudio profundo de comportamiento y técnicas de ransomware incluyendo análisis de cifrado y comunicación C2) doi: 10.1007/978-3-319-20550-2_1

Korkosh, A., Samsonsen, E., y Moan, J. (2025). *Automating forensic disk image generation for digital forensics education* (Bachelor's thesis). Norwegian University of Science and Technology.

Ligh, M. H., Case, A., Levy, J., y Walters, A. (2014). *The art of memory forensics: Detecting malware and threats in windows, linux, and mac memory.* Wiley.

MacFarlane, J. (2025). *Pandoc: A universal document converter.* https://pandoc.org/. (Herramienta de conversión entre múltiples formatos de documento. Visitado 2026)

MITRE Corporation. (2025). *Caldera: Adversary emulation platform.* https://caldera.mitre.org/. (Framework de emulación adversaria basado en MITRE ATT&CK para automatizar simulaciones de ataque. Visitado 2026)

Mockapetris, P. (1987). *Domain Names – Implementation and Specifica-*

tion (RFC n.º RFC 1035). Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/rfc1035. (Visitado 2026)

Mohurle, S., y Patil, M. (2017). A brief study of wannacry threat: Ransomware attack 2017. *International Journal of Advanced Research in Computer Science*, *8*(5), 1938–1940. (Análisis del impacto global de WannaCry en mayo de 2017)

Pham, C., Tang, D., Chinen, K.-i., y Beuran, R. (2016). Cyris: a cyber range instantiation system for facilitating security training. En *Proceedings of the 7th symposium on information and communication technology* (p. 251–258). New York, NY, USA: Association for Computing Machinery. Descargado de https://doi.org/10.1145/3011077.3011087 doi: 10.1145/3011077.3011087

Postel, J. (1981). *Transmission Control Protocol* (RFC n.º RFC 793). Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/rfc793. (Visitado 2026)

Python Cryptographic Authority. (2025). *Cryptography: Python library for encryption and cryptographic recipes.* https://pypi.org/project/cryptography/. (Librería para operaciones criptográficas (AES, RSA, hashing, etc.). Visitado 2026)

Python Software Foundation. (2025a). *tarfile - read and write tar archive files.* https://docs.python.org/3/library/tarfile.html. (Módulo de la biblioteca estándar de Python para leer y escribir archivos TAR. Visitado 2026)

Python Software Foundation. (2025b). *zipfile - work with zip archives.* https://docs.python.org/3/library/zipfile.html. (Módulo de la biblioteca estándar de Python para leer y escribir archivos ZIP. Visitado 2026)

Red Hat, Inc. (2025). *Ansible documentation.* https://docs.ansible.com/. (Herramienta de automatización de TI para configuración, despliegue y orquestación. Visitado 2026)

ReversingLabs. (2017). *Ransomware yara rules - wannacry detection signatures.* Repositorio en GitHub: https://github.com/reversinglabs/reversinglabs-yara-rules/tree/develop/yara/ransomware. (Colección de reglas YARA para detección de patrones binarios de WannaCry ransomware. Incluye firmas main_1, main_2, main_3, start_service_3 para identificación forense. Visitado 2026)

Ronacher, A. (2025). *Jinja2 documentation.* https://jinja.palletsprojects.com/. (Motor de plantillas para Python utilizado por Ansible. Visitado 2026)

Scanlon, M., Du, X., y Lillis, D. (2017). Eviplant: An efficient digital forensic challenge creation, manipulation and distribution solution. *Digital Investigation*, *20*, S29–S36. doi: 10.1016/j.diin.2017.01.010

Schmidt, A., Škrabal, M., y Franke, K. (2023). Improving trace synthesis by utilizing computer vision for user action emulation. En *Dfrws 2023.* Descargado de https://dfrws.org/wp-content/uploads/2023/07/schmidt-improvingtracesynthesis.pdf

Spiekermann, D., y Keller, J. (2022). Requirements for crafting virtual network

packet captures. *Journal of Cybersecurity and Privacy*, *2*(3), 516–526. Descargado de https://www.mdpi.com/2624-800X/2/3/26 doi: 10.3390/jcp2030026

The Samba Team. (2017, mayo). *Samba security releases - eternalblue/ms17-010*. https://www.samba.org/samba/history/security.html. (Análisis de seguridad confirmando que Samba no es vulnerable a EternalBlue/MS17-010 debido a diferencias en la implementación del protocolo SMB respecto a Microsoft Windows. Visitado 2026)

The Samba Team. (2024). *Samba: Opening windows to a wider world.* https://www.samba.org/. (Implementación de código abierto de protocolo SMB/CIFS para interoperabilidad Windows-Linux. Visitado 2026)

The Wireshark Team. (2024). Wireshark user's guide [Manual de software informático]. Descargado de https://www.wireshark.org/docs/wsug_html_chunked/

Ukwandu, E., Farah, M. A. B., Hindy, H., Brosset, D., Kavallieros, D., Atkinson, R., ... Bellekens, X. (2020). A review of cyber-ranges and test-beds: Current and future trends. *Sensors*, *20*(24), 7148. (Revisión de cyber ranges y testbeds con taxonomía multidimensional. Visitado 2026) doi: 10.3390/s20247148

World Economic Forum. (2024). *Strategic cybersecurity talent framework.* https://www3.weforum.org/docs/WEF_Strategic_Cybersecurity_Talent_Framework_2024.pdf. (Marco estratégico para abordar la escasez global de talento en ciberseguridad. Visitado 2026)

Yamin, M. M., Katt, B., y Gkioulos, V. (2019). Cyber ranges and security testbeds: Scenarios, functions, tools and architecture. *Computers & Security*. doi: 10.1016/j.cose.2019.101636