Trabajo Fin de Máster

Máster Universitario en Ingeniería de Telecomunicación

Mejora y Validación de una Aplicación para la detección de ataques HTTP basados en URI mediante IDS

Autor: Pedro García Frutos

Tutor: Francisco Javier Muñoz Calle

Dpto. Ingeniería Telemática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

Mejora y Validación de una Aplicación para la detección de ataques HTTP basados en URI mediante IDS

Autor:
Pedro García Frutos

Tutor:
Francisco Javier Muñoz Calle
Profesor Colaborador

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Máster: Mejora y Validación de una Aplicación para la detección de ataques HTTP basados en URI mediante IDS

Autor: Pedro García Frutos

Tutor: Francisco Javier Muñoz Calle

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A Isasi

A mi familia y amigos

A mis maestros

Resumen

Este trabajo consiste en el estudio de las detecciones generadas por ciertos IDS (Intrusion Detection System) de distinto tipo, y bajo ciertas condiciones, para poder establecer unas conclusiones que nos permitan hacer un uso más eficiente y adecuado de los mismos.

Con este propósito en mente se han creado dos aplicaciones: la primera de ellas, denominada Aplicación Centralizadora de Detectores (ACD) [1], nos permite la inserción de múltiples IDS de distinto tipo de una forma centralizada, rápida, sencilla y flexible.

Por otro lado, la segunda de las aplicaciones desarrolladas recibe el nombre de: Aplicación Distribuidora de Tareas (ADT) [2], y tiene como propósito la distribución centralizada y controlada de tareas en un conjunto de equipos para conseguir reducir los tiempos de ejecución de las mismas.

Los resultados obtenidos con el uso conjunto de ambas aplicaciones y los estudios realizados pueden consultarse en el repositorio de GitHub habilitado para tal fin [3].

Abstract

This work consists of the detections generated by certain IDS (Intrusion Detection System) of different types of study, and under certain conditions, in order to establish conclusions that allow us to make a more efficient and adequate use of them.

With this purpose in mind, two applications have been created: the first one, called Detector Centralization Application (ACD) [1], does allow the insertion of multiple IDS of different types in a centralized, fast, simple and flexible way.

On the other hand, the second of the developed applications is called: Task Distributor Application (ADT) [2], and its purpose is the centralized and controlled distribution of tasks in a set of computers to reduce the execution times of the tasks.

The results obtained with the joint use of both applications and the studies carried out can be consulted in the GitHub repository enabled for this purpose [3].

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

La desconfianza es la madre de la seguridad.

- Aristófanes -

Las telecomunicaciones forman parte de nuestro presente, los últimos acontecimientos vividos no han hecho sino acrecentar aún más las dependencias de las mismas en un mundo cada vez más conectado.

Operaciones como realizar una compra 'a golpe de click', o realizar cualquier tipo de trámtite burocrático telemáticamente, es ya algo tan común que lo hemos asumido como parte de nuestra rutina.

Nuestra identidad digital a día de hoy es incluso tan importante como la real, y es por este tipo de cosas por los que los fraudes digitales y ciberdelincuentes están a la orden del día, y han desarrollado tantos mecanismos y técnicas para sacar provecho de la situación como alcanza la creatividad humana.

Con este trabajo, junto con tantos otros proyectos y herramientas, se pretende contribuir con nuestro granito de arena a proveer de ciertas herramientas, que en un futuro puedan servir para hacer de las telecomunicaciones un espacio un poco más seguro para todos.

1.1 Motivación

Un IDS (del inglés Intrusion Detection System) es un sistema cuyo propósito es el de alertar de un ataque cuando se produce (por ello se consideran sistemas reactivos) sin realizar ninguna acción sobre el mismo, es decir, su único propósito se basa en alertar sobre su existencia para que se tomen las medidas pertinentes. [4]

Este proyecto parte ya de una base establecida de detectores o IDS que tienen por objeto el análisis de 'Datasets' de uris para clasificarlas como uris susceptibles de ser 'attacks' o por el contrario uris 'clean'.

Estos sistemas ya desarrollados presentan ciertas deficiencias en su funcionamiento que en este proyecto tratarán de ser subsanadas.

Otras de las características que nos impulsan a la hora de realizar este trabajo es que estos sistemas son heterogéneos y están descentralizados. Esto hace que cualquier acción sobre los mismos deba realizarse de forma individualizada para cada uno de ellos, teniendo que emplear más tiempo y recursos en la subsanación de errores, configuración y lanzamiento de las detecciones.

1.2 Objetivos

Teniendo en cuenta lo expuesto en el punto anterior, los objetivos de este proyecto son:

- Analizar el comportamiento de los IDS implementados sobre un conjunto dado de 'Dataset', de forma que podamos establecer comparaciones entre ellos y extraer algunas conclusiones que nos permitan en un futuro hacer un uso más adecuado y eficiente de los mismos. Para poder establecer estos análisis deberemos:

- Desarrollar una herramienta que permita la integración rápida y flexible de IDS, y que provea de una configuración y ejecución centralizadas para todos ellos a la que denominaremos 'Aplicación Centralizadora de Detectores' (a partir de ahora lo nombraremos como 'ACD').
- Desarrollar mecanismos que posibiliten una reducción significativa de los tiempos de detección. Esto será posible gracias a una aplicación que nos permita la distribución centralizada de tareas entre varios equipos. A dicha aplicación la denominaremos como 'Aplicación Distribuidora de Tareas' (a partir de este momento nos referiremos a ella como 'ADT').

1.3 Plan de trabajo

Fases	Tiempo
Estudios sobre trabajo previo e investigación sobre los lanzamientos	80 horas
Realización de ACD	110 horas
Realización de ADT e integración con ACD	150 horas
Pruebas	50 horas
Realización sobre la memoria del proyecto	60 horas
Total	450 horas

Tabla 1 Plan de trabajo del proyecto

Estudios sobre trabajo previo e investigación sobre los lanzamientos

Fase en la que se estudiaron los diversos detectores existentes, sus escenarios y los errores que estos presentaban. También se realizó un estudio del modelo de lanzamiento más adecuado para las uris.

En esta fase se determinó la sustitución de ciertos objetivos iniciales por otros en base al estudio realizado.

Más concretamente se determinó lo siguiente:

- CheckScript presentaba problemas en el lanzamiento de las uris y Checkpoint alteraba las mismas en el curso de la detección. Por lo que se decidió abandonar su uso para este proyecto.
- MLA también presentaba problemas en el lanzamiento de las uris y además el algoritmo empleado provocaba errores en las detecciones, otorgándole un carácter impredecible.
- Debido a los múltiples problemas surgidos, que emplearon mucho más tiempo del esperado y que implicaron incluso la re-elaboración y adición de algunos detectores, se descartó el empleo de HTCondor en el proyecto.

Estos puntos se verán con detalle en el capítulo siguiente de 'Antecedentes'.

Realización de ACD

Desarrollo de una de las herramientas principales del proyecto. Es un software centralizado que integra a detectores de diferentes naturalezas y que permite realizar múltiples detecciones simultáneas de un mismo IDS en un mismo equipo.

En esta etapa tuvo lugar la introducción de los IDS. Para dicha etapa, hubo que investigar la correcta adecuación del software de detección 'IL' (Inspector Log) [5] a CentOS 7 y al software propio desarrollado.

También se modificó casi en su totalidad 'MLA' (ModSecurity Log Analyzer) [6], a la vez que se implementó una versión hecha desde cero de la nueva versión del detector 'ModSecurity'.

Todo esto se verá con más detalles en capítulos posteriores.

Realización de ADT e integración con ACD

Desarrollo de la segunda de las herramientas principales del proyecto. Permite el despliegue y ejecución de un proceso en múltiples equipos remotos; estos procesos reciben el nombre de 'tareas'.

ADT permite la creación de tareas que cumplan ciertas condiciones (establecidas en el capítulo 4), el despliegue de las mismas, su ejecución, y la recogida automática de los resultados generados por estas, amén de disponer de un control de estado que permite la monitorización en tiempo real de las tareas desplegadas.

Integra a ACD para dotar a las detecciones de mayor velocidad al diseminarla en un conjunto de equipos.

Pruebas

Fase en la que se utilizaron 'Datasets' reales para comprobar el correcto funcionamiento del software desarrollado en este proyecto. Para ello se hizo uso de los equipos del laboratorio de telemática.

Los resultados pueden obtenerse en GitHub [3].

Con los datos obtenidos en este apartado se reelaboró el capítulo de 'Validación de resultados'.

Realización sobre la memoria del proyecto

Tiempo de realización del presente documento.

1.4 Herramientas utilizadas

Hardware:

- **Equipos del laboratorio:** para la realización de este proyecto y ejecución de las pruebas se ha hecho uso de los equipos de la red del laboratorio de telemática. Se han llegado a usar hasta 28 equipos de manera simultánea. Las características de dichos equipos son:

<table>
<tr><th>Hardware</th><th colspan="2">Características</th></tr>
<tr><td>CPU</td><td colspan="2">Intel(R) Core(TM) i3-4130 CPU @ 3.40 GHz</td></tr>
<tr><td rowspan="2">Memoria RAM</td><td rowspan="2">12 GiB</td><td>8 GiB DIMM DDR3 Synchronous 1600 MHz (0.6 ns)</td></tr>
<tr><td>4GiB DIMM DDR3 Synchronous 1600 MHz (0.6 ns)</td></tr>
<tr><td>GPU</td><td colspan="2">GeForce GT 730</td></tr>
<tr><td>Disco duro</td><td>/dev/sda disk 1024 GB SPCC Solid State</td><td>/dev/sdb disk 500 GB TOSHIBA DT01ACA0</td></tr>
</table>

Tabla 2 Hardware de los equipos

- **Checkpoint 3200 Seccurity Gateway:** dispositivo WAF (del inglés Web Application Firewall) que mediante la monitorización del tráfico http permite proteger a un servidor de tráfico malicioso [7]. El checkpoint 3200 está pensado para un uso en pequeños entornos empresariales. Cuenta con filtrado de URL, IPS, antivirus, antibots y seguridad para email [8]. El principal uso que se ha hecho de él en este proyecto ha sido el de detecciones de URL's.

Software:

- **Sistemas operativos:**
  - **Windows 10:** requerido para el uso de checkpoint.
  - **CentOS 7:** sistema operativo en el que se ha desarrollado el software de este proyecto y han tenido lugar las pruebas.

- **Aplicaciones:**
  - **SmartConsole R80.10:** únicamente compatible con Windows. Es el software que nos permite realizar la configuración para el checkpoint 3200. Entre sus opciones se cuenta: establecer políticas, añadir reglas, establecer perfiles de funcionamiento… [9].
  - **GitHub:** Herramienta que ha permitido el alojamiento del software desarrollado para este proyecto y los resultados obtenidos en repositorios en la 'nube'.

- **Servidores:**
  - **Apache:** es un servidor web HTTP de código abierto. Está desarrollado y mantenido por una comunidad de usuarios en torno a la 'Apache Software Foundation'.

    Actualmente, y desde el 1996, es el servidor web más usado en todo el mundo debido a su seguridad y estabilidad. [10]

    Usado junto con ModSecurity.

  - **Nginx:** es un servidor web. Su primer lanzamiento fue en 2004 y se ha vuelto popular de forma muy rápida debido a las ventajas que ofrece a la hora de gestionar un tráfico elevado por encima de las 10000 conexiones al mismo tiempo. [10]

    Usado junto con Nemesida.

Lenguajes de programación empleados:

1. **C:** lenguaje de programación apreciado especialmente por su eficiencia. Es uno de los lenguajes más populares de la actualidad. [11]
2. **Python:** lenguaje de alto nivel utilizado en el desarrollo de aplicaciones de todo tipo. A diferencia de 'C' no es necesario su compilación antes de su ejecución, además es un lenguaje más sencillo e intuitivo que goza de cada vez mayor popularidad, por contra, es menos potente que 'C' [12]. Se ha utilizado la versión '3.6' para el desarrollo del software de este trabajo.
3. **Shell-Script (bash):** programa que permite a los usuarios la interacción con un sistema mediante un conjunto de comandos internos y una sintaxis determinada [13]. En nuestro caso hemos hecho uso del shell 'bash'.

1.5 Introducción a las herramientas de detección empleadas

En este punto trataremos de introducir las herramientas de detección utilizadas en los análisis de los 'Datasets' de este proyecto.

Un **IDS** o Sistema de detección de intrusos (del inglés Intrusion Detection System) es una herramienta utilizada en ciberseguridad para la monitorización y detección de actividades sospechosas, generando una alerta cuando estas son detectadas. Las alertas generadas serán estudiadas posteriormente y se tomarán las medidas adecuadas para remediar la amenaza [14]. Las respuestas a estas alertas pueden automatizarse para realizar acciones instantáneas.

De forma similar a los IDS los **IPS** (del inglés Intrusion Prevention System) tienen la capacidad de monitorizar el tráfico de red en busca de actividad maliciosa, pero además, cuenta como una de sus principales funcionalidades la de ejecutar acciones sobre dicha actividad maliciosa (como bloquearla).

Por otro lado, un **WAF** (del inglés Web Application Firewall) protege la capa de aplicación y está diseñado específicamente para analizar cada petición HTTP/S en dicha capa. Por tanto, un WAF podría definirse como una herramienta que además de la monitorización del tráfico de la capa de aplicación, es capaz de generar alertas y realizar un filtrado de dicho tráfico, es decir, tiene la capacidad de detectar **y actuar** directamente sobre las amenazas bloqueándolas de ser necesario.

A modo de resumen podría establecerse que tanto los IPS como los WAF comparten la capacidad de **monitorizar y filtrar el tráfico de red**, la principal diferencia se encontraría en que el IPS podría actuar sobre un tipo de tráfico más amplio, aunque en un entorno de trabajo real se recomienda la acción conjunta de ambas soluciones ya que WAF puede trabajar sobre algunos tráficos muy específicos que IPS no [15].

En este trabajo se han hecho uso de la capacidad de detección de estas herramientas con el propósito de registrar el número de ataques presentes en un conjunto de 'datasets' de entrada.

Las detecciones en este tipo de herramientas pueden estar basadas en:

- **Firmas:** se establecen un conjunto de firmas o reglas. A partir de estas se evalúa el tráfico. Para que las detecciones basadas en firmas sean lo más efectivas posibles, es necesario que las firmas o reglas cargadas estén actualizadas.

- **Políticas:** se establecen una serie de políticas de forma que, el tráfico que no satisfaga dichas políticas, es detectado como amenaza, y en el caso de WAF e IPS puede ser descartado automáticamente.

- **Anomalías:** se realiza un análisis estadístico del tráfico cursado y se establece un estándar de comparación. Cuando el tráfico se aleja de esta norma, se genera una alerta.

Para el desarrollo de este trabajo hemos empleado las detecciones **basadas en firmas**.

Las herramientas utilizadas en el análisis de las uris presentes en los 'datasets' de entrada han sido:

- **ModSecurity:** es un firewall de aplicaciones web (WAF) multiplataforma de código abierto desarrollado por SpiderLabs de Trustwave. Tiene un lenguaje de programación robusto basado en eventos que brinda protección contra una variedad de ataques contra aplicaciones web y permite el monitoreo, registro y análisis en tiempo real del tráfico HTTP. [16]

- **Snort:** es el sistema de prevención de intrusiones IPS de código abierto más importante del mundo. Snort IPS cuenta con una serie de reglas que ayudan a definir la actividad de red maliciosa. Cuando se detecta tráfico que vulnere dichas reglas se genera una alerta. [17].

- **Nemesida:** es un firewall de aplicaciones web (WAF) basado en un algoritmo clásico de aprendizaje automático que es capaz de detectar ataques con un tiempo de respuesta mínimo y alta precisión, casi sin detectar ataques falsos positivos. [18]

2 ANTECEDENTES

En este capítulo se abordarán las condiciones de partida del proyecto que servirán para dar contexto y justificación a las decisiones de diseño tomadas a lo largo del desarrollo.

2.1 Datasets analizados

Los datasets analizados en este proyecto han sido suministrados por: Francisco Javier Muñoz Calle ,tutor de este trabajo, y se recogen en la tabla siguiente:

Datasets		Número de uris	Observaciones
Reales	Biblio_Etiquetado	47402996	Tráfico registrado del servidor web Apache que atiende las solicitudes a la URL "bib.us.es" en el tiempo comprendido entre el 01/01/2017 y el 17/07/2017
	Biblio_Anonimizado	47402996	Surgido al 'anonimizar' las uris presentes en el Dataset: Biblio-Etiquetado
	INVES	14151496	Recolectado de las trazas del servidor web del servicio de investigación de la Universidad de Sevilla (http://fama.us.es) durante Mayo de 2018
De ataques	0days_A-420	420	Generados con CVE[1] de 2017 y 2018 y CVSS[2]>9
	0days_revisado	397	Revisión de 0days_A-420 (2014-2018)
	Fwaf-2200	2200	Subconjunto con CVE identificados (reglas de Snort) con fecha anterior a 2017
	Fwaf-badqueries	48065	Ataques WAF con fecha anterior a 2017
	Ataques-800	833	CVE incluidos en firmas Snort con fecha anterior a 2017

[1] Common Vulnerabililities and Exposures: lista sobre vulnerabilidades de seguridad conocidas.
[2] Common Vulnerability Scoring System: sistema que proporciona una puntuación para reflejar la gravedad de una detección.

	Ataques-1100	1177	CVE 2016 y 2017
	Osvdb	6896	Ataques generados a partir de OSVDB[3] con fecha anterior a 2009
	Rdb	933	Ataques generados a partir de RDB con fecha anterior a 2009

Tabla 3 Datasets analizados

2.2 Detectores de partida

2.2.1 CheckScript

Checkscript es un proyecto de trabajo de fin de máster [19] que se basa en un esquema como el de la figura:

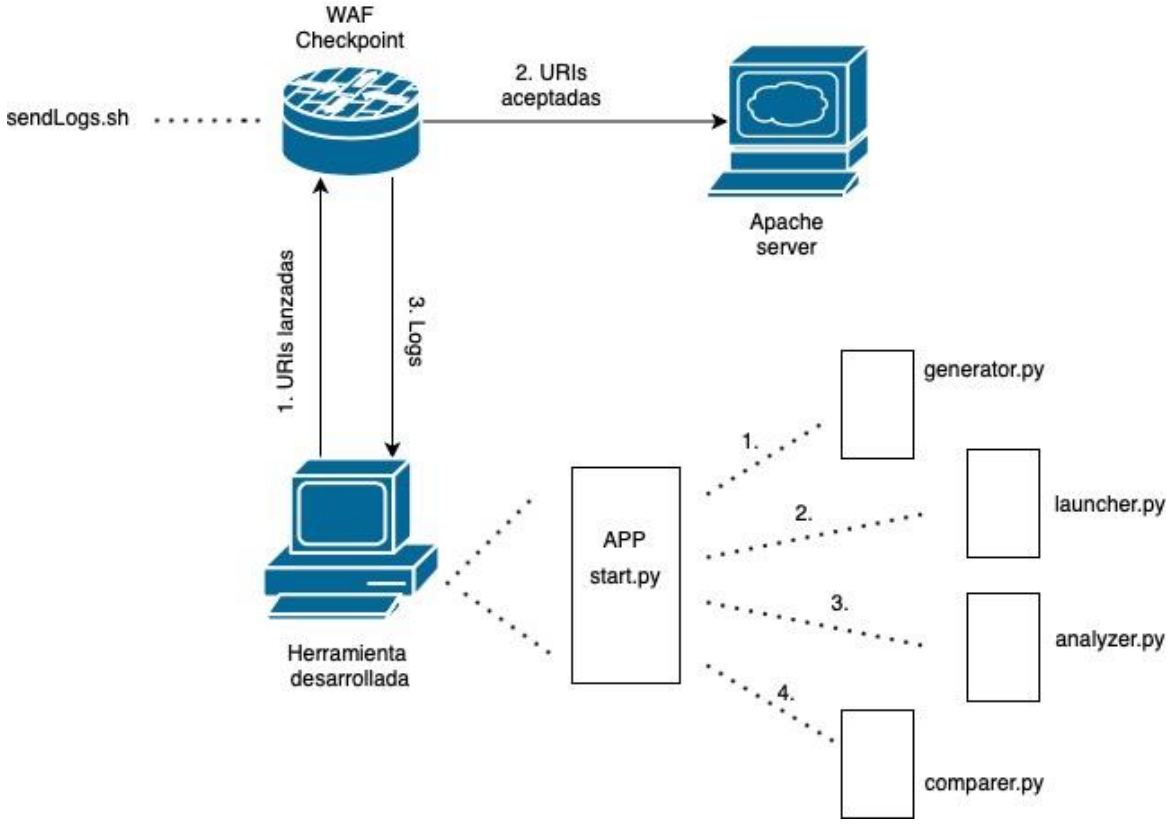


Ilustración 1 Esquema de funcionamiento de CheckScript [19]

En este esquema podemos observar cómo se realiza el lanzamiento del fichero de uris desde el equipo que contiene la herramienta desarrollada por el compañero, hacia un equipo servidor con apache.

En la misma figura puede apreciarse la existencia del WAF de checkpoint que realiza la función de

[3] Open Source Vulnerability Database: base de datos de vulnerabilidades de código abierto.

encaminador entre el equipo cliente y el equipo servidor, y en medio de este proceso genera un registro de detecciones 'log' del tráfico cursado, es decir, las uris emitidas por el equipo cliente.

Los logs generados por el WAF de Checkpoint son recogidos por el equipo cliente y analizados. En base a dicho análisis se establece la clasificación entre uris 'clean' y uris 'attacks'.

Para este proyecto se montó dicho escenario tal y como estaba recogido en el trabajo del compañero, con objeto de analizar los lanzamientos e incluir ciertas mejoras en el proceso de detección.

En la siguiente figura se muestra dicho escenario:

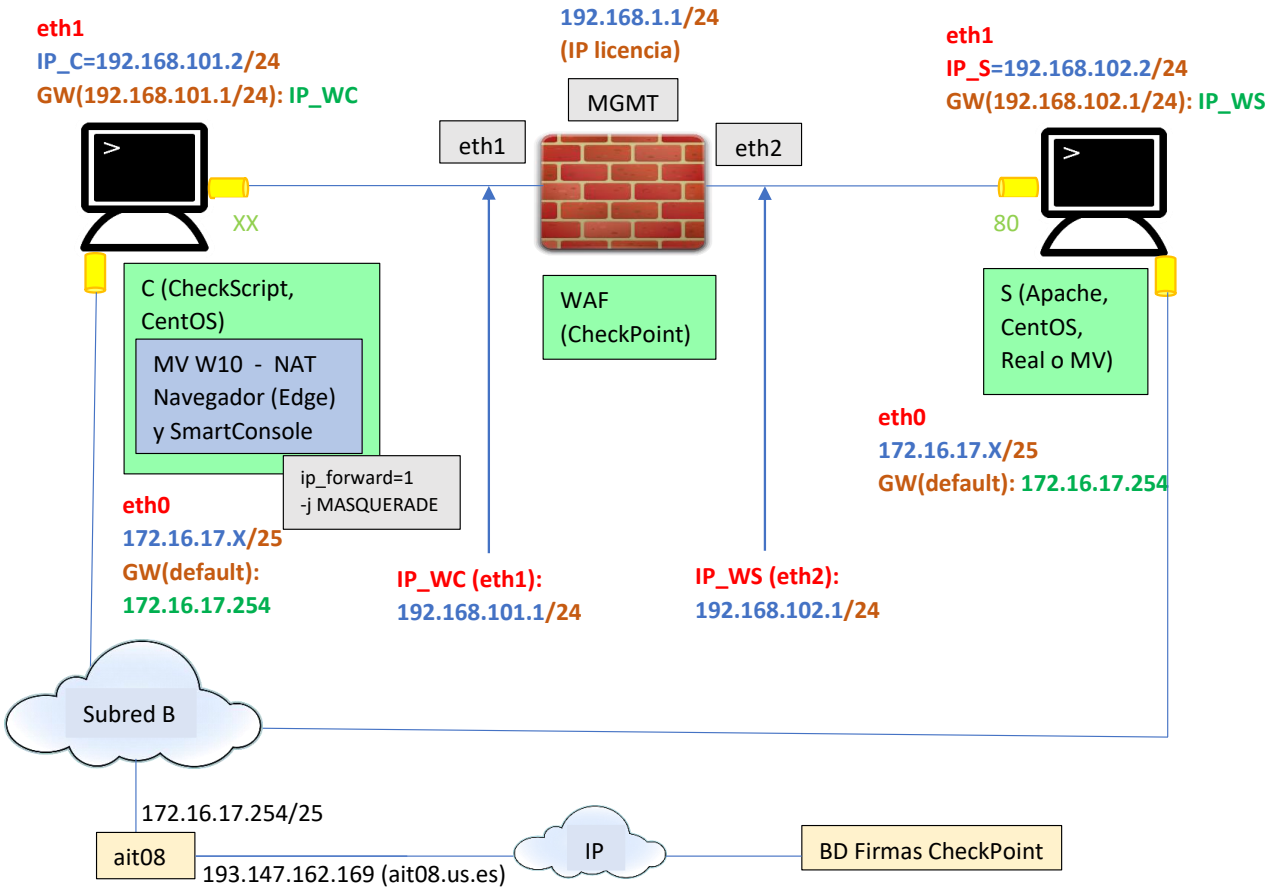


Ilustración 2 Escenario de CheckScript

Tras la realización de numerosas pruebas se llegaron a ciertas conclusiones no previstas que conllevaron a la no inclusión de Checkpoint como uno de los IDS de este proyecto. Estas conclusiones fueron:

1. Codificación excesiva de las uris: se comprobó que antes del lanzamiento se realizaba la codificación de un buen número de caracteres dentro de las uris, incluso más de los mínimos exigidos por el protocolo htttp para que estas pudiesen viajar. Esta codificación 'agresiva' alteraba considerablemente los resultados obtenidos en las detecciones.

   Se intentó eliminar el módulo de codificación previo al lanzamiento y se descubrió que el propio método 'request' de Python realizaba una codificación de las uris, por lo que en ningún caso estas llegaban al servidor tal y como figuraban en el fichero de entrada.

2. Más allá de la codificación previa al lanzamiento que alteraba las uris originales, se concluyó que el WAF de Checkpoint presentaba ciertas inconsistencias y un comportamiento errático que nos hacía imposible prever su comportamiento. Algunas de las anomalías observadas consistían en:

   2.1. Se comprobó que si se realizaba el lanzamiento de uris consecutivas repetidas como:

```
http://192.168.2.2/fosagent/repl/download-file?basedir=4&filepath=....Windowswin.ini

http://192.168.2.2/admin/admin_process.php?act=editpollform&id=1%20and%201=1::/www/admin/admin_process.php?act=cateditform&id=1%20and%201=1

http://192.168.2.2/fosagent/repl/download-file?basedir=4&filepath=....Windowswin.ini

http://192.168.2.2/admin/admin_process.php?act=editpollform&id=1%20and%201=1::/www/admin/admin_process.php?act=cateditform&id=1%20and%201=1
```

   En Checkpoint solo se registra una vez cada una de las uris cuando deberían registrarse dos veces.

   Básicamente se concluyó que Checkpoint solo detecta una uri como ataque una vez. En consecutivos lanzamientos de la misma uri no genera registro, como si contuviese una especie de memoria caché.

   2.2. También se determinó que realizaba cierta codificación sobre las uris recibidas. Como la sustitución del carácter '+' o el carácter '%20' por un espacio ' ' .

   Este hecho es significativamente grave, ya que codificaciones imprevistas por parte de Checkpoint de las uris de llegada, hacen imposible establecer una relación entre las uris originales del fichero de entrada, y las recibidas por Checkpoint.

   2.3. Otro aspecto que demuestra la inconsistencia en los datos aportados por este dispositivo, es que se comprobó que para el lanzamiento de la uri:

```
http://192.168.2.2/dir/%20/alias/
```

   El analizador de red detectó:

```
Request URI: /dir/%20/alias/
Request Version: HTTP/1.1
```

   Y Checkpoint registró en el log lo siguiente:

```
resource: http://192.168.2.2/dir/ /alias/(+)http://192.168.2.2/dir/ /alias/;
```

   En el que se observa que no solo decodifica el '%20' por su correspondiente ' ', sino que además

registra el lanzamiento como si se hubiese producido **dos veces** cuando solo se realizó una vez.

El no poder determinar la causa de este comportamiento por parte de Checkpoint, junto con la codificación previa al lanzamiento que se realizaba de las uris, nos hizo desechar su uso para este proyecto ya que los datos obtenidos no eran consistentes.

2.2.2 MLA

El siguiente de los detectores de partidas estudiados fue 'MLA'.

'MLA' o 'ModSecurity Log Analyzer' [6], está basada en la versión 2.7 de ModSecurity. Esta versión es un módulo de seguridad que se incorpora al servidor Apache para realizar las detecciones de las uris.

Uno de los problemas que afectaban a este detector -como a cada uno de los estudiados en este capítulo- se encontraba en el proceso de lanzamiento.

Para el lanzamiento de las uris se utilizaba el comando 'curl'. Este comando se mostraba más sólido y eficaz que la implementación de lanzamiento de Python estudiada en el apartado anterior, no obstante, presentaba ciertas deficiencias que no fueron detectadas hasta su estudio en este trabajo.

Las conclusiones extraídas de dicho estudio tras el lanzamiento de **millones** de uris es que los caracteres:

- [
-]
- {
- }
- #
- ' ' (carácter de espacio)

Provocaban que la uri no fuera lanzada, o lo que es aún más grave, alteraciones en las mismas. Por ejemplo: el carácter '#' hace que todo lo que venga posteriormente a dicho carácter se 'corte' y no sea enviado, ya que se detecta como si fuese un comentario.

La robustez demostrada por el comando 'curl' en los estudios, frente a los pocos caracteres que exigen ser codificados para que el lanzamiento llegue correctamente a su destino, hizo que tomásemos el lanzador de este proyecto como base del nuestro, añadiendo eso sí, cambios y funcionalidades adicionales.

Los cambios perpetrados para que el lanzamiento de las uris pudiese realizarse de forma correcta, consistieron en realizar la codificación de los caracteres 'conflictivos'. En la siguiente tabla se muestra la codificación realizada.

Carácter	Codificación
[	**\[**
]	**\]**
{	**\{**
}	**\}**
#	**%23**

Carácter de espaciado: ‘ ‘	**%20**

Tabla 4 Codificación de carácteres

Nota: como puede deducirse de la tabla, ni siquiera los caracteres de apertura y cierre de corchetes/llaves requieren de una codificación como tal, simplemente demandan del carácter ‘\’ para evitar interpretaciones de los mismos durante el lanzamiento.

Otros de los problemas estudiados en este proyecto radican en su algoritmo:

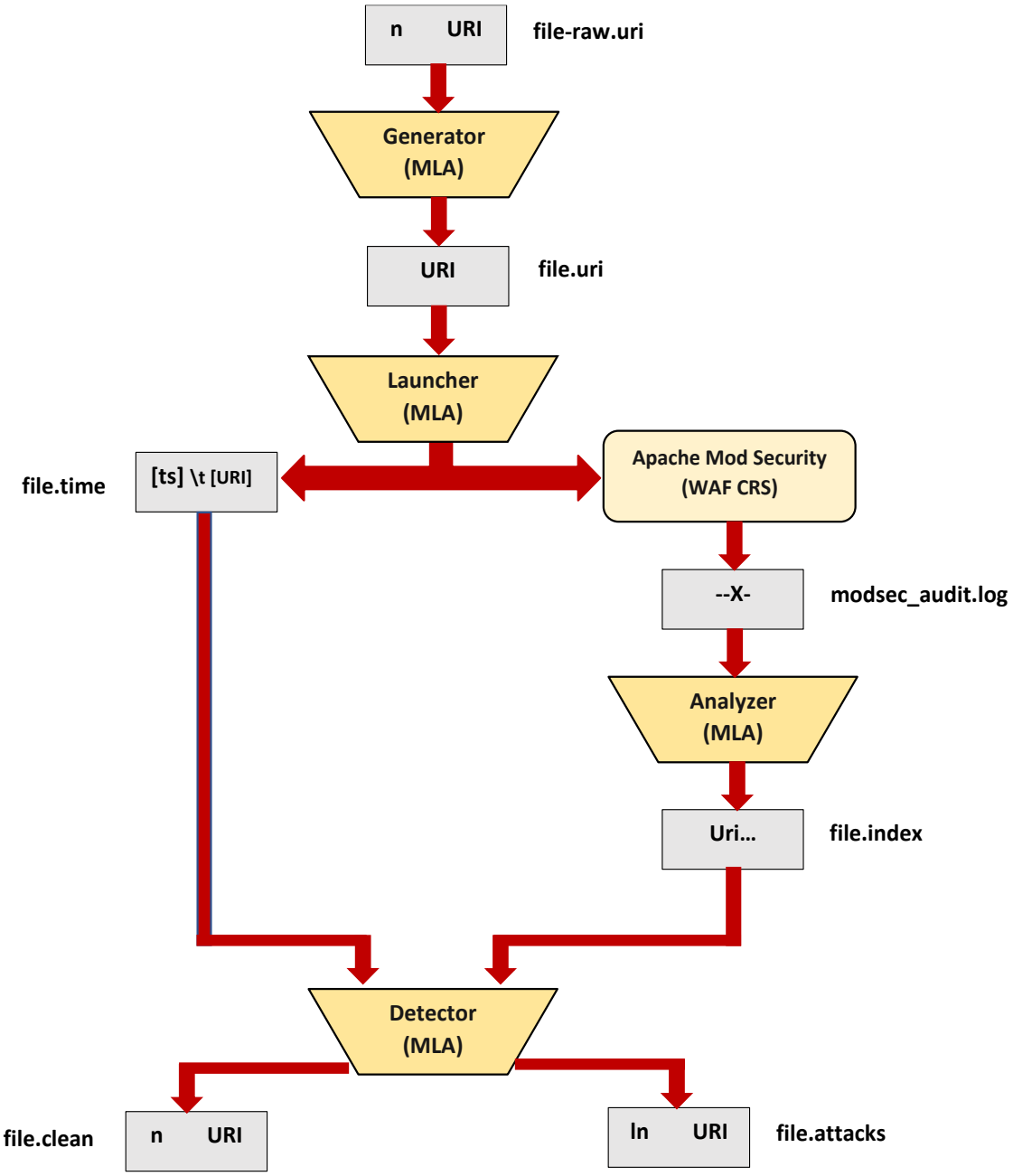


Ilustración 3 Algoritmo original de MLA

Básicamente el algoritmo se desglosa en lo siguiente:

1. Generación de fichero '.uri' a partir de fichero de entrada '-raw.uri'.
2. Cuando se realiza el lanzamiento, se genera un fichero de extensión '.time' que relaciona cada uri con el instante de tiempo en el que se ha realizado.
3. Generación de archivo '.log' generado por ModSecurity. Contiene las detecciones realizadas para las uris lanzadas.
4. Generación de archivo '.index' que consiste en un resumen del '.log' para facilitar la legibilidad de las detecciones.
5. Clasifición de las uris en 'attacks' y 'clean'. Para ello, se comparan las uris y el 'TimeStamp' (tiempo de llegada de la uri al servidor) de las uris obtenidas en el fichero de 'index', con las uris y el tiempo de lanzamiento registrados en el fichero '.time'.

   De esta forma se pretende relacionar las uris de entrada con las que llegan al servidor. Se deduce que si una uri lanzada es idéntica a otra registrada en el servidor, y que además, el tiempo de lanzamiento y de llegada al servidor coinciden, se trata de la misma uri.

Aunque el planteamiento sobre el papel parezca correcto, en la práctica se percibieron dos problemas que invalidan la estructura, y nos hicieron modificarla a otra que veremos con mayor detenimiento en el capítulo siguiente. Estos problemas fueron:

- Tras el lanzamiento de las uris se procede inmediatamente al procesado del 'log' sin establecer ningún tipo de sincronismo. Esto provoca que en ejecuciones sucesivas puedan darse diferentes resultados debido a que el 'log' no se terminó de escribir por completo cuando comenzó la generación del 'index'.

  Para solucionar esto, en el proyecto desarrollado se ha establecido que hasta que el fichero de 'log' no haya sido escrito y volcado por completo, no se proceda a la creación del 'index'.

- El otro problema reportado está relacionado con los tiempos:
  - Por una parte, es posible el envío consecutivo de uris idénticas que sean enviadas y recibidas en un mismo instante temporal (siendo más rigurosos con el lenguaje, lo más correcto sería decir que los 'TimeStamps' registrados tanto en el cliente -lanzador- como en el servidor -receptor- coinciden) este hecho no haría posible establecer una relación directa y precisa de las uris enviadas con las recibidas.
  - El otro problema detectado es de mayor gravedad. Puede ocurrir, que en función de la carga de trabajo del equipo, el instante de lanzamiento registrado para una uri no coincida con el instante de llegada. Este hecho provocaría que la uri lanzada y la uri recibida no sean percibidas como la misma, cuando sí lo son.

  Estos problemas relacionados con los tiempos han propiciado que la estructura de lanzamiento y análisis desarrollada para este trabajo no se apoye en instantes temporales para establecer la relación entre la uri enviada por el cliente y la recibida por el servidor.

Otro cambio realizado de esta versión de MLA a nuestro proyecto es haber recodificado algunas partes en Python (Analyzer y Detector en 'Ilustración 3') buscando mejorar la eficiencia de las detecciones. Este hecho se verá con más detalle en capítulos posteriores.

Por último, durante el proceso de investigación para dar con un procedimiento de lanzamiento para las

uris adecuado, se descubrió que la nueva versión de ModSecurity (V3) no precisaba de servidor para funcionar, simplemente analizaba los ficheros uri de entrada mediante la carga de reglas de manera totalmente offline. Este hecho hizo que también se diseñase e integrase en este proyecto esta nueva versión, **ya que posibilita el lanzamiento de uris sin realizar ningún tipo de modificación sobre las mismas**.

2.2.3 IL

'IL' o 'Inspector Log', es un software [5] que integra las versiones offline (es decir, aquellas que no precisan de un servidor para funcionar, y por ende de ninguna codificación de las uris en el proceso de lanzamiento) de los detectores: ModSecurity, Snort y Nemesida.

Este software ha sido adaptado e incorporado a nuestro proyecto. A continuación se presentarán las diferentes configuraciones y posibilidades brindadas por la herramienta:

Argumentos requeridos	Descripción
-l logFile	Archivo de traza a procesar.

Tabla 5 Argumentos requeridos IL

Argumentos opcionales	Descripción	Valor por defecto
-t <list|apache|wellness|uri|elist>	Formato del archivo de traza.	apache
-r ruleDir	Directorio con las reglas de Snort/ Fichero de configuración de ModSecurity.	./rules
-m rulefile	Fichero de reglas de Nemesida.	-
-o <salida log limpio>	Archivo con los elementos de la traza que no han generado alertas.	-
-n	Aplicar las reglas ignorando mayúsculas y minúsculas	regla de acuerdo a la etiqueta "nocase" existente o al switch de la expresión regular correspondiente.
-e	Activar información extendida en las alertas (mansaje de alerta + sid)	Solo se muestra sid
-w	Generar avisos cuando no se puede decodificar caracteres %encoded	-

Tabla 6 Argumentos opcionales IL

Formato de trazas	Descripción
list	Lista de URIs sin método ni campos adicionales.
apache	Formato estándar de apache.
wellness	Formato proporcionado para las trazas de wellness.
uri	Primera línea contiene el número de uris. Cada línea incluye la longitud de la uri.
elist	Lista de petición uri código_respuesta tamaño_respuesta

Tabla 7 Formato de trazas en IL

En este trabajo hemos utilizado las siguientes opciones:

Detector	Invocación
ModSecurity	./ms-inspectorlog -l file.uri -t list -r modsecurity.conf
Nemesida	./inspectorlog -l file.uri -t list -m file_rules.txt
Snort	./inspectorlog -l file.uri -t list -r dir_rules_snort

Tabla 8 Invocaciones de IL usadas en este proyecto

Por último, mostraremos un esquema conceptual de IL:

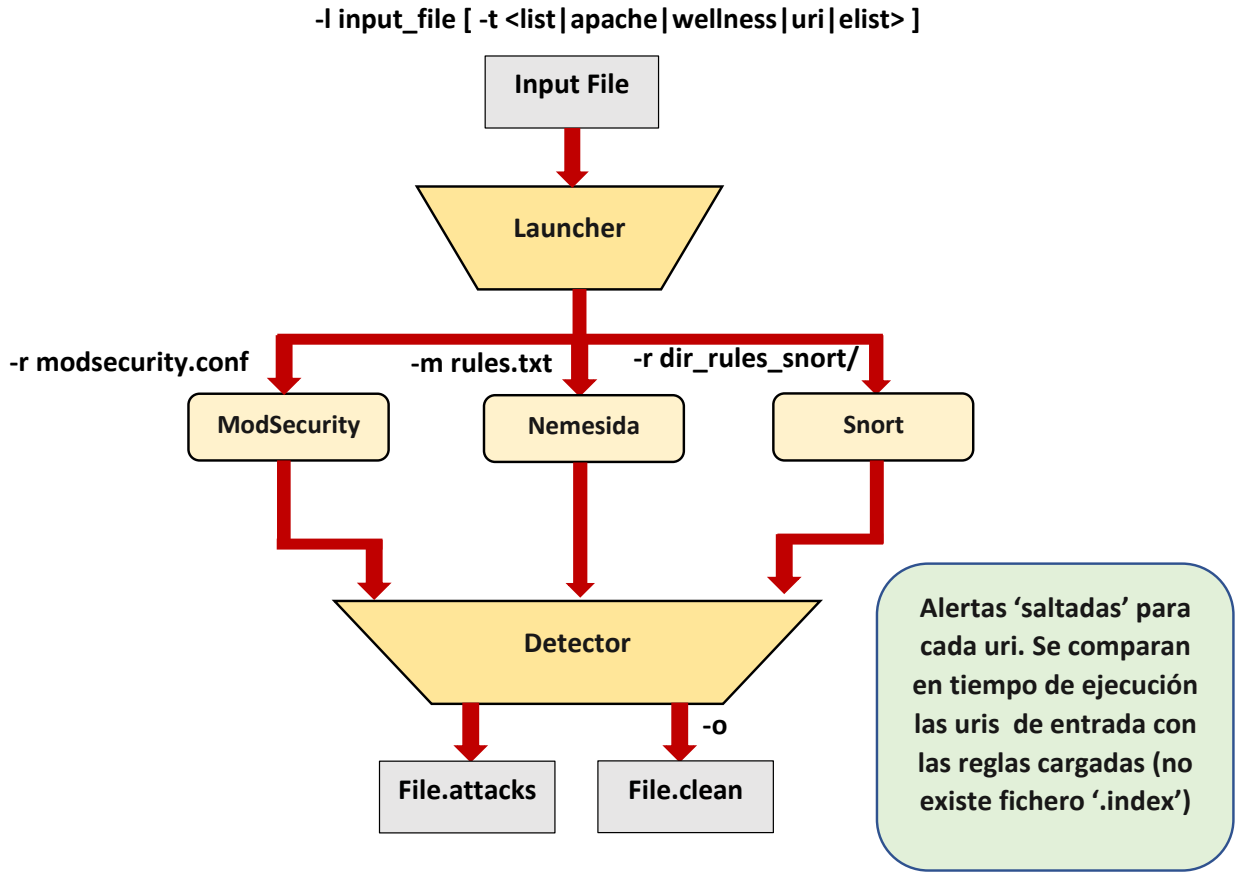


Ilustración 4 Diagrama conceptual IL

2.2.4 NemesidaScript

NemesidaScript [20] presenta una estructura similar a la estructura vista en MLA pero realizándose la detección sobre Nemesida, haciendo uso del servidor Nginx.

Al ser el último de los detectores estudiados, ya se tenía conocimientos de los errores que se producía en el resto de detectores a la hora de realizar los lanzamientos de las uris. Por lo que realmente para su implementación lo único que se hizo fue seguir los pasos de instalación de Nemesida + Nginx, y adaptar el 'analizador del log' a nuestro proyecto.

Las diferentes configuraciones aceptadas por esta herramienta se detallan en las tablas siguientes:

Argumentos opcionales	Descripción	Valor por defecto
-h, --help	Muestra mensaje de ayuda.	-
-o output	Fichero de salida con URIs específicas cada salto de línea.	input_file_name.uri
-u url	URL protegida por Nemesida WAF especificada en el fichero de configuración de Nginx.	http://localhost
-p port	Puerto específico para lanzar las uris.	80
-e error_log	Log de error de Nginx que contiene la información acerca de las URLs bloqueadas por Nemesida WAF.	/var/log/nginx/error.log
-a acces_log	Log que contiene la información sobre los accesos al servidor.	/var/log/nginx/acces.log

Tabla 9 Argumentos opcionales NemesidaScript

Argumentos requeridos	Descripción
-i input	Fichero de entrada en formato RAW (filename-raw.uri).
-f file_location	Fichero '.uri' formado a partir del fichero de entrada.
-id id	Valor numérico añadido para identificar los ficheros generados.

Tabla 10 Argumentos requeridos NemesidaScript

La representación de su diagrama funcional puede verse a continuación:

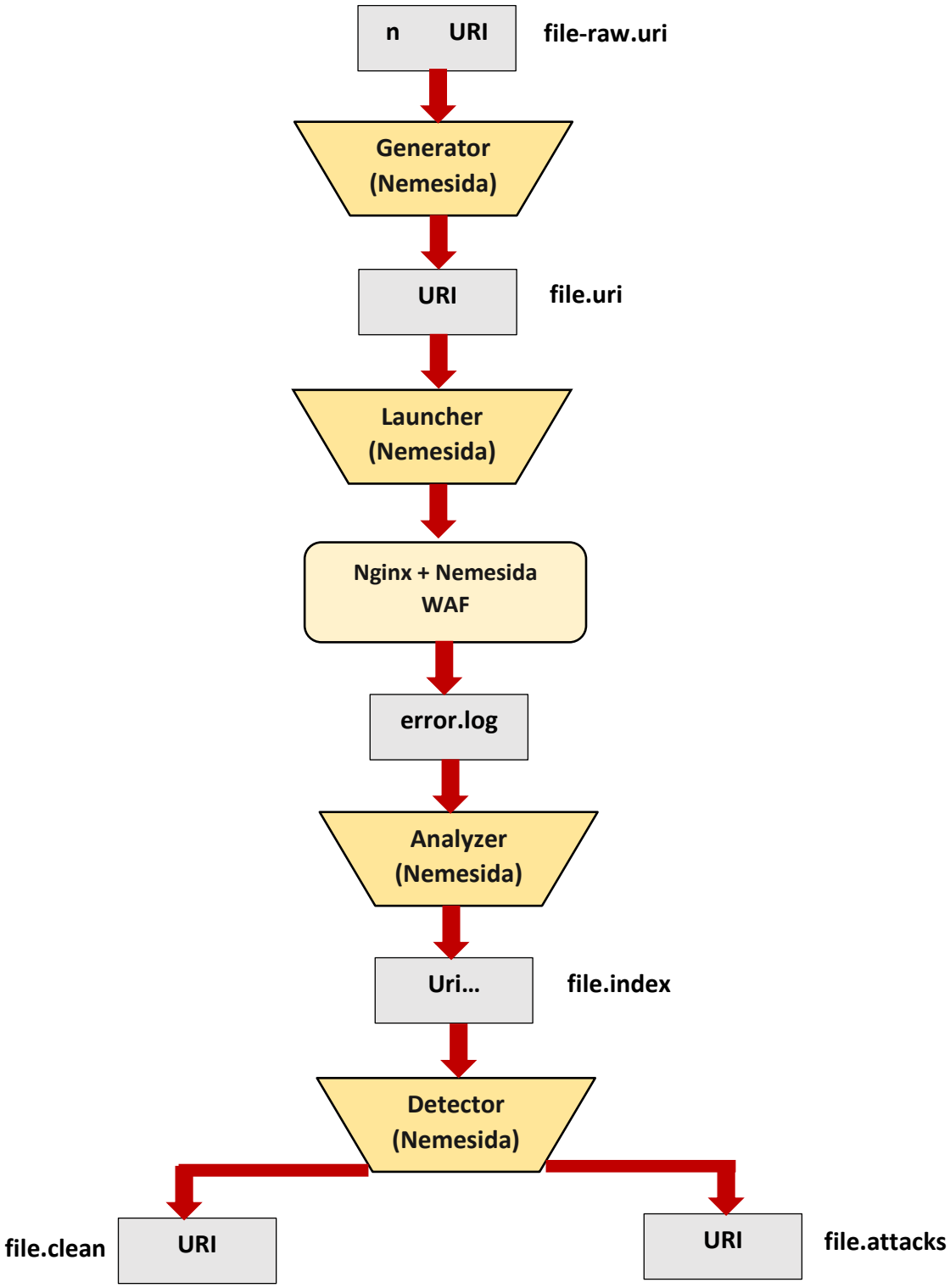


Ilustración 5 Diagrama funcional de NemesidaScript

3 APLICACIÓN CENTRALIZADORA DE DETECTORES (ACD)

En este apartado explicaremos la estructura que presenta la Aplicación Centralizadora de Detectores (ACD) así como sus características y funcionamiento.

Recordemos que ACD es una aplicación, que mediante una estructura común y una configuración centralizada, permite agrupar de forma rápida y efectiva a IDS de distintos tipos, teniendo que añadir solo una función adicional para lograr su inserción.

3.1 Estructura

Antes de entrar en los detalles del software desarrollado, es importante conocer con un grado mayor de profundidad la estructura común de los IDS presentados en capítulos anteriores.

La estructura se basa en un proceso de tres fases:

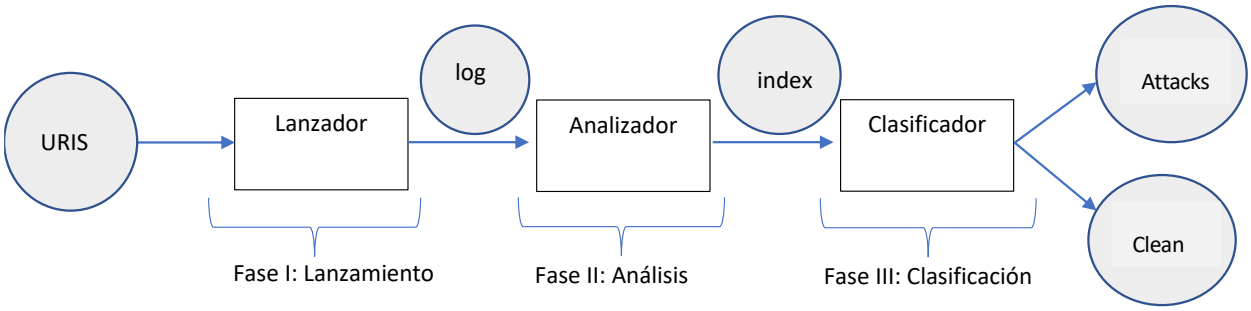


Ilustración 6 Esquema de funcionamiento común a todos los IDS

1. **Lanzamiento:** fase en la que se procede al envío de las uris contra el detector.

2. **Análisis:** fase en la que se procesa el registro generado por el detector (log) con el fin de generar un fichero resumen (index) de los ataques que favorezca su legibilidad por parte de un ser humano.

3. **Clasificación:** fase en la que se comparan las uris de entrada con el fichero resumen de los ataques (index) para poder establecer una clasificación entre 'uris attacks' y 'uris clean'.

Ejemplificaremos cada una de las fases descritas haciendo uso de un fichero '.uri' de entrada compuesto por dos 'uris'. La primera de ellas será detectada como 'attack' y la segunda como 'clean'. En este ejemplo haremos uso del IDS 'ModSecurityV3', uno de los detectores implementados en este proyecto y que describimos con más detalle más adelante.

- Fichero de uris de entrada:

```
/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-O%20/tmp/dsifnfnfdwsFFS)
/html/en/confAccessProt.html
```

- Log generado tras el lanzamiento (Tras la ejecución de la **Fase 1**):

```
---2nQdZawY---A--
[06/Jun/2022:11:38:38 +0200] 1654508318 127.0.0.1 12345 127.0.0.1 80
---2nQdZawY---B--
GET /appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-O%20/tmp/dsifnfnfdwsFFS) HTTP/1.1
Host: localhost
User-Agent: Apache/2.2.15 (Red Hat) (internal dummy connection)
Accept: Yes


---2nQdZawY---H--
ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?:;|\{|\|\|\||&|&&|\n|\r|\$\(|\$\(\(|`|\${|<\(|>\(|\(\s*\))\s*(?:{|\s*\(\s*|\w+=(?:[^\s]*|\$.*|\$.*|<.*|>.*|\'.*\'|\".*\")\s+|!\s*|\$)*\s*(?:'|\")*(?:[\?\*\[\]\(\)\-\|+\w'\"\./\\\\]+/)?[\\\\'\"]*(?: (5275 characters omitted)' against variable `ARGS:hostName' (Value: `$(/usr/bin/wget http://1.1.1.1:12345/file -O /tmp/dsifnfnfdwsFFS)' ) [file "detectores/mod_security_offline/rules/REQUEST-932-APPLICATION-ATTACK-RCE.conf"] [line "140"] [id "932105"] [rev ""] [msg "Remote Command Execution: Unix Command Injection"] [data "Matched Data: $(/usr/bin/wget found within ARGS:hostName: $(/usr/bin/wget http://1.1.1.1:12345/file -O /tmp/dsifnfnfdwsFFS)"] [severity "2"] [ver "OWASP_CRS/3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-shell"] [tag "platform-unix"] [tag "attack-rce"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "OWASP_CRS/WEB_ATTACK/COMMAND_INJECTION"] [tag "WASCTC/WASC-31"] [tag "OWASP_TOP_10/A1"] [tag "PCI/6.5.2"] [hostname "127.0.0.1"] [uri "/appliancews/getLicense"] [unique_id "1654508318"] [ref "o0,15v37,65"]
.
.
.
---2nQdZawY---Z—
```

- Salida ‘index’ generada por el analizador tras el procesado del log (**Fase 2**):

```
TimeStamp [06/Jun/2022:11:38:38 +0200]  Uri [/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-O%20/tmp/dsifnfnfdwsFFS)]  PLmin [1]  Score [20]  Nattacks [4]  [932105]  [932130]  [932150]  [932160]
```

- Salidas generadas tras el procesado del 'index' por parte del clasificador (**Fase 3**):

    - Fichero que contiene las uris detectadas como attacks:

```
Packet [1]        Uri
[/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-
O%20/tmp/dsifnfnfdwsFFS)]
```

    - Fichero que contiene las uris detectadas como clean:

```
Packet [2]        Uri [/html/en/confAccessProt.html]
```

    - Fichero que contiene información extendida sobre las uris detectadas como attacks:

```
---------------------- Statistics of URIs analyzed-----------------------
[2] input, [1] clean, [1] attacks
--------------------------- Analysis results ----------------------------
Packet [1]        Uri
[/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-
O%20/tmp/dsifnfnfdwsFFS)]     PLmin [1]        Score [20]        Nattacks [4]     [932105]
        [932130]        [932150]        [932160]
```

En este ejemplo pueden distinguirse con claridad cada una de las etapas comprendidas desde el lanzamiento de las uris hasta su detección. Más adelante, en el *Anexo A*, se especificará con más detalle el significado de los ficheros generados por cada uno de los detectores integrados.

En nuestro proyecto hemos aprovechado esta estructura común de la que gozan muchos de los detectores para realizar una herramienta que permita agilizar la inserción de nuevos IDS.

No solo es importante que las fases mostradas estén presentes en los detectores implementados. También es importante que se respete cierto formato de salida común a cada una de las fases; esto nos permitirá la introducción de nuevos softwares de detección minimizando los cambios necesarios.

Para poder llevar a cabo lo anterior, es **imprescindible** que el formato de la entrada y los ficheros generados cumplan al menos los siguientes requisitos:

1. **Fichero *uri* de entrada:** se aceptan dos tipos de formatos para el fichero de entrada:

    a. ***Basic:*** cada línea del fichero estará conformada única y exclusivamente por la uri a lanzar (formato del ejemplo).

b. ***Extended:*** las uris irán precedidas por un *"id"*. El formato que presente el identificador nos es indiferente, pero es imprescindible que sea único para cada uri y que vaya separado de esta por un tabulador. Ejemplo:

```
01-12-A000001        /agricola/salud/arquitectura/node
```

2. **Fichero de *log*:** este fichero es generado por el IDS implementado, por lo que su formato variará considerablemente en función del mismo. Debido a esta variación, el procesador del *'log'* (conocido en nuestro proyecto como *'analizador/analyzer'*) **será la parte que deba implementar el usuario que quiera hacer uso de nuestra herramienta**.

3. **Fichero *index*:** resumen de fichero de *log*. Los únicos requisitos imprescindibles en la generación de este fichero son:

   a. Campos separados por una tabulación.
   b. Presencia de campo ***Uri [x]***.

4. **Ficheros de resultados:** existen tres tipos de ficheros de resultados para cada detección y todos presentan el mismo formato independientemente del IDS utilizado:

   a. **Fichero *attacks:*** fichero con las uris registradas como attacks. Siguiendo con el ejemplo visto anteriormente, este fichero presentaría para cada uri detectada como attacks la siguiente estructura:

```
Packet [1]      Uri
[/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-O%20/tmp/dsifnfnfdwsFFS)]
```

   Como puede apreciarse existen dos campos (Packet […] y Uri […]) **separados por una tabulación**.

   El primero de los campos variará en función del formato del fichero de entrada utilizado. Si el formato es '*basic'* se usará el campo *'Packet',* dicho campo nos informa sobre la posición ocupada por la uri detectada como attacks en el fichero de entrada.

   Si el formato del fichero de entrada es *'extended'* en lugar del campo '*Packet*' estará presente el identificador único de esa uri. Ejemplo:

```
ID [01-12-A000001]   Uri [/agricola/salud/arquitectura/node]
```

b. **Fichero *clean*:** fichero con las uris registradas como clean. Exactamente el mismo formato que el fichero *'attacks'* pero con las uris detectadas como *'clean'.*

c. **Fichero *-info.attacks*:** fichero resumen de los resultados obtenidos en el lanzamiento. Muestra el número total de uris analizadas (campo '*input*'), junto con el número de attacks (campo *'attacks'*) y clean registradas (campo *'clean'*). También se muestran las uris detectadas como attacks con información adicional. Siguiendo con el ejemplo utilizado para mostrar las diferentes fases de ACD:

```
---------- Statistics of URIs analyzed----------
[2] input, [1] clean, [1] attacks
---------- Analysis results ----------
Packet [1]        Uri
[/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%2
0-O%20/tmp/dsifnfnfdwsFFS)]  PLmin [1]        Score [20]        Nattacks [4]
          [932105]        [932130]        [932150]        [932160]
```

Para finalizar este apartado, y una vez vista la estructura y formato que deben presentar los detectores que quieran implementarse en la herramienta desarrollada, pasamos a concretar con mayor grado de detalle la estructura de *ACD:*

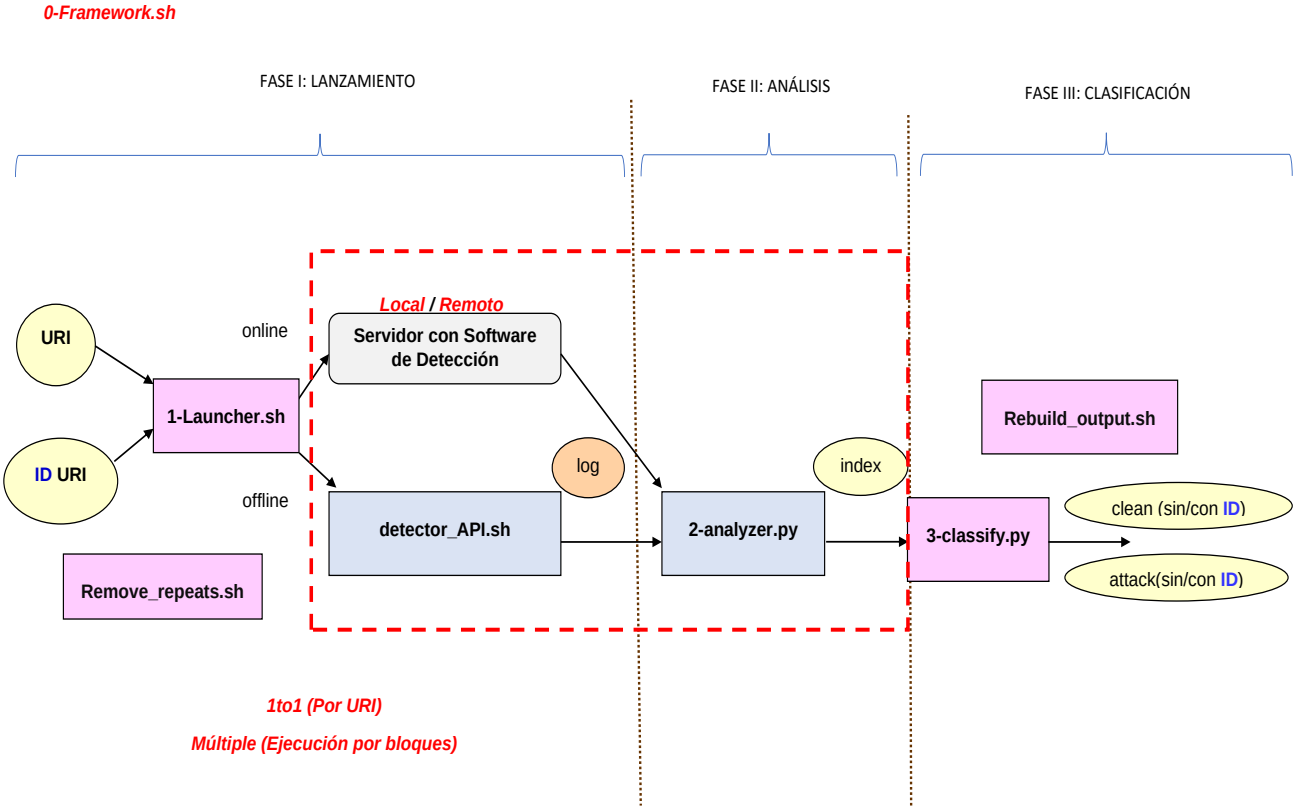


Ilustración 7 Esquema de funcionamiento de ACD

Como puede comprobarse, las fases descritas con anterioridad pueden verse perfectamente definidas en

el dibujo. Otro hecho destacable es que, la zona enmarcada dentro de la línea discontinua es la que el usuario debe modificar si pretende realizar la inserción de uno o varios detectores adicionales. El procedimiento a seguir se explicará con detalle en el *Anexo C.*

3.2 Características

En este apartado veremos con más profundidad las características y posibilidades que nos brinda la herramienta ACD.

A partir del esquema de ACD comentaremos algunas de sus características:

1. **Remove_repeats.sh**: antes de realizar el lanzamiento de las uris, se ha incorporado un script activable desde la configuración del software, que nos permite eliminar las uris repetidas del fichero de entrada. Esto nos posibilita suprimir información redundante, y en ciertos casos en los que los ficheros cuentan con un gran número de uris repetidas, acelerar la detección.

   Al final del proceso de análisis se cuenta con otro script que realiza la reconstrucción de los ficheros de resultados, de manera que de cara al usuario se ha producido el lanzamiento del fichero completo. De esta forma, podría decirse que este proceso es **trasparente** al usuario.

2. **Launcher.sh:** como ya hemos visto anteriormente, este cuenta con dos formatos de entrada, el formato 'basic' y 'extended'. Antes de procederse al lanzamiento de las uris se comprueba que el formato de entrada es apto (autodetección) y que las uris comienzan por el carácter '/'.

3. **Modos de funcionamiento y tipos de lanzamiento**: podríamos decir que el comportamiento de ACD viene determinado por los modos de funcionamiento y los tipos de lanzamiento. Estos son combinables entre sí dando lugar a un abanico de posibilidades (6 en total). Dichas posibilidades se recogen en el esquema siguiente:

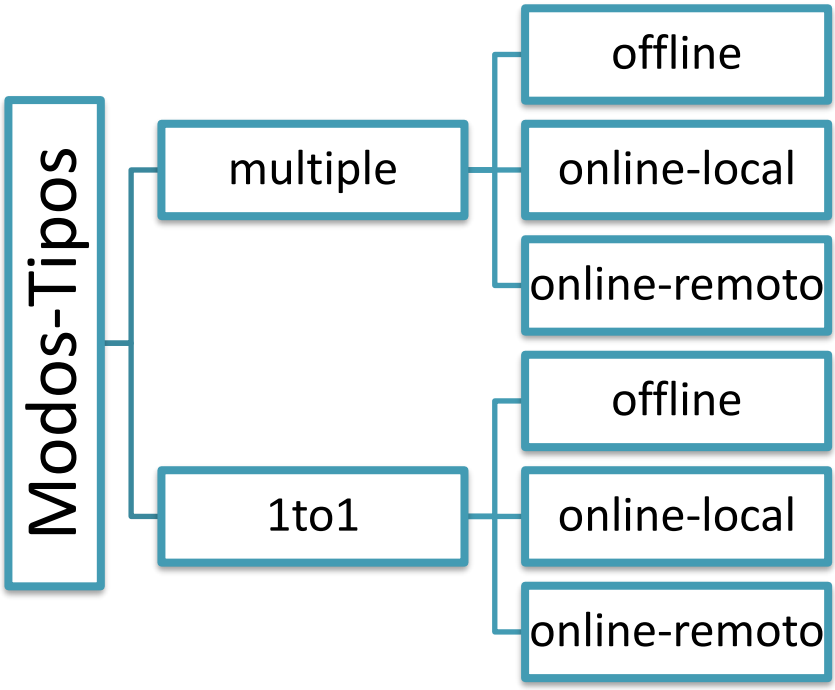


Ilustración 8 Resumen modos de funcionamiento y tipos de lanzamiento

3.1. Modos de funcionamiento:

3.1.1. Múltiple: las fases de lanzamiento, análisis y clasificación se hacen **'en bloque'**, es decir, primero se procede al lanzamiento de todas las uris presentes en el fichero de entrada, a continuación se analiza el log generado por el lanzamiento de **todas** las uris del fichero, y por último, se realiza la clasificación de la totalidad de las uris de entrada en 'attacks' y 'clean'.

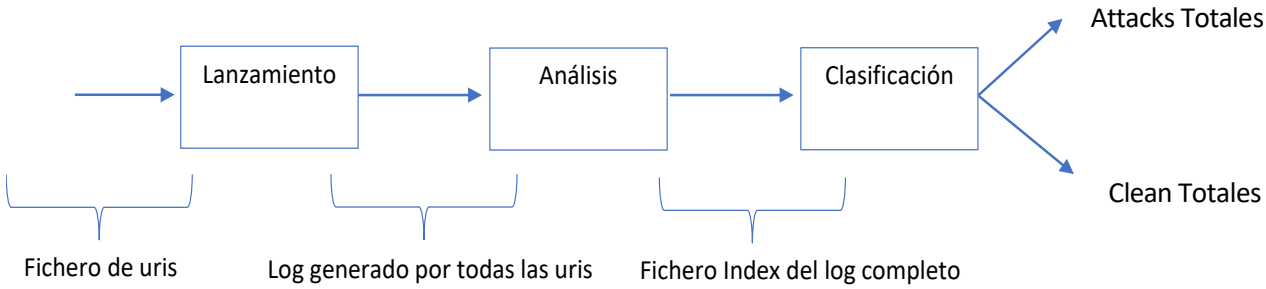


Ilustración 9 Esquema de funcionamiento modo múltiple

3.1.2. 1to1: las fases de *lanzamiento, análisis y clasificación* se hacen **individualmente** para cada uri, es decir, se procede al lanzamiento de una sola uri del fichero de entrada, posteriormente se pasaría a la fase de análisis, y por último al proceso de clasificación **para solamente esa uri**. Una vez analizada la uri, se procede a la siguiente del fichero, y así sucesivamente hasta finalizar todas las entradas.

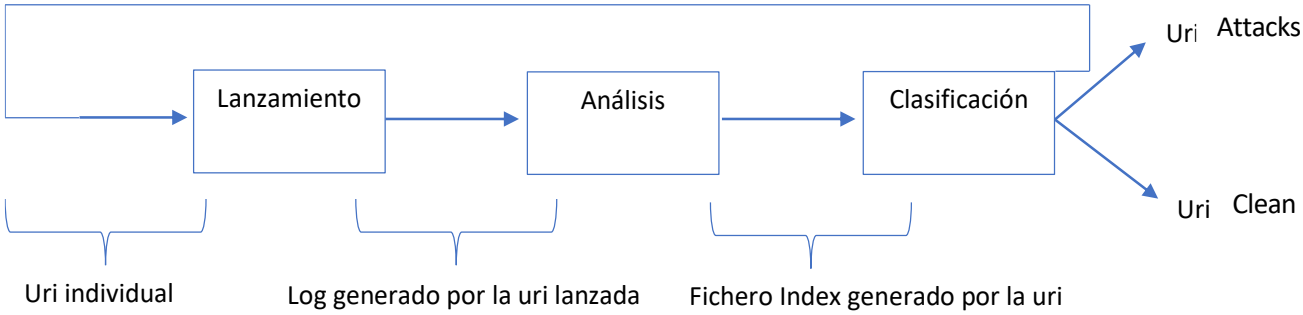


Ilustración 10 Esquema de funcionamiento modo 1to1

Nota1: cuando una uri se considera *'clean'* el *log* para esa uri, y por tanto el *index*, estarán vacíos.

Nota2: los resultados obtenidos con ambas modalidades (1to1 y múltiple **deberán ser idénticos**).

Este método es menos eficiente que el múltiple, pero nos permite determinar con precisión el resultado para esa uri en cada una de las fases del análisis debido a que el proceso se está llevando a **cabo con una sola uri**, y por ende, todo lo generado en cada una de las diferentes fases le corresponde de manera inequívoca a dicha uri.

También es importante destacar en este punto que el modo *'1to1',* además de para tener un proceso más controlado en el lanzamiento (hecho del que pueden beneficiarse futuros IDS que se implementen con la herramienta), surgió con el propósito de investigar si el análisis individualizado de las uris mejoraba el rendimiento frente al análisis conjunto. Más adelante, en el capítulo de resultados, comprobaremos que esto no es así.

Realmente, en una fase más temprana del desarrollo en la cual el analizador estaba realizado en Shell-script, el modo *'1to1'* sí era más rápido que el modo *'multiple';* sin embargo, descubrimos que debida a la naturaleza de Shell-script en la cual se requiere **abrir multitud de procesos para llevar a cabo los análisis de los logs, dichos análisis se demoraban demasiado en el tiempo,** constituyendo un cuello de botella importante**.**

Debido a lo anterior, optamos por emplear Python en los analizadores mejorando muy significativamente los tiempos. La mejora fue tal, que desplazamos el cuello de botella de la fase de análisis, a la del lanzamiento, haciendo que la modalidad *'multiple'* fuese más eficiente, y relegando la modalidad *'1to1'* a un segundo plano, originando que solo sea recomendable su uso en aquellos casos en los que se demande un control muy preciso del proceso de análisis.

3.2. Tipos de lanzamiento:

3.2.1. Offline: tipo de lanzamiento en el cual se realiza el procesado de las uris en ausencia del servidor. El lanzamiento y la carga de reglas que darán lugar al log se realizan de manera local a través de una API.

3.2.2. Online-local: tipo de lanzamiento en el cual las uris se lanzan contra un servidor local que cuenta con un software de detección. Tras el lanzamiento, este software de detección integrado en el servidor es el responsable de la generación del log.

Nota: una vez se produce el lanzamiento de las uris, el programa detiene su ejecución a la espera de la escritura completa del log por parte del software encargado de la detección. De esta forma nos aseguramos que el log se ha escrito por completo y de manera correcta.

3.2.3. Online-remoto: similar al caso anterior, pero en este caso el elemento que cuenta con el software de detección se encuentra en un equipo remoto (caso CheckScript visto en capítulos anteriores). El esquema de funcionamiento sería el siguiente:

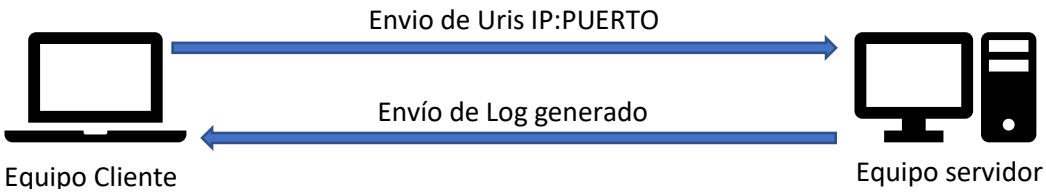


Ilustración 11 Esquema de funcionamiento tipo online-remoto

El equipo cliente realiza el envío de las uris a una IP y puertos determinados y se mantiene a la espera. El equipo servidor estará a la escucha en ese puerto y generará el log correspondiente. Dicho log será enviado al equipo cliente, y este procederá a su análisis y posterior clasificación de las uris.

4. **Capacidad 'multi-instancia':** uno de los mecanismos que han posibilitado una mejora muy significativa de los tiempos de análisis es el de *'multi-instancia'*.

En estudios previos a este proyecto se comprobó que la ejecución de una instancia de análisis consumía muy poca CPU, de forma que se planteó la posibilidad de ejecutar varias instancias de un mismo detector simultáneamente, con el propósito de paralelizar los análisis y hacer un uso más eficiente de la capacidad de cómputo de los equipos.

Conseguir dotar a ACD de esta capacidad en el tipo de lanzamiento *'offline'* es sencillo, puesto que solo se requiere copiar los directorios del proyecto tantas veces como instancias se requieran, y ejecutar el script que desencadena todo el proceso (0-Framework.sh). El lanzamiento de las uris y carga de reglas se hace de forma relativa, por lo que las diferentes instancias pueden funcionar de manera autónoma sin interceder con las demás.

En el caso *'online'* es algo más complejo. Si tuviésemos un servidor único para todas las instancias las uris se entremezclarían haciendo imposible su reconstrucción y clasificación posterior. Para evitar esto, hemos conseguido incorporar ciertos elementos de los detectores al directorio del proyecto, de manera que exista un detector único e independiente para cada instancia.

Cuando hacemos uso de un detector *'online'* -que recordemos, son aquellos que requieren la presencia de un servidor con un software de detección- se procede a arrancar dicho servidor localmente **en el primer puerto libre disponible a partir de uno fijado en la configuración de la herramienta**.

Es decir, imaginemos que queremos arrancar apache, el cual contendrá el módulo de seguridad de ModSecurity. Si en la configuración de la herramienta tenemos fijado el puerto 80 se comprobará si este puerto está disponible. Si lo está, se arranca en ese puerto, si no lo está, se comprueba el puerto 81 y así sucesivamente hasta encontrar el siguiente puerto libre. El servidor empleado y el puerto de escucha utilizado se almacenan en el *'log'* de ACD.

Una vez hallado el puerto se modificarán en tiempo de ejecución los archivos de configuración tanto del servidor como del módulo de seguridad con la información del puerto obtenida, y se procederá al arranque del servidor. Más adelante, en el *Anexo B* entraremos en más detalles sobre cómo es este proceso para cada uno de los detectores integrados.

Cuando finaliza el análisis de los ficheros de entrada, se invoca a un script (detener_servidor_instancia.sh) que detiene el servidor.

5. **2-analyzer.py:** analizador del log generado por el detector. Puesto que el formato del log depende del detector que lo origina, será necesario establecer un analizador por cada detector a implementar.

6. **3-classify.py:** es una parte común a todos los detectores. Se encarga de clasificar las uris como 'attacks' o 'clean' en función de la información obtenida en las etapas anteriores. El algoritmo por el que se rige es el siguiente:

   - Se obtiene la uri de la primera línea del fichero de entrada.

   - Se compara con uri de la primera línea del fichero de index:

     - Diferentes: la uri de entrada se anota en el fichero *'clean'* y se repite el proceso con la siguiente uri presente en el fichero de entrada.

     - Iguales: la uri de entrada se anota en fichero 'attacks'. Se comprueba si se ha llegado al final del fichero index:

       - Fichero finalizado: todas las uris restantes presentes en el fichero de entrada se anotan en el fichero *'clean'.*
       - Fichero no finalizado: avanzamos puntero de lectura de fichero *'index'* a siguiente línea y repetimos el proceso.

   Nota: es importante mencionar que, mientras que en la modalidad offline no existe un proceso de lanzamiento contra un servidor, dado que las uris se analizan mediante la carga local de unas determinadas reglas, y por tanto no es necesario realizar modificación alguna sobre las uris de entrada, en la modalidad online sí es necesario como ya vimos en punto anteriores, realizar una mínima codificación de dichas uris para que estas puedan lanzarse como tráfico http contra un servidor.

   Esta alteración de las uris propia de la modalidad online requiere que cuando realicemos la comparación entre las uris del index y las del fichero de entrada, codifiquemos las uris de entrada para poder obtener la misma uri que llegó al servidor y que por tanto la comparación tenga sentido.

A continuación se muestra un diagrama de flujo del proceso:

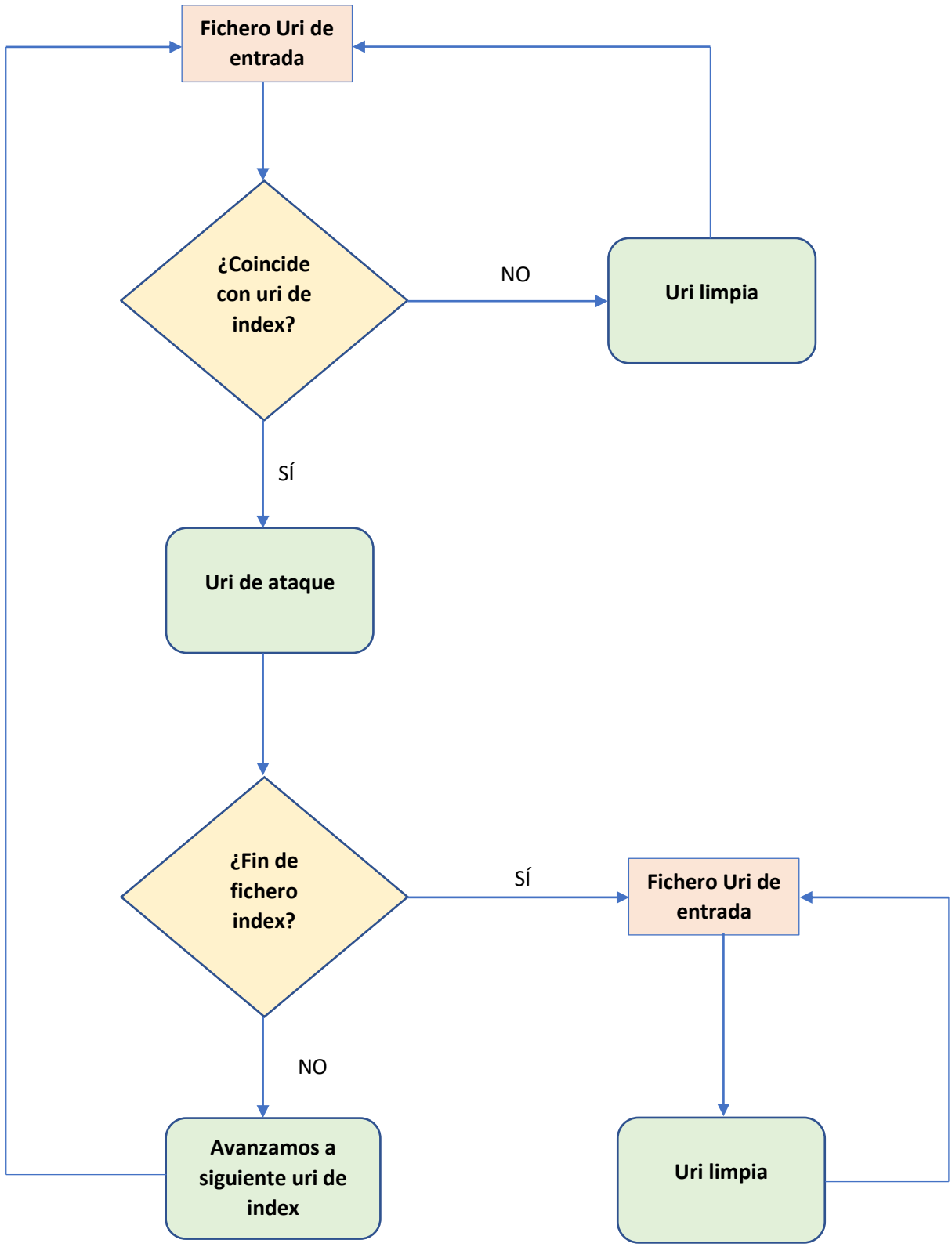


Ilustración 12 Algoritmo de clasificación de ACD

4 APLICACIÓN DISTRIBUIDORA DE TAREAS (ADT)

ADT (Aplicación Distribuidora de Tareas) es un software que permite el despliegue, ejecución, recogida de resultados y control de estado de una tarea en un conjunto de equipos remotos.

El propósito que cumple ADT en nuestro proyecto es el de otorgarnos una herramienta que, por una parte nos conceda mejoras en cuanto a calidad de vida en el uso de ACD, pero también, y como objetivo principal, darnos la capacidad de distribuir eficazmente nuestro software en un conjunto de equipos remotos designados y aportarnos información útil sobre el estado de ejecución de la tarea.

Aunque la implementación de ADT en este proyecto responde a una necesidad que teníamos de distribuir y controlar ACD en una granja de equipos para acelerar las detecciones, esta tiene un propósito **general**, es decir, todas aquellas tareas que como ACD cumplan con una serie de requisitos, pueden hacer uso de ADT y disfrutar de las ventajas que nos brinda.

Los requisitos que debe cumplir una tarea para ser integrables en ADT son los siguientes:

- Debe poseer una estructura de manera que reciba uno o varios ficheros de entrada y genere una o varias salidas en base al procesado de el/los fichero(s) de entrada.

- Cuando la tarea en cuestión finaliza con el análisis de una o más entradas, debe escribir un fichero de texto vacío con el nombre de el/los fichero(s) de entrada ya procesados en el directorio 'entradas_finalizadas' creado automáticamente dentro del directorio raíz de la tarea en cuestión.

- Los resultados a recoger tendrán que almacenarse en un directorio que deberá indicarse en el fichero de configuración de la tarea.

Un esquema de funcionamiento simplificado que ilustra el funcionamiento de ADT sería el siguiente:

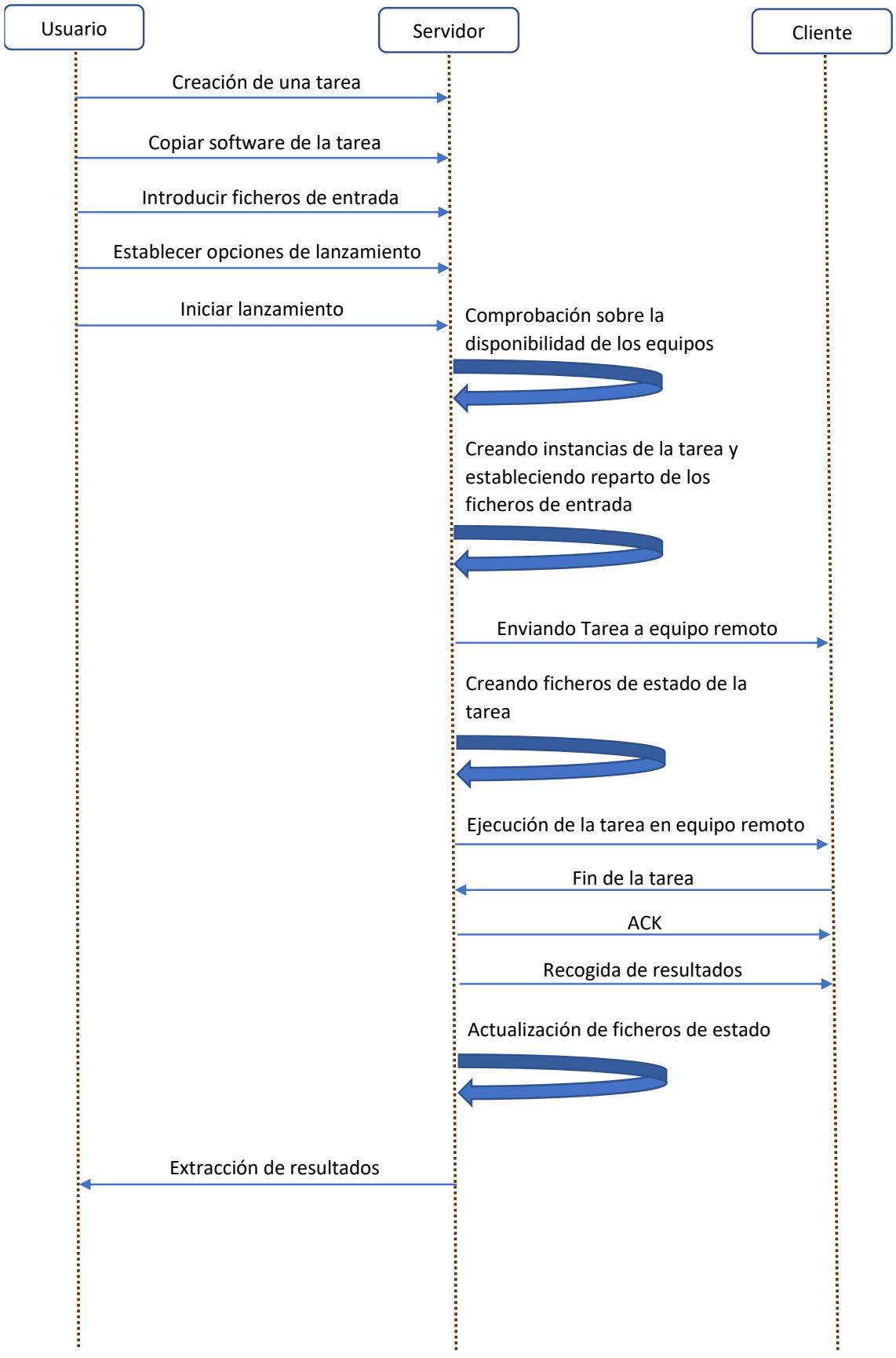


Ilustración 13 Esquema de funcionamiento de ADT

4.1 Funcionamiento y características

Aunque más adelante -en los anexos dedicados a ADT- comentaremos con detalle todas las características y configuraciones posibles de ADT, resulta interesante expresar con un mayor grado de detalle algunos aspectos sobre las posibilidades ofrecidas por esta.

4.1.1 Menús

El primer punto a comentar es que ADT cuenta con un menú global a todas las tareas que nos permite lo siguiente:

1. **Mostrar estado de los equipos:** realiza un análisis de los equipos indicados en la configuración y nos indica la disponibilidad de estos.
2. **Apagar equipos remotos:** apaga los equipos seleccionados en el menú de forma remota.
3. **Crear Tarea:** crea una tarea.
4. **Borrar Tarea:** elimina una tarea.
5. **Seleccionar Tarea:** accede al menú de una tarea ya existente.
6. **Mostrar Tareas:** muestra las tareas disponibles.
7. **Lanzar escucha:** lanza en segundo plano el script de escucha. Dicho script abrirá un puerto seleccionado en la configuración de ADT, y se mantendrá a la espera de un mesaje UDP por parte de los equipos clientes informando sobre la disponibilidad de uno o más resultados para recoger.
8. **Detener escucha:** detiene el script de escucha.

Otro aspecto importante a destacar es que, además del menú global, cada tarea presenta un menú propio que solo afecta a esa tarea. Esta misma estructura de menú se repite para los ficheros de configuración; existe uno general para toda la ADT denominado ‘cloud_config_interna.conf’ y otro específico para cada tarea que recibe el nombre de ‘cloud_tarea.conf’.

Una vez creada una tarea, el menú específico de la misma presenta las siguientes opciones:

1. **Lanzamiento completo:** inicia el lanzamiento de la tarea a los equipos remotos designados en la configuración.
2. **Consultar estado:** una vez realizado el lanzamiento, podemos comprobar en tiempo real mediante esta opción el estado en el que se encuentra la tarea.
3. **Recoger resultados:** recogida manual de los directorios de resultados de los equipos seleccionados.
4. **Matar tarea:** detiene la tarea de los equipos seleccionados.
5. **Limpiar directorios en equipos remotos:** elimina los directorios de la tarea desplegados en los equipos remotos.
6. **Limpiar tarea:** elimina todos los ficheros de estado generados para esa tarea.
7. **Fusionar ficheros:** fusiona los resultados obtenidos de los ficheros fragmentados durante el lanzamiento. El estado de los resultados tras la fusión debe ser idéntico al que se daría si los ficheros de entrada no hubiesen sido fragmentados.
8. **Ejecutar o Enviar ficheros en los equipos remotos:** permite la ejecución de comandos en los equipos remotos y la subida de ficheros hacia los mismos, así como la descarga desde estos.

4.1.2 Fusionador de ficheros

En este punto es de imperiosa necesidad detallar una de las funcionalidades más útiles implementadas dentro de ADT y que más ha contribuido a mejorar los tiempos de ACD. Esta funcionalidad es la fragmentación de los ficheros de entradas y la posterior fusión de los resultados obtenidos. Esto nos permite obtener exactamente los mismos resultados que si no hubiésemos realizado la fragmentación de los ficheros de entrada, pero con la ventaja de poder distribuir un fichero de entrada extenso entre varios equipos. De esta forma, podemos paralelizar el procesamiento del mismo y mejorar los tiempos de ejecución de las tareas.

El proceso sería el siguiente:

1. Usuario crea una tarea.

2. Copia el software de dicha tarea en el directorio adecuado.

3. Introduce los ficheros de entrada. En este punto el usuario cuenta con dos posibilidades:

    a. Introduce los ficheros de entrada en su directorio correspondiente para que posteriormente se realice el reparto entre los equipos designados.

    b. Además de introducir los ficheros en el directorio del apartado 'a', el usuario cuenta con la opción de introducirlos en otro directorio de entrada adicional. Los ficheros introducidos en este directorio adicional serán fragmentados en tantas partes como se haya establecido en la configuración de ADT. Los fragmentos de los ficheros serán traspasados al directorio del apartado 'a' para que posteriormente sean repartidos entre los equipos disponibles.

        Nota: los ficheros fragmentados reciben un nombre que los relaciona con el fichero original, así como un número que identifica a qué parte del fichero original corresponde esa división. Por ejemplo: si dividimos un fichero llamado '0days.uri' en dos partes, el primero de los fragmentos recibirá por nombre '0days_000.uri', y el segundo '0days_001.uri'**. El número máximo de divisiones permitidas para un fichero es de 1000.**

4. Usuario establece la configuración deseada para la tarea.

5. Usuario Ejecuta el lanzamiento de la tarea.

6. Una vez los ficheros de entrada hayan sido procesados y el usuario haya obtenido los resultados, puede iniciar el proceso de fusión de los mismos. La reconstrucción se produce de forma que los resultados tras la fusión son indistinguibles a los obtenidos si no hubiésemos realizado la fragmentación de los ficheros de entrada.

4.1.3 Protocolo de comunicación

Con el objetivo de automatizar el proceso de recogida de los resultados de los equipos remotos en los que ha tenido lugar el despliegue de la tarea, se ha diseñado el siguiente protocolo de comunicación:

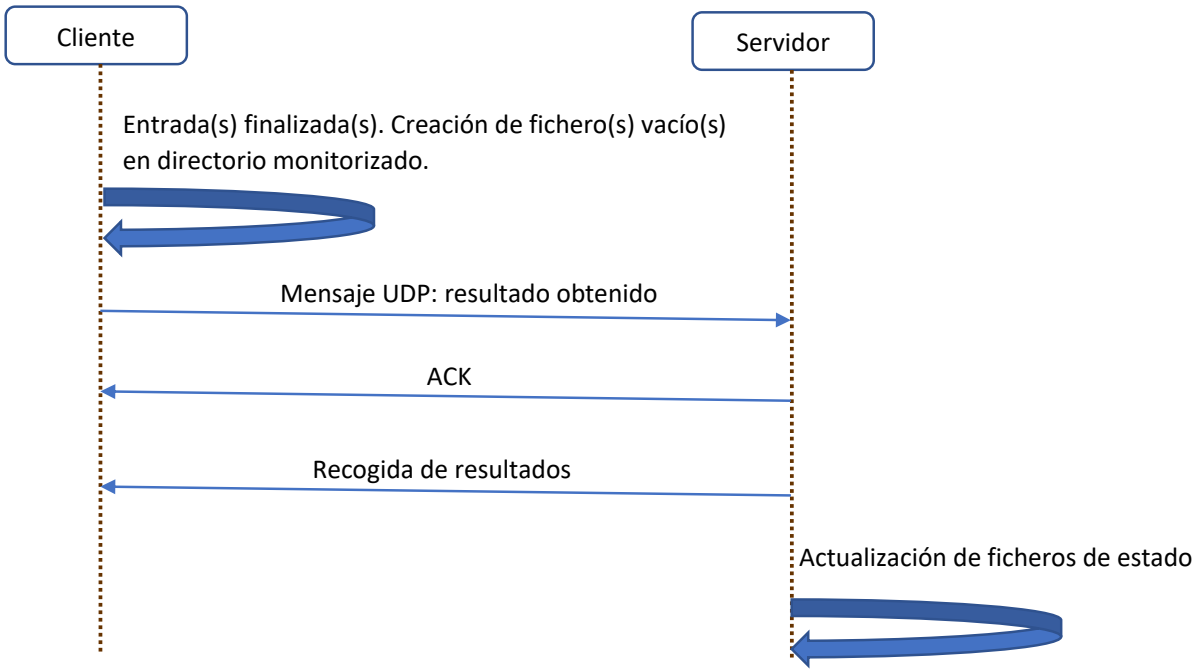


Ilustración 14 Protocolo de comunicación de ADT

En la 'Ilustración 12', 'cliente' hace referencia al equipo remoto en el que se ha desplegado la tarea, y 'servidor' al equipo que contiene a ADT.

Adentrándonos con más detalle en el protocolo de comunicación diseñado, el procedimiento que se sigue es el siguiente:

Nota: a partir de este momento llamaremos 'S' al equipo servidor que contiene a ADT y 'C' al equipo cliente en el que se ejecutan las tareas.

Cuando desde S ejecutamos una tarea en C, no solo se realiza la ejecución de dicha tarea en una sesión 'byobu' (la sesión byobu nos permite que el proceso se ejecute en una sesión en segundo plano, de forma que un usuario ajeno al proyecto pueda hacer un uso normal del equipo sin ser consciente de la ejecución de la tarea) sino que también se ejecuta un script de monitorización en segundo plano.

Dicho script se encarga de la monitorización de un directorio determinado fijado en la configuración interna de ADT, en el cual la tarea deberá escribir un fichero de texto vacío con el nombre de el/los fichero(s) de entrada que haya terminado de procesar, y para los que ha generado uno o más resultados.

Cuando se produce la escritura de dichos ficheros se desencadena el envío de un mensaje UDP al servidor. Dicho mensaje contiene la siguiente información:

- IP de C
- Puerto de escucha de C
- Nombre de la tarea desplegada
- Bit de FIN. Se enviará FIN = 1 para informar a S que la ejecución de la tarea en C ha concluido.

IP CLIENTE	PUERTO ESCUCHA CLIENTE	NOMBRE TAREA	FIN

Ilustración 15 Formato del mensaje UDP

Con esta información S puede ejecutar el script de recogida de resultados de esa tarea en el equipo e instancia correspondientes. Una vez la recogida se efectúa, S actualiza los ficheros de estado para esa tarea.

Nota1: con el fin de mejorar los tiempos y optimizar la capacidad de cómputo de los equipos, ADT permite el despliegue y ejecución ordenados de múltiples instancias para una misma tarea en un mismo equipo. El número de instancias que se despliegan para una tarea dada es configurable.

Por ejemplo: si quisiéramos que cada C albergarse 5 instancias de la tarea *ACD*, se crearían, enviarían y ejecutarían en cada equipo C las tareas: *ACD1, ACD2, ACD3, ACD4* y ACD*5*.

Nota2: el script de escucha en S es activado con el lanzamiento y ejecución de la primera tarea. Este script puede detenerse usando la opción *'8 Detener escucha'* del menú global.

Nota3: si tras el envío de un mensaje UDP, C no recibe un ACK por parte de S en un tiempo determinado establecido en la configuración de ADT, C procederá al reenvío del mensaje. El número de intentos para hacer llegar el mensaje a S también es establecido en la configuración de ADT.

4.1.4 Notificación al usuario vía e-mail

La aplicación puede configurarse para notificar al usuario bajo dos circunstancias:

1. La tarea ha terminado, pudiendo ser con o sin errores. En el cuerpo del mensaje se detalla el resultado de la tarea.
2. Fallo en uno/varios equipos. En el cuerpo se detallaría el equipo/lista de equipos que han dejado de estar disponibles de forma que el usuario pueda emprender las acciones correspondientes.

4.1.5 Esquema de componentes de ADT

De un modo similar a lo visto en la *ilustración 5* del capítulo 3 para ACD, concluiremos este apartado con un esquema de componentes de ADT que nos permita hacernos una idea sobre las relaciones existentes entre las diferentes partes que la conforman.

En este esquema se contempla la creación de una tarea, su posterior envío y ejecución en un equipo remoto, y por último, la recogida de los resultados obtenidos y la actualización de estado de la tarea.

Nota: la zona delimitada por la línea anaranjada es ejecutada en el equipo cliente/remoto, mientras que el resto de scripts serán ejecutados en el equipo servidor/local que contiene a la herramienta ADT.

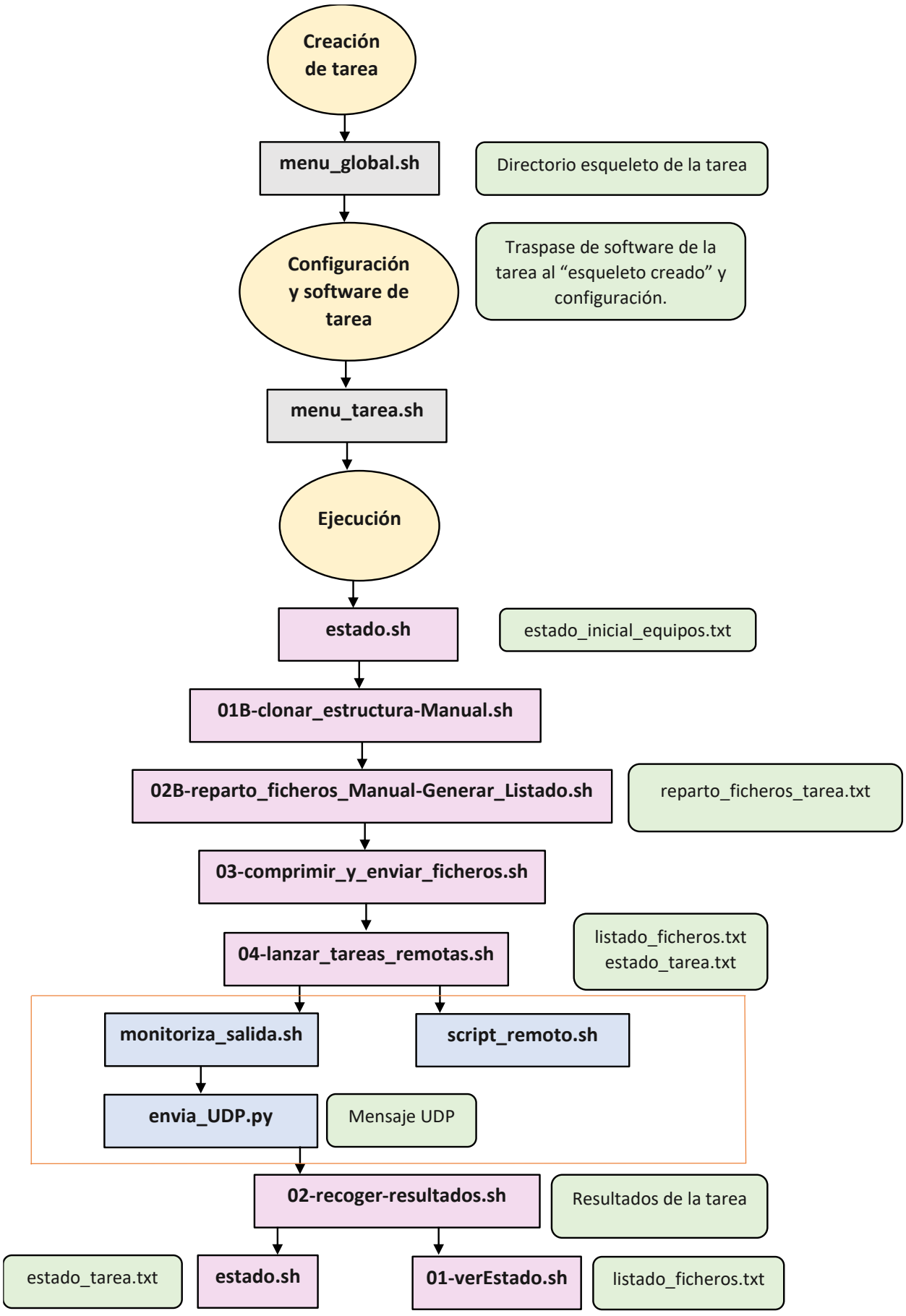


Ilustración 16 Esquema de componentes de ADT

5 VALIDACIÓN DE RESULTADOS

En este apartado del proyecto representaremos los resultados obtenidos tras analizar los 'Datasets' definidos en los objetivos del mismo. Para la obtención de dichos resultados se han empleado y han trabajado conjuntamente los dos softwares desarrollados para este trabajo (ACD y ADT).

5.1 Descripción del escenario de trabajo

5.1.1 Mapa de red

Para la consecución de los análisis se han hecho uso de **28 equipos de laboratorio**. A continuación, se detallará el escenario sobre el que ha tenido lugar las pruebas y la obtención de resultados. El mapa de red de los equipos es el siguiente:

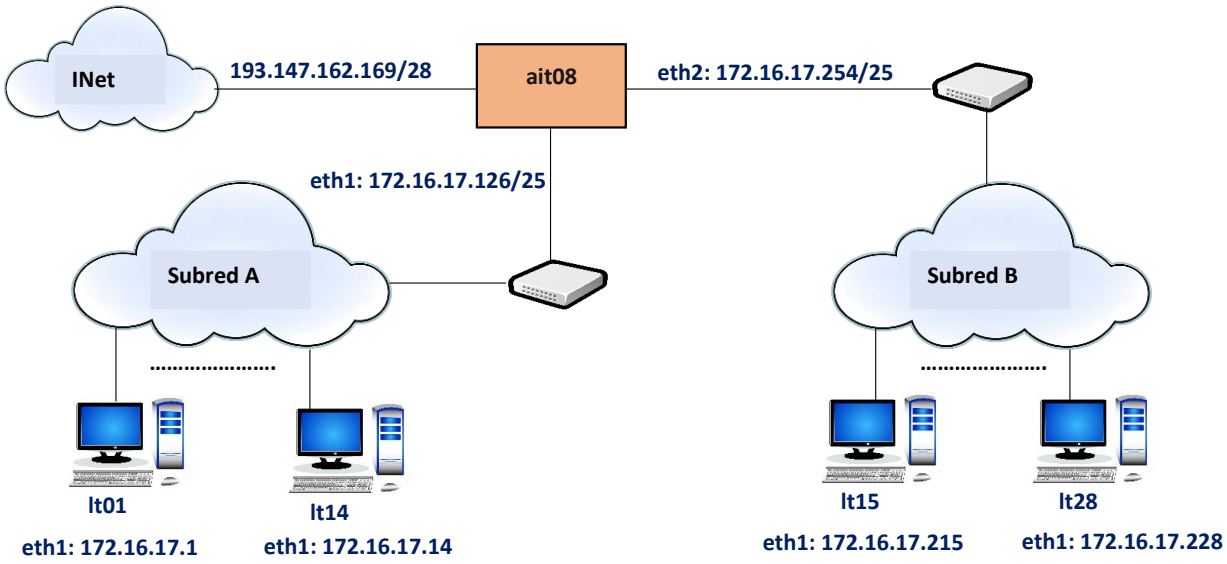


Ilustración 17 Mapa de red de los equipos

Subred	Máscara	Rango IPs	IP Equipos	Equipo
A	255.255.255.128	172.16.17.0-127 IP Subred: 172.16.17.0 IP Broadcast: 172.16.17.127 Libres: 172.16.17.15-124	172.16.17.1	lt01
			…	…
			172.16.17.14	lt14
			172.16.17.126	ait08.us.es
B	255.255.255.128	172.16.17.128-255 IP Subred: 172.16.17.128 IP Broadcast: 172.16.17.255 Libres: 172.16.17.129-214 172.16.17.229-252	172.16.17.215	lt15
			…	…
			172.16.17.228	lt28
			172.16.17.254	ait08.us.es

Tabla 11 Subredes A/B del laboratorio

5.1.2 Equipos

Todos los equipos usados cuentan con el mismo Software y Hardware. En la siguiente tabla se recogen las principales características de los mismos:

<table>
<tr><th colspan="3">Hardware</th><th>SO</th><th>Software</th></tr>
<tr><td>CPU</td><td colspan="2">Intel(R) Core(TM) i3-4130 CPU @ 3.40 GHz</td><td rowspan="6">CentOS 7</td><td rowspan="6">Apache/2.4.6 (CentOS)
ModSecurityV2.7
ModSecurityV3.0
Nginx-1.18.0
Nwaf-1.18
Inotify-tools
Sshpass
Python36
Byobu
Dialog
git</td></tr>
<tr><td rowspan="2">Memoria RAM</td><td rowspan="2">12 GiB</td><td>8GiB DDR3 1600 MHz</td></tr>
<tr><td>4GiB DDR3 1600 MHz</td></tr>
<tr><td>GPU</td><td colspan="2">GeForce GT 730</td></tr>
<tr><td rowspan="2">Disco duro</td><td colspan="2">/dev/sda 1024GB SPCC Solid State</td></tr>
<tr><td colspan="2">/dev/sdb 500GB TOSHIBA DT01ACA0</td></tr>
</table>

Tabla 12 Principales Características de los equipos

5.1.3 IDS

Con el fin de aclarar y agilizar la lectura a partir de este momento, estableceremos una nomenclatura para cada uno de los IDS implementados que se mantendrá hasta el final de este documento:

- **IL-NEMESIDA:** IDS 'Nemesida' implementado dentro de la herramienta IL (Inspector Log).
- **IL-SNORT:** IDS 'Snort' implementado dentro de la herramienta IL.
- **IL-MODSECV3-PLX:** IDS 'ModSecurity' implementado en su versión 3 dentro de la herramienta IL. El sufijo 'PLX' determina el nivel de paranoia (Paranoia Level) con el que se ha llevado a cabo la detección.
- **MLA-MODSECV2-PLX:** IDS 'ModSecurity' en su versión 2. Esta implementación requiere de un servidor Apache. El sufijo 'PLX' determina el nivel de paranoia (Paranoia Level) con el que se ha llevado a cabo la detección.
- **MLA-MODSECV3-PLX:** IDS 'ModSecurity' en su versión 3. El sufijo 'PLX' determina el nivel de paranoia (Paranoia Level) con el que se ha llevado a cabo la detección.
- **NEMESIDASCRIPT:** implementación del IDS 'Nemesida' basada en el proyecto 'NemesidaScript' [20]. Requiere de servidor Nginx.

En la siguiente tabla se muestran los IDS así como los modos de funcionamiento, tipos de lanzamiento y reglas para los que se han extraído los resultados:

Detector	Modo de funcionamiento	Tipo de lanzamiento	Reglas
MLA-MODSECV2	online-local	múltiple	owasp-modsecurity-crs-3.2.0 (modificadas)
MLA-MODSECV3	offline	múltiple	owasp-modsecurity-crs-3.2.0 (modificadas)
IL-MODSECV3	offline	múltiple	owasp-modsecurity-crs-3.2.0 (modificadas)
IL-SNORT	offline	múltiple	http_uri-er-20220224.rules http_uri-sn-20220324.rules
IL-NEMESIDA	offline	múltiple	Nemesida-rules-bin-20220109.txt
NEMESIDASCRIPT	online-local	múltiple	rules.bin

Tabla 13 Resumen de detectores utilizados

Nota1: el fichero de reglas 'rules.bin' cargado para el detector 'Nemesida' es el que viene por defecto para la instalación de 'Nemesida' en su versión '1.18'.

Nota2: los ficheros de reglas de 'owasp-modsecurity-crs-3.2.0' son una versión modificada de la versión original. Estas reglas han sido proporcionadas por el tutor de este trabajo *Francisco Javier Muñoz Calle*.

5.1.4 Datasets

Los 'Datasets' para los cuales se han realizado los análisis han sido los mostrados Tabla 3 Datasets analizados

5.2 Resultados obtenidos

Importante: con el fin de compactar los datos, y dado que todas las detecciones de 'IL-MODSECV3' coinciden por completo con las obtenidas para el IDS 'MLA-MODSECV3, hemos optado por omitir de los resultados las detecciones de 'IL-MODSECV3' para no ser tan redundantes.

También, con objeto de mejorar la legibilidad de los datos, utilizaremos a lo largo de este apartado **en todas las tablas y gráficas 'MODSECV3' para referirnos a 'MLA-MODSECV3', y 'MODSECV2' para referirnos a 'MLA-MODSECV2',** de forma que por ejemplo: 'MODSECV3-PL1' estaría haciendo referencia a una detección en 'PL=1' de la implementación 'MLA-MODSECV3'.

Nota1: todas las representaciones gráficas de este apartado se han hecho de manera que, **la barra amarilla parte del origen (0) hasta el número total de uris disponibles en el dataset en cuestión,** mientras que la barra naranja indica el número de detecciones registradas. Con esto se pretende representar visualmente la relación entre detecciones/uris totales. También se ha añadido en la parte derecha de cada gráfica un porcentaje que representa esta misma relación al que hemos denominado 'eficiencia'.

Nota2: las detecciones obtenidas **tienen en cuenta las uris repetidas**, es decir, si una uri presente en un dataset se encuentra repetida 'n' veces, y dicha uri es detectada como attacks, se reportarán 'n' attacks.

5.2.1 Datasets de ataques

5.2.1.1 0days_A-420

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	148	35	420
NEMESIDASCRIPT	160	35	
IL-SNORT	284	67	
MODSECV3-PL1	134	32	
MODSECV2-PL1	135	32	
MODSECV3-PL2	346	82	
MODSECV2-PL2	347	83	
MODSECV3-PL3	368	88	
MODSECV2-PL3	369	88	
MODSECV3-PL4	379	90	
MODSECV2-PL4	379	90	

Tabla 14 Detecciones 0days_A-420

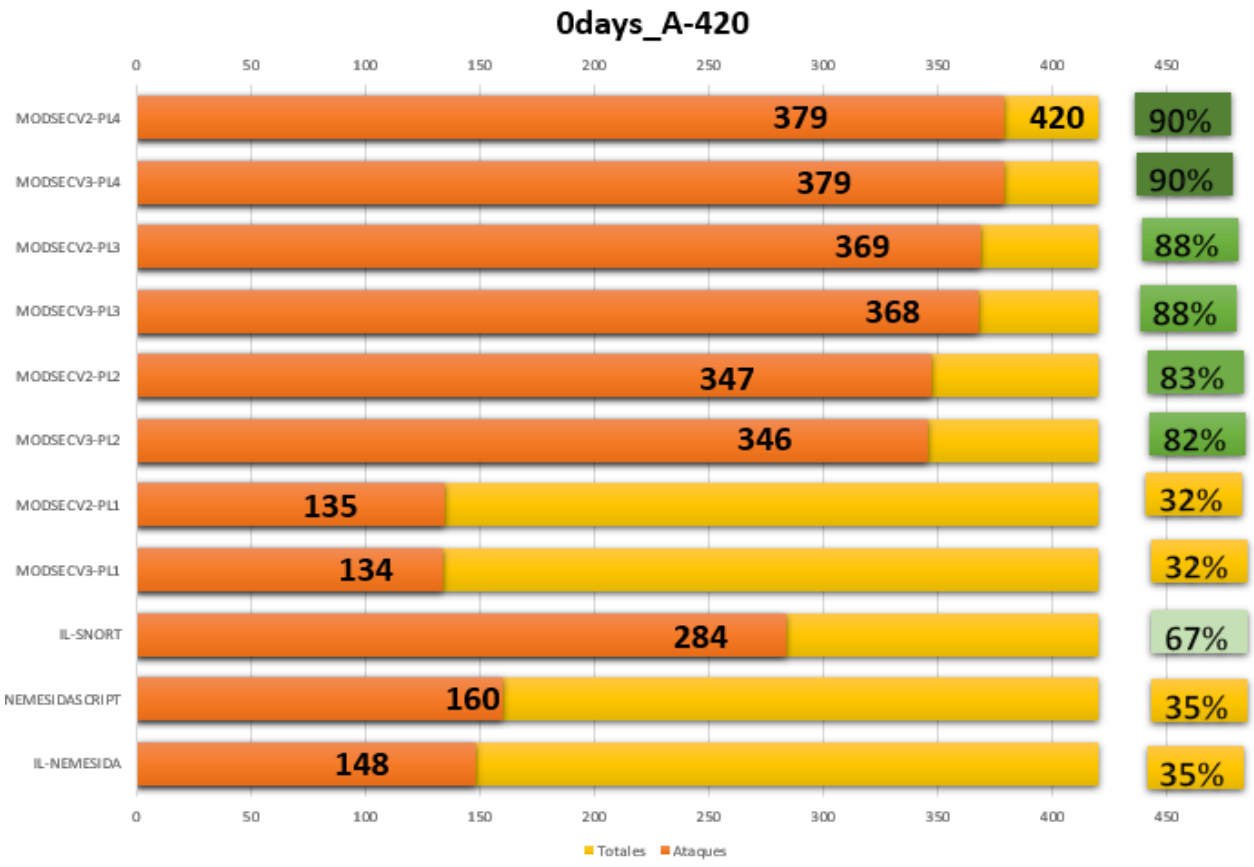


Ilustración 18 Detecciones 0days_A-420

5.2.1.2 0days_revisado

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	126	32	397
NEMESIDASCRIPT	132	33	
IL-SNORT	90	23	
MODSECV3-PL1	114	29	
MODSECV2-PL1	114	29	
MODSECV3-PL2	147	37	
MODSECV2-PL2	147	37	
MODSECV3-PL3	217	55	
MODSECV2-PL3	217	55	
MODSECV3-PL4	353	89	
MODSECV2-PL4	353	89	

Tabla 15 Detecciones 0days_revisado

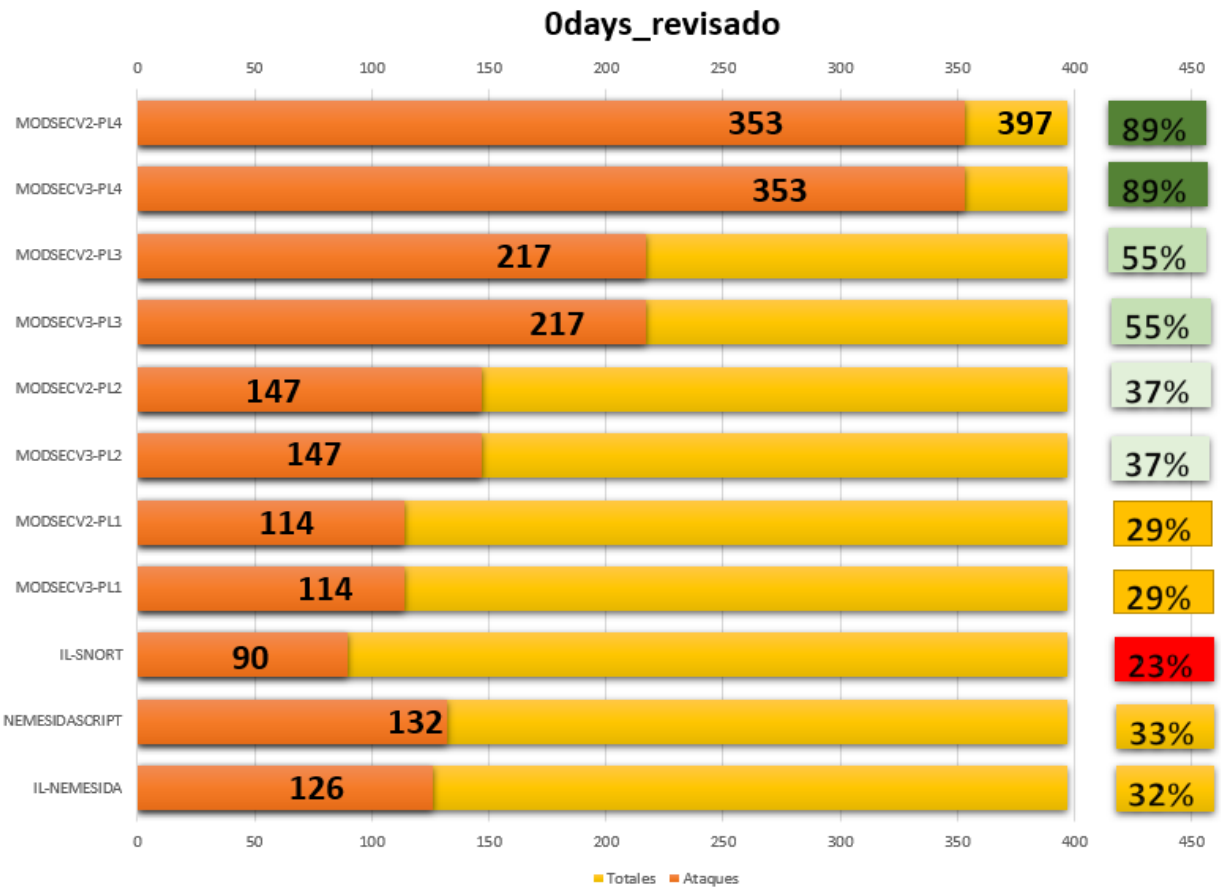


Ilustración 19 0days_revisado

5.2.1.3 Ataques-800

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	591	71	833
NEMESIDASCRIPT	72	9	
IL-SNORT	136	16	
MODSECV3-PL1	485	58	
MODSECV2-PL1	457	55	
MODSECV3-PL2	532	64	
MODSECV2-PL2	504	61	
MODSECV3-PL3	545	65	
MODSECV2-PL3	517	62	
MODSECV3-PL4	600	72	
MODSECV2-PL4	574	69	

Tabla 16 Detecciones Ataques-800

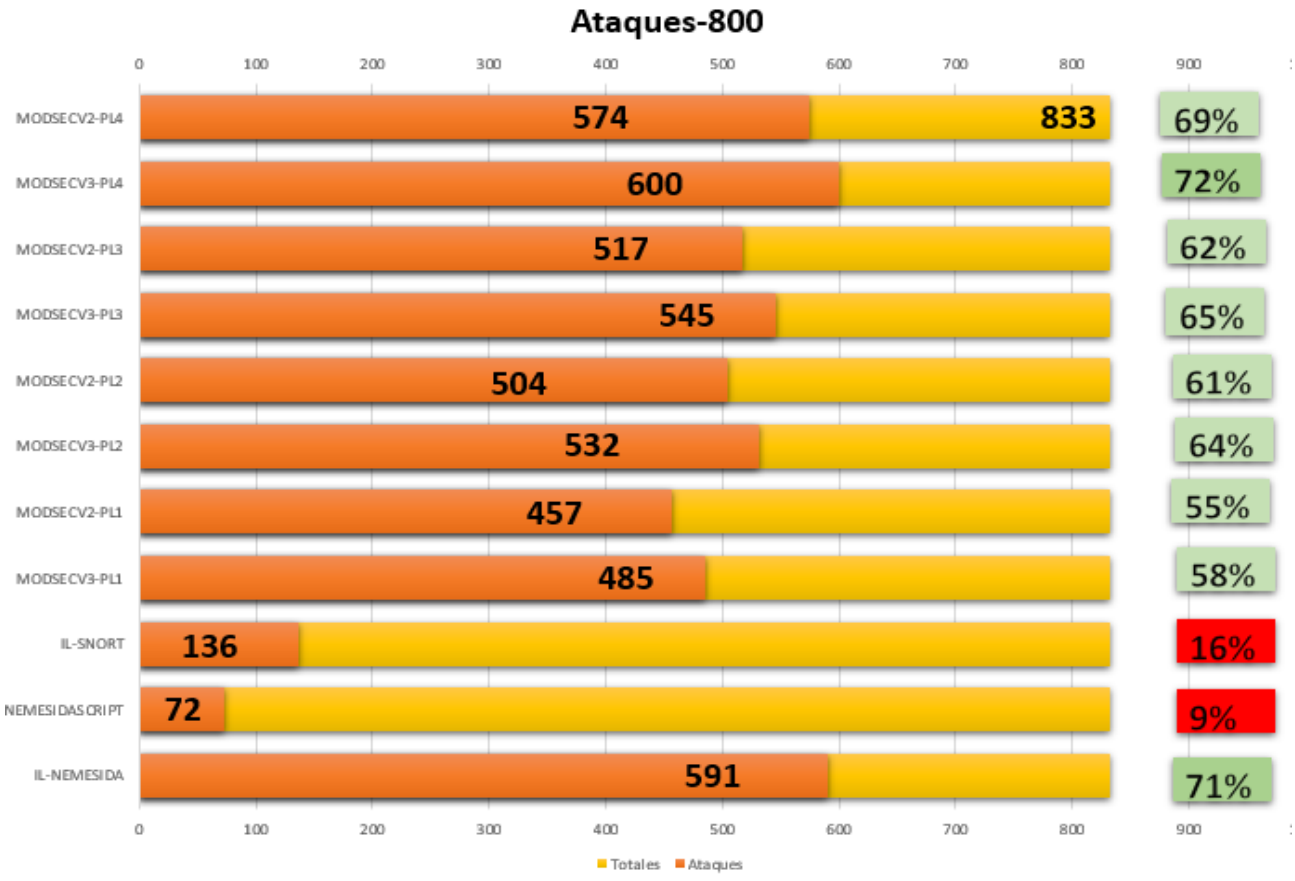


Ilustración 20 Detecciones Ataques-800

5.2.1.4 Ataques-1100

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	872	74	
NEMESIDASCRIPT	158	13	
IL-SNORT	304	26	
MODSECV3-PL1	781	66	
MODSECV2-PL1	760	65	
MODSECV3-PL2	840	71	
MODSECV2-PL2	820	70	
MODSECV3-PL3	859	73	1177
MODSECV2-PL3	839	71	
MODSECV3-PL4	915	78	
MODSECV2-PL4	897	76	

Tabla 17 Detecciones Ataques-1100

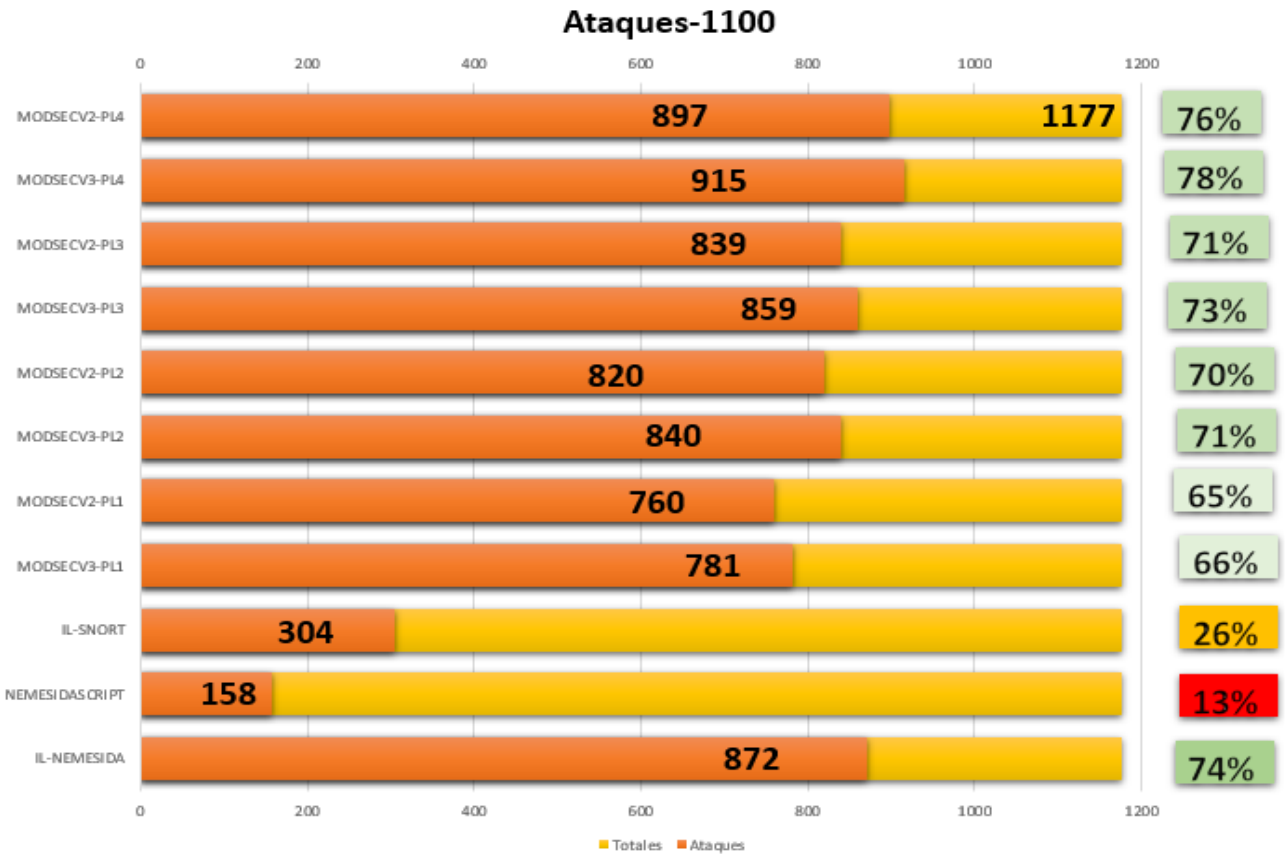


Ilustración 21 Detecciones Ataques-1100

5.2.1.5 Fwaf-2200

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	2152	98	2200
NEMESIDASCRIPT	2089	95	
IL-SNORT	1329	60	
MODSECV3-PL1	2004	91	
MODSECV2-PL1	1937	88	
MODSECV3-PL2	2042	93	
MODSECV2-PL2	1971	90	
MODSECV3-PL3	2068	94	
MODSECV2-PL3	1994	91	
MODSECV3-PL4	2082	95	
MODSECV2-PL4	2012	91	

Tabla 18 Detecciones Fwaf-2200

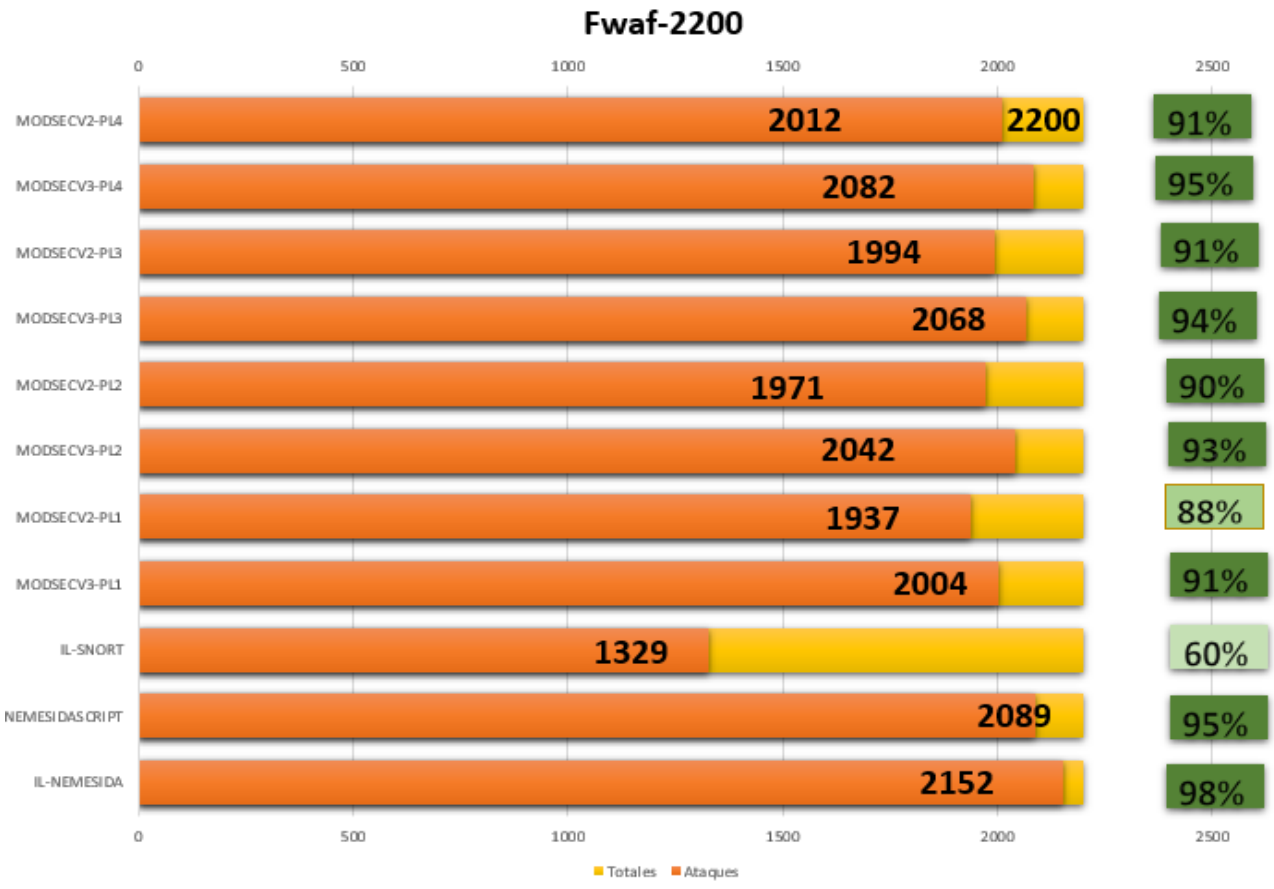


Ilustración 22 Detecciones Fwaf-2200

5.2.1.6 Fwaf-badqueries

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	25210	52	48065
NEMESIDASCRIPT	22412	47	
IL-SNORT	7200	15	
MODSECV3-PL1	22304	46	
MODSECV2-PL1	21451	45	
MODSECV3-PL2	24514	51	
MODSECV2-PL2	23594	49	
MODSECV3-PL3	25179	52	
MODSECV2-PL3	24160	50	
MODSECV3-PL4	31022	65	
MODSECV2-PL4	30152	63	

Tabla 19 Detecciones Fwaf-badqueries

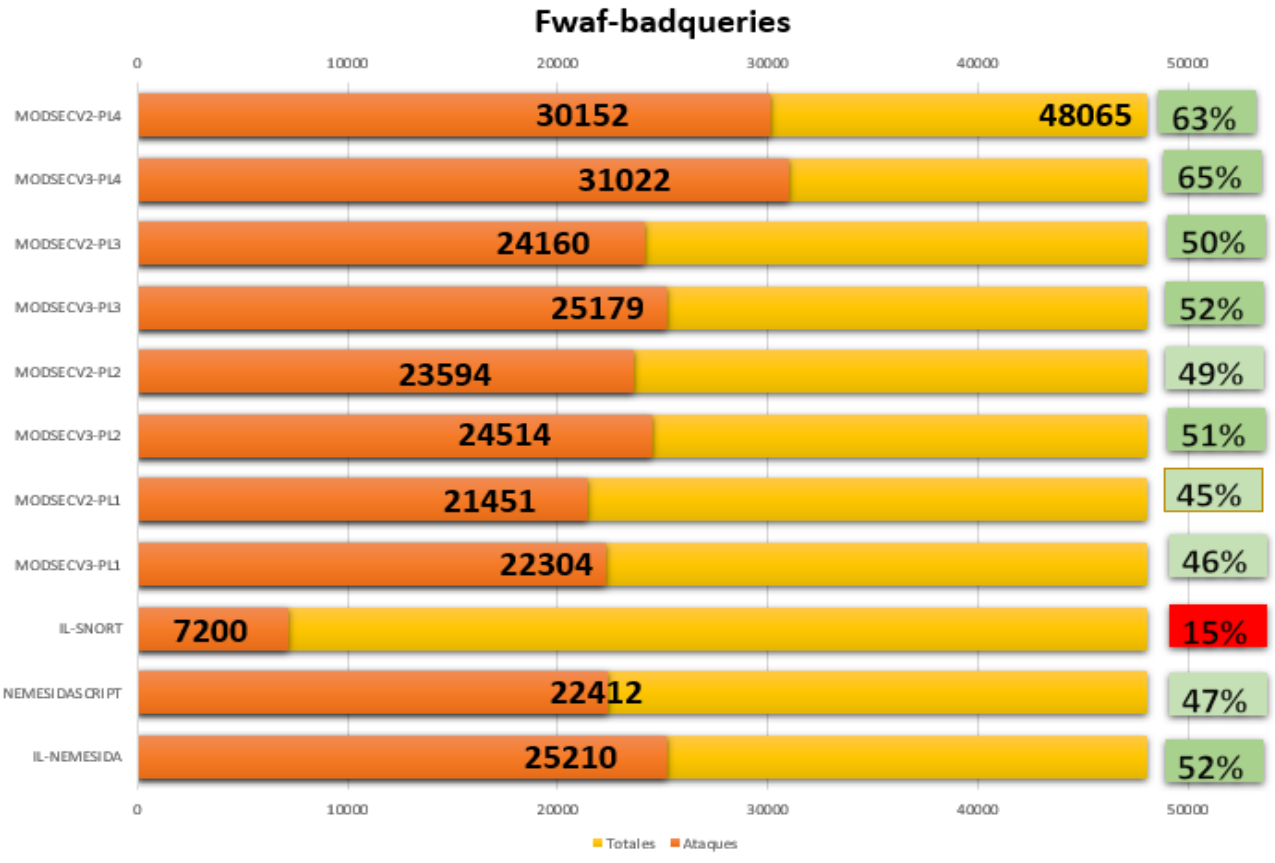


Ilustración 23 Detecciones Fwaf-badqueries

5.2.1.7 Osvdb

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	3612	52	6896
NEMESIDASCRIPT	3169	52	
IL-SNORT	2340	34	
MODSECV3-PL1	4288	62	
MODSECV2-PL1	4215	61	
MODSECV3-PL2	5015	73	
MODSECV2-PL2	4942	72	
MODSECV3-PL3	5048	73	
MODSECV2-PL3	4975	72	
MODSECV3-PL4	5667	82	
MODSECV2-PL4	5594	81	

Tabla 20 Detecciones Osvdb

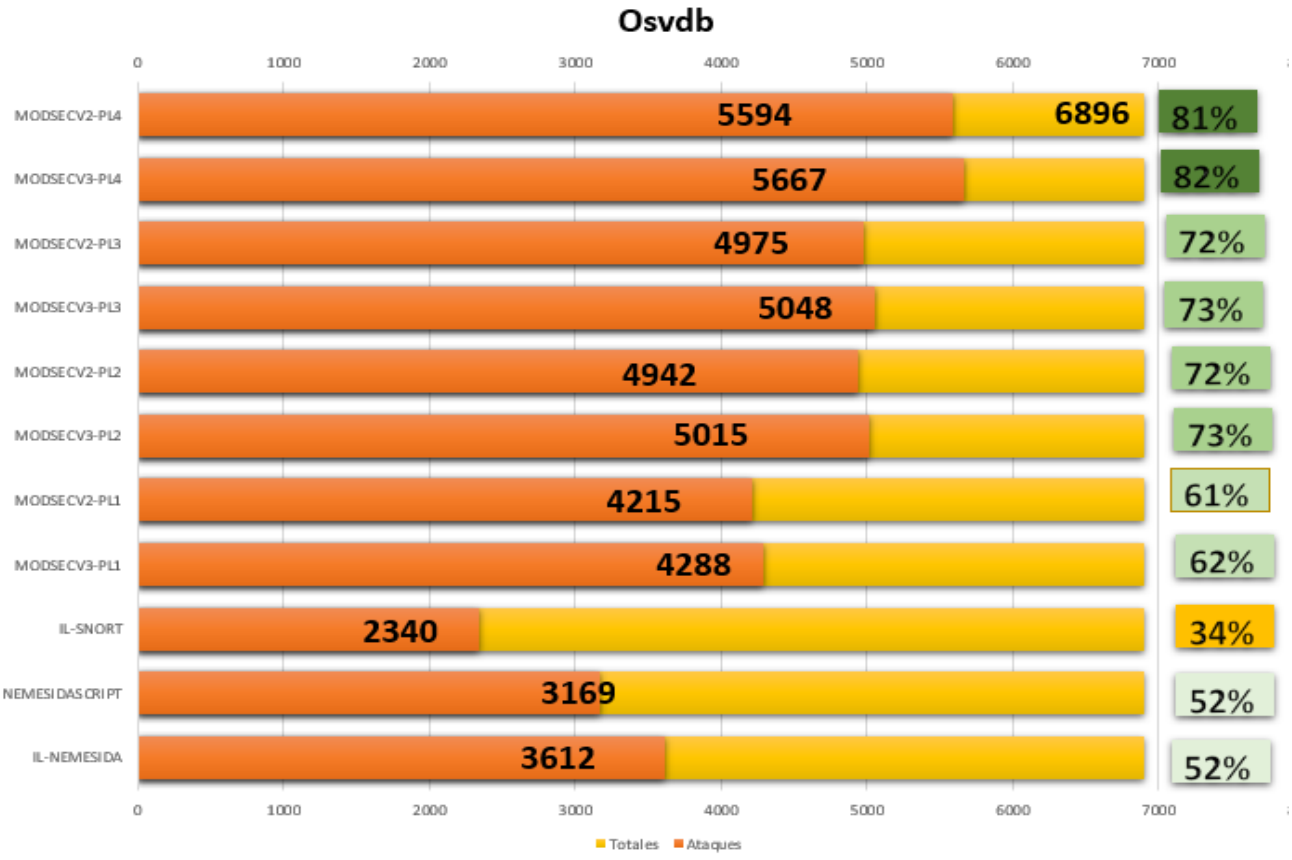


Ilustración 24 Detecciones Osvdb

5.2.1.8 Rdb

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	623	67	933
NEMESIDASCRIPT	116	12	
IL-SNORT	121	13	
MODSECV3-PL1	508	54	
MODSECV2-PL1	481	52	
MODSECV3-PL2	560	60	
MODSECV2-PL2	533	57	
MODSECV3-PL3	568	61	
MODSECV2-PL3	539	58	
MODSECV3-PL4	614	66	
MODSECV2-PL4	587	63	

Tabla 21 Detecciones Rdb

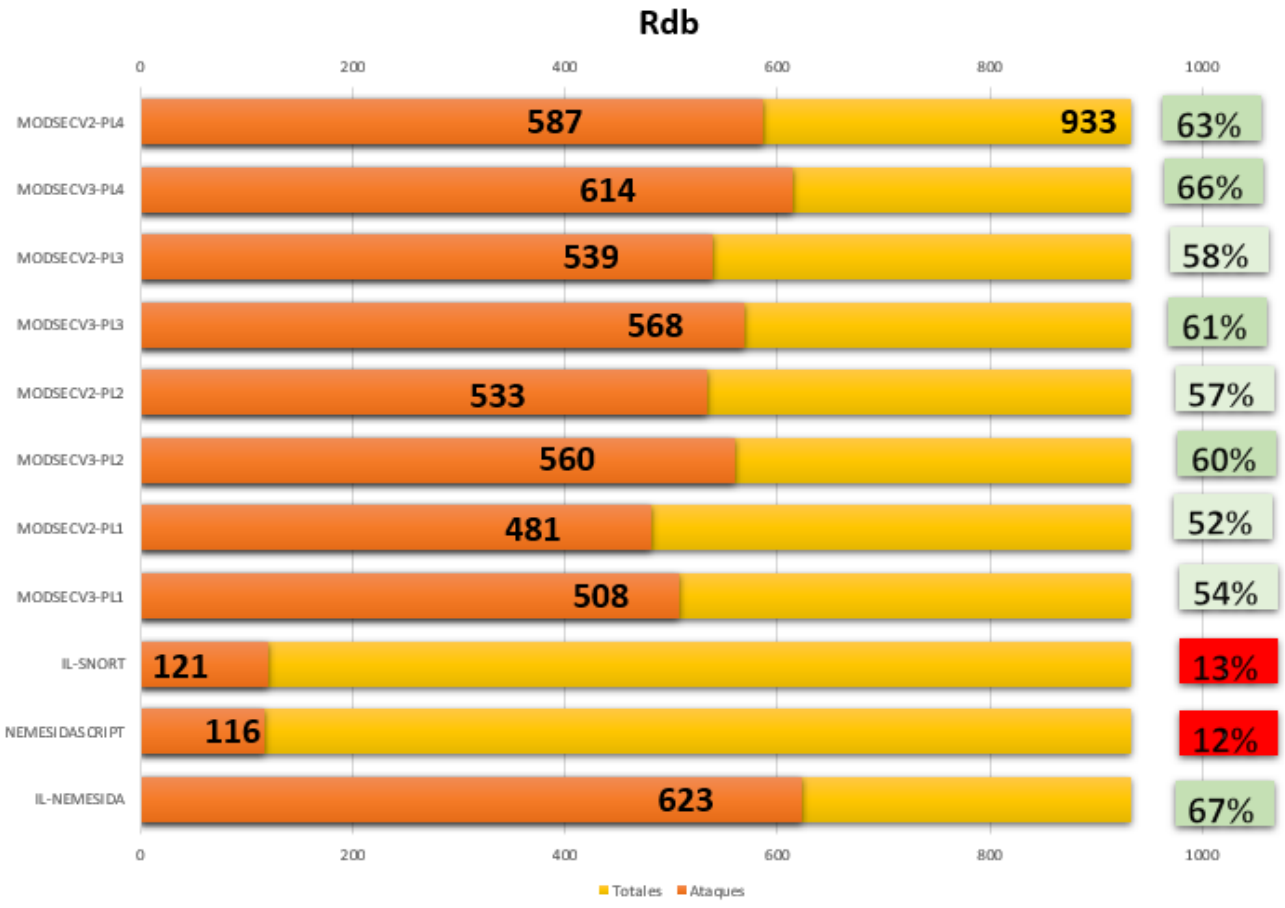


Ilustración 25 Detecciones Rdb

5.2.2 Datasets reales

5.2.2.1 BIBLIO_ANONIMIZADO

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	18704	0	47402996
NEMESIDASCRIPT	2914	0	
IL-SNORT		0	
MODSECV3-PL1		0	
MODSECV2-PL1	2094	0	
MODSECV3-PL2	2517	0	
MODSECV2-PL2	8996	0	
MODSECV3-PL3	578732	1.2	
MODSECV2-PL3	424149	0.9	
MODSECV3-PL4	596756	1.3	
MODSECV2-PL4	442299	0.9	

Tabla 22 Detecciones BIBLIO_ANONIMIZADO

Ilustración 26 Detecciones BIBLIO_ANONIMIZADO

5.2.2.2 BIBLIO_ETIQUETADO

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	18483	0	47402996
NEMESIDASCRIPT	2824	0	
IL-SNORT	2523	0	
MODSECV3-PL1	8992	0	
MODSECV2-PL1	2090	0	
MODSECV3-PL2	10074	0	
MODSECV2-PL2	3172	0	
MODSECV3-PL3	578728	1.2	
MODSECV2-PL3	424145	0.9	
MODSECV3-PL4	596752	1.3	
MODSECV2-PL4	442295	0.9	

Tabla 23 Detecciones BIBLIO_ETIQUETADO

Ilustración 27 Detecciones BIBLIO_ETIQUETADO

5.2.2.3 INVES

IDS	Ataques	Eficiencia (%)	Totales
IL-NEMESIDA	2320	0	14149237
NEMESIDASCRIPT	16	0	
IL-SNORT	2425913	17	
MODSECV3-PL1	2260	0	
MODSECV2-PL1	2259	0	
MODSECV3-PL2	3630	0	
MODSECV2-PL2	3629	0	
MODSECV3-PL3	1928554	14	
MODSECV2-PL3	1868703	13	
MODSECV3-PL4	3185066	23	
MODSECV2-PL4	3185066	23	

Tabla 24 Detecciones INVES

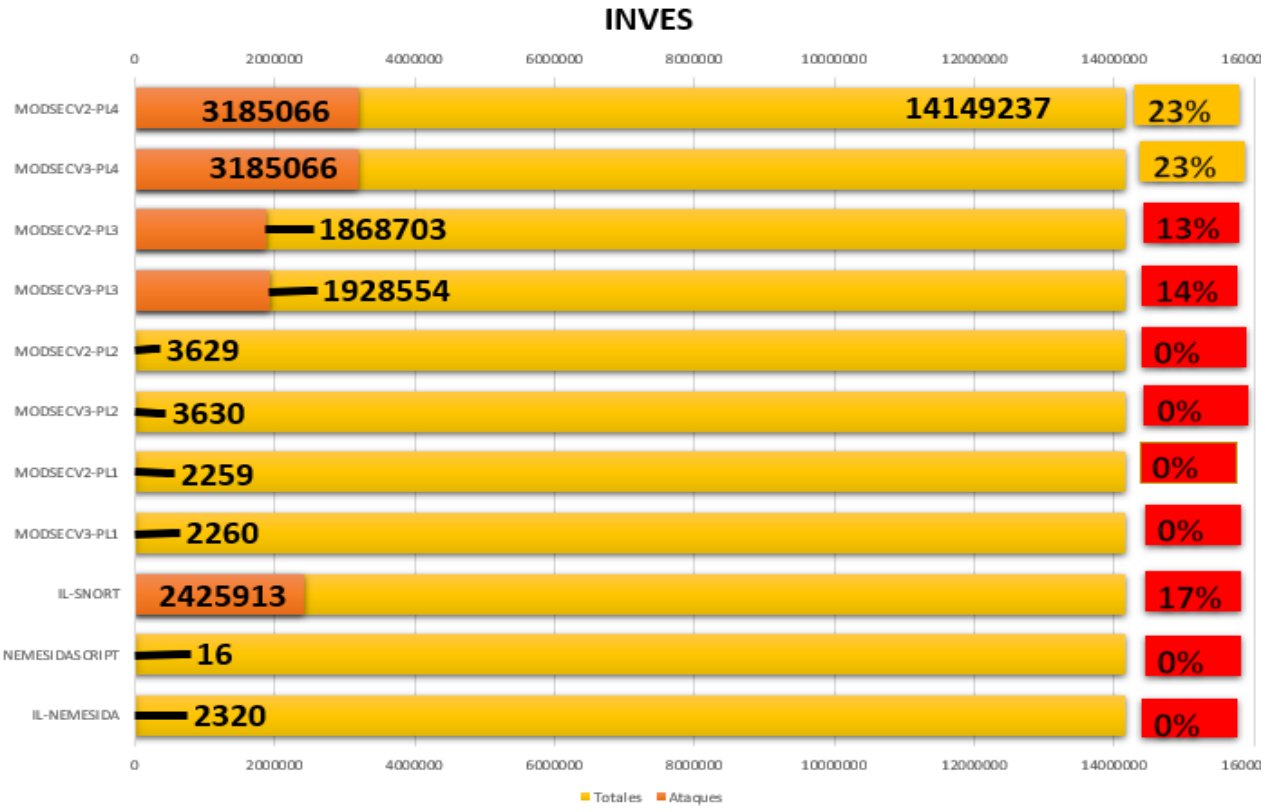


Ilustración 28 Detecciones INVES

5.3 Análisis de los resultados

Una vez hemos mostrado los resultados obtenidos en cuanto a detecciones se refiere, pasaremos a realizar algunos análisis sobre los mismos que nos ayuden a extraer ciertas conclusiones.

Salvo que se indique lo contrario, y *en pos de* la concisión, elegiremos los resultados obtenidos para tres de los Datasets usados en el proyecto. En la elección de estos Datasets hemos procurado condensar la mayor casuística posible. Los Datasets seleccionados pueden verse en la siguiente tabla:

Datasets		Número de uris
Reales	INVES	14151496
De ataques	Fwaf-badqueries	48065
	Ataques-800	833

Tabla 25 Dataset seleccionados para el estudio

5.3.1 ModSecurityV2 Vs ModSecurityV3

En este subapartado compararemos la eficiencia entre la implementación de 'ModSecurityV2' (MLA) y 'ModSecurityV3' en cualquiera de sus implementaciones (IL y MLA. Recordemos que ambas implementaciones de ModSecurityV3 coinciden en todas las detecciones realizadas).

Este estudio es interesante porque al ser dos implementaciones basadas en el mismo IDS (ModSecurity), que cargan las mismas reglas, y que una version (V2) funciona mediante el envío de uris a un servidor (Apache), y la otra version (V3) funciona mediante la carga de reglas offline, nos permite evalúar en cierto modo el efecto sobre las detecciones de codificar las uris en el lanzamiento; ya que recordemos, los IDS que funcionan realizando las detecciones a través de un servidor, requieren de la codificación previa al envío de ciertos caracteres para permitir el lanzamiento de las uris a través de la red.

Si establecemos una tabla comparativa de las eficiencias de MODSECV2 VS MODSECV3 para los 'Datasets' seleccionados obtenemos lo siguiente:

Dataset	Eficiencia (%) MODSECV3 | MODSECV2							
	PL1	PL1	PL2	PL2	PL3	PL3	PL4	PL4
Ataques-800	58	55	64	61	65	62	72	69
Fwaf-badqueries	46	45	51	49	52	50	65	63
INVES	0	0	0	0	14	13	23	23

Tabla 26 Comparativa eficiencia ModSecurityV2 Vs ModSecurityV3

En este estudio puede comprobarse como generalmente ModSecurityV3 obtiene un mayor número de detecciones que su contraparte ModSecurityV2. Esto es achacable a que en los lanzamientos de ModSecurityV2 tiene lugar **la codificación de ciertos caracteres prohibidos, por otros permitidos, que posibilitan el lanzamiento de las uris, de forma que dichos caracteres prohibidos no llegan al servidor, provocando una disminución en las detecciones.**

Este fenómeno puede observarse de un modo más visual en la gráfica siguiente:

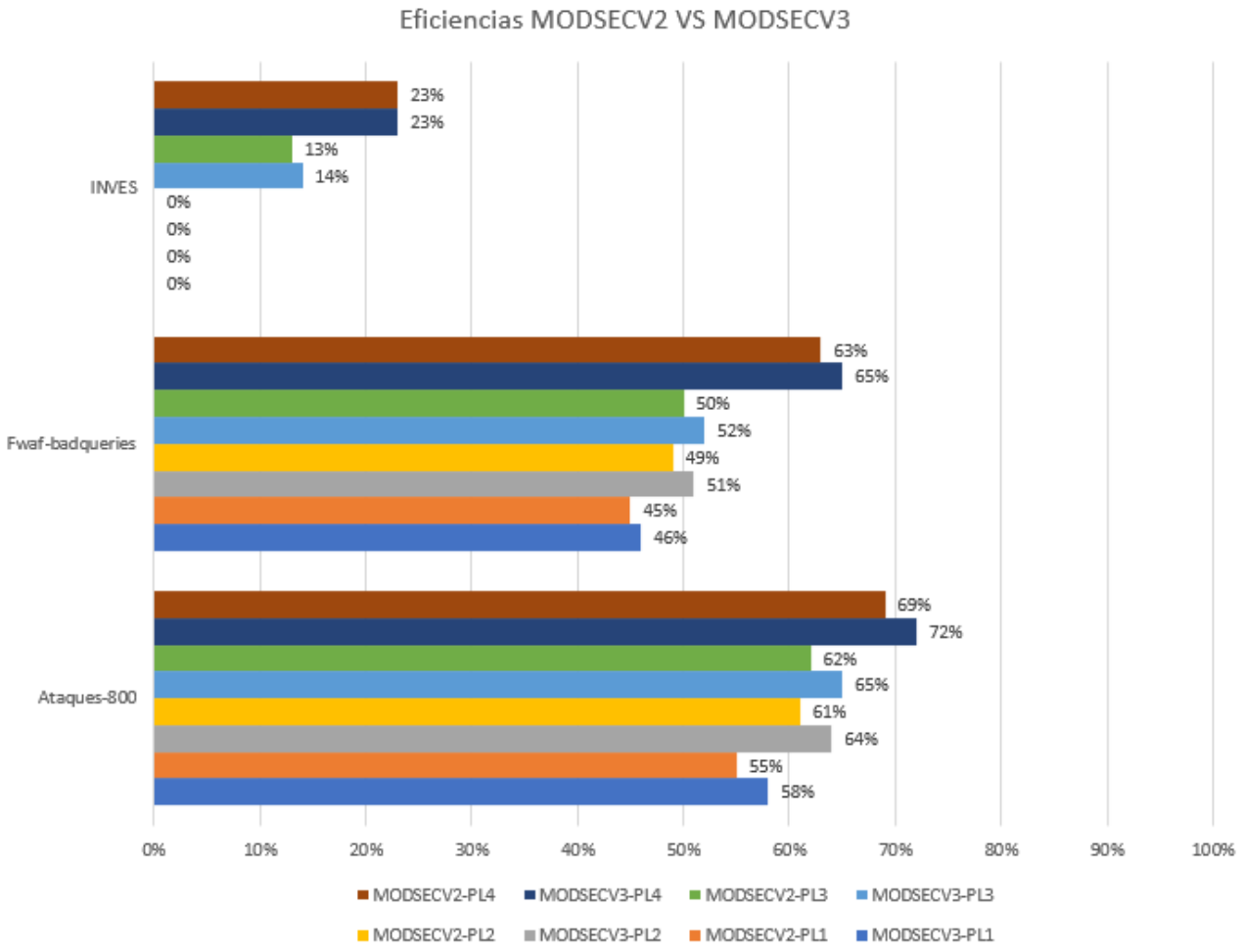


Ilustración 29 Eficiencias MODSECV2 VS MODSECV3

No obstante, tras comparar las detecciones realizadas por cada uno de los IDS se ha observado que, aunque mínimas, en ocasiones la pre-codificación de dichos caracteres también puede generar detecciones **que solo se encuentran en MODSECV2**. A continuación se adjunta una tabla resumen de dicho estudio:

Dataset	Uris presentes solo en MODSECV2		Uris presentes solo en MODSECV3	
	PL1	PL3	PL1	PL3
Ataques-800	0	0	28	28
Fwaf-badqueries	37	45	890	1064
INVES	0	390	1	60241

Tabla 27 Comparativa de uris presentes en cada IDS

Las uris que difieren entre ambas versiones para detecciones en ‘PL=1’ y ‘PL=3’ de estos ‘Datasets’ pueden ser consultados en [21].

5.3.2 Estudio de eficiencia por solapamiento de IDS

En este estudio queremos evaluar el efecto producido por el uso conjunto de varios IDS sobre un mismo Dataset para sopesar en qué grado aumenta la eficiencia en las detecciones al solapar varios IDS.

Para el desarrollo de este apartado nos centraremos en los resultados obtenidos por los IDS:

- MLA/IL-ModSecuriyV3 (a los que seguiremos denominando como 'MODSECV3'). Usaremos los resultados obtenidos en 'PL=1' y 'PL=3'.
- IL-SNORT. Al que para simplificar denominaremos 'Snort'.
- IL-NEMESIDA. Al que para simplificar denominaremos 'Nemesida'.

En primer lugar, observamos que la eficiencia individual obtenida para la combinación de IDS-Dataset empleada en este apartado es la siguiente:

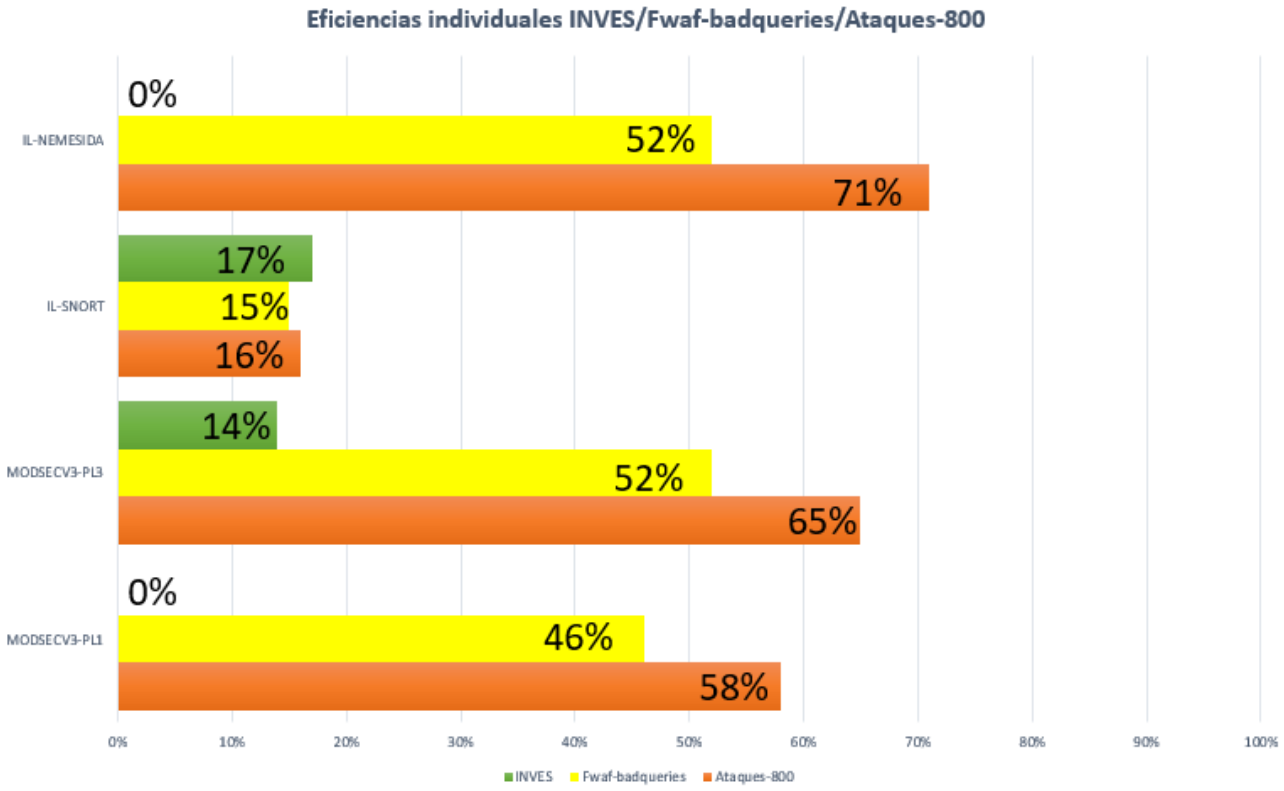


Ilustración 30 Eficiencias individuales INVES/Fwaf-badqueries/Ataques-800

Con esta información, y comparando las diferentes uris detectadas como attacks en cada Dataset por cada uno de los detectores, podemos elaborar la tabla siguiente:

IDS	Ataques-800		Fwaf-badqueries		INVES	
	Uris	Eficiencia (%)	Uris	Eficiencia (%)	Uris	Eficiencia (%)
NEMESIDA+SNORT+PL1	+41	+ ≈5	+6429	+ ≈14	+3448	+ ≈0
NEMESIDA+SNORT+PL3	+75	+ ≈9	+8302	+ ≈18	+431088	+ ≈3
NEMESIDA+SNORT	+11	+ ≈1	+177	+ ≈1	+2232	+ ≈0
NEMESIDA+PL1	+36	+ ≈4	+6316	+ ≈14	+1514	+ ≈0
NEMESIDA+PL3	+74	+ ≈9	+8204	+ ≈18	+1575	+ ≈0
PL1+SNORT	+9	+ ≈1	+545	+ ≈2	+1934	+ ≈0
PL3+SNORT	+7	+ ≈1	+217	+ ≈1	+429513	+ ≈3

Tabla 28 Resumen de detecciones con solapamientos

Nota1**: hemos usado PL1 y PL3** como simplificación de MODSECV3-PL1 y MODSECV3-PL3.

Nota2: el aumento de las uris y las eficiencias se han calculado en base al detector que más alertas ha generado. Por ejemplo, si tomamos el caso del Dataset 'Ataques-800' para el solapamiento entre los IDS 'NEMESIDA+SNORT+PL3' observamos un aumento en las detecciones de 75 uris y de aproximadamente un 9% en la eficiencia. **Estos aumentos son con respecto al IDS que más detecciones generó**, que en este caso sería con respecto a 'NEMESIDA'.

De la tabla anterior podemos obtener las siguientes conclusiones:

1. En los 'Datasets de ataques' podemos ver que la combinación entre IDS que provocan cierta mejora en el número de detecciones es cuando se combinan 'NEMESIDA+MODSECURITY', especialmente cuando se combinan 'NEMESIDA+PL3'. La combinación 'NEMESIDA+SNORT+PLX' apenas aporta mejora alguna, ya que el desempeño de Snort en los 'Datasets' de ataques ha sido significativamente inferior al resto de IDS de esta comparativa.

2. A diferencia de los 'Dataset de ataques' el 'Dataset real' (INVES) ha obtenido la mayor mejora con la combinación 'PL3+SNORT'. La combinación 'NEMESIDA+SNORT+PL3' apenas obtiene mejora alguna sobre 'PL3+SNORT', ya que NEMESIDA apenas ha generado detecciones.

En la gráfica siguiente puede apreciarse el aumento en la eficiencia al solapar los tres detectores (NEMESIDA+SNORT+PL3) en el caso más beneficioso para cada uno de los 'Datasets'.

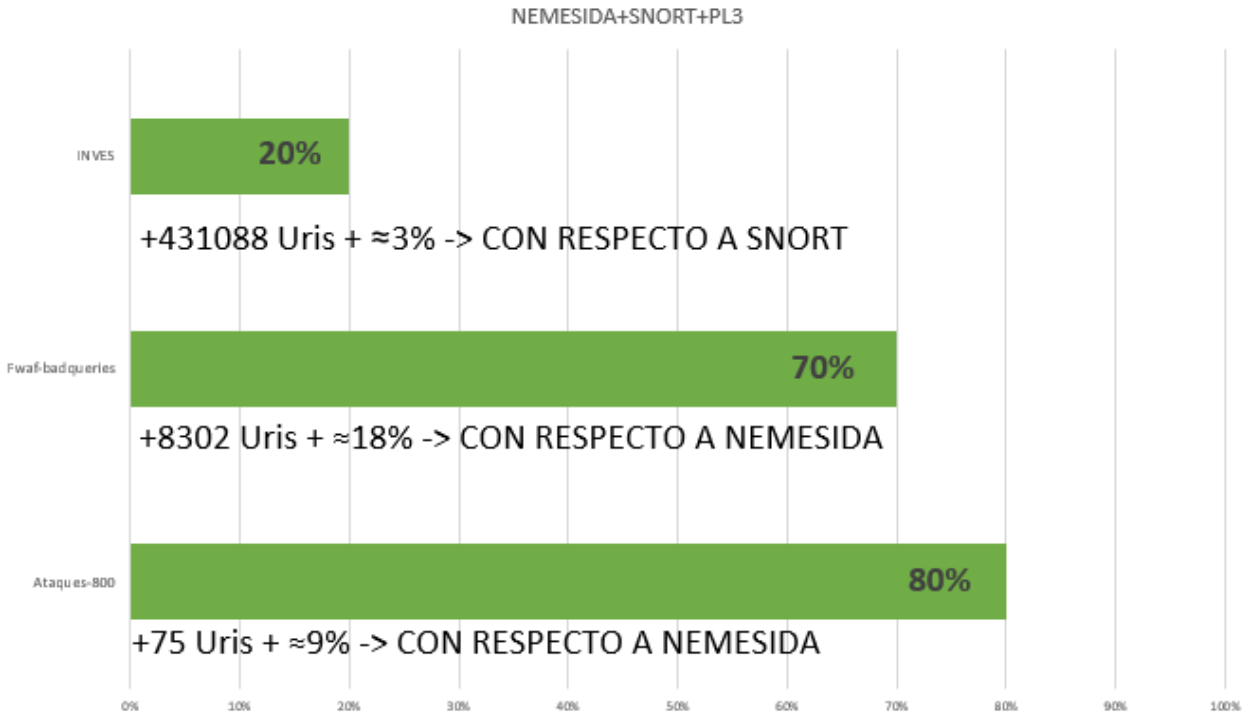


Ilustración 31 Aumento en la eficiencia por solapamiento

Como puede observarse se muestra la eficiencia obtenida con el solapamiento, junto a la mejora con respecto al caso más favorable del uso de un solo IDS.

De este análisis podemos extraer que el uso conjunto de IDS proporciona una mejora en las detecciones frente a un uso individual de cada uno de ellos. Habría que evaluar en cada caso en función de la naturaleza del 'Dataset' si este 'esfuerzo' estaría justificado, ya que al menos en el estudio que nos ocupa, el aumento en las detecciones de los 'Dataset reales' ha sido mínima, puesto que el número de alertas registradas han sido muy bajas, a diferencia de lo visto en los 'Dataset de ataques' analizados.

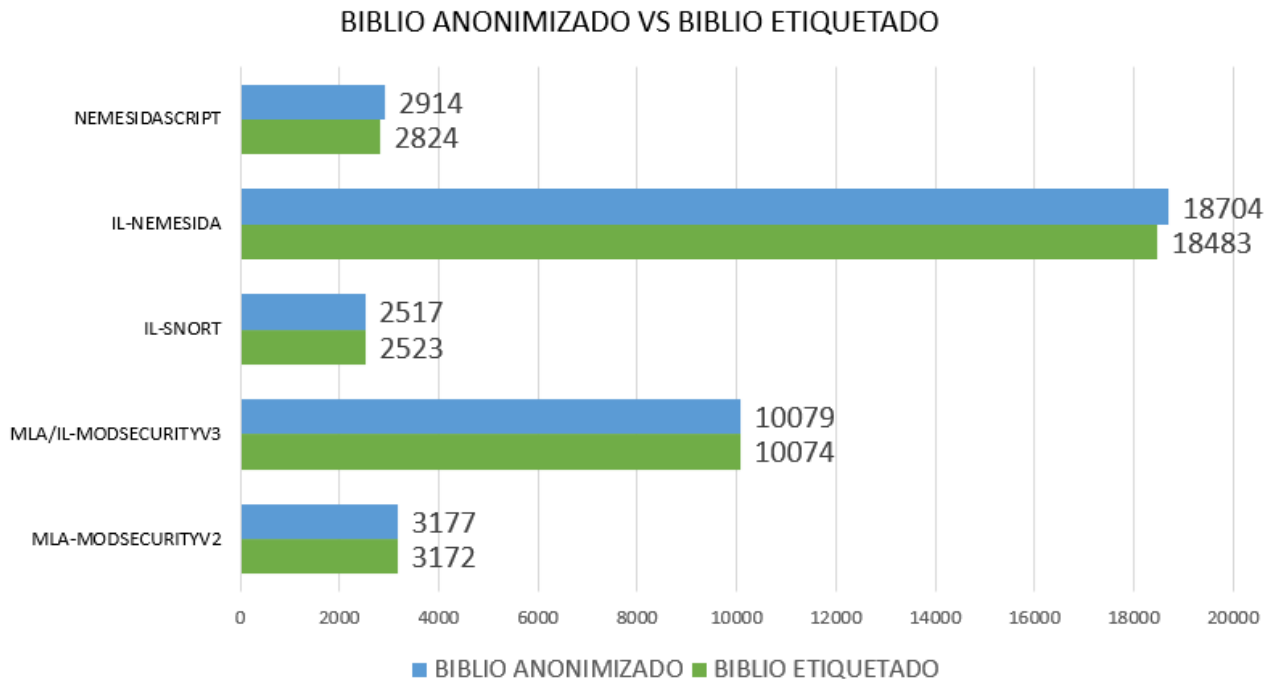
Este estudio puede verse con más detalle con las uris de diferencia para cada IDS en [22].

5.3.3 Estudio de impacto en el etiquetado de BIBLIO

En este apartado estudiaremos el impacto de anonimizar las uris del 'Dataset BIBLIO ETIQUETADO'. Es decir, estableceremos una comparación entre las detecciones generadas para 'BIBLIO ETIQUETADO' Vs 'BIBLIO ANONIMIZADO'.

Nota: con objeto de simplificar el estudio tomaremos solo las detecciones **en PL2** para aquellos IDS basados en ModSecurity.

El resultado de establecer dicha comparativa queda reflejado en la siguiente gráfica:



Aquí podemos observar unas leves diferencias en el número de detecciones generados por cada IDS para cada uno de los Datasets.

Observando estos datos podemos establecer que, **salvo para el IDS Snort, la anonimización de los datos se ha traducido en un leve aumento de las detecciones.** Snort es el único IDS que ve disminuido en 6 el número de detecciones tras la anonimización del Dataset.

A continuación, se ofrece una tabla comparativa BIBLIO ANONIMIZADO Vs BIBLIO ETIQUETADO con las uris que difieren entre sí.

Dataset	MLA-ModSecurityV2	MLA/IL-ModSecurityV3	IL-Snort	IL-Nemesida	NemesidaScript
BIBLIO ANONIMIZADO	6	6	0	410	232
BIBLIO ETIQUETADO	1	1	6	189	142

Tabla 29 Comparativa de uris únicas presentes en cada uno de los Dataset

Un estudio más detallado que muestra las uris de diferencia entre ambos 'Datasets' puede consultarse en [23].

5.4 Análisis temporales

Ahora que hemos visto los resultados obtenidos en clave de detecciones, pasaremos a aportar información sobre el segundo de los dos grandes objetivos a cumplir, la reducción de los tiempos de ejecución.

Nota: todas las detecciones a partir de este momento han tenido lugar con PL=4, siendo este nivel de Paranoia el más desfavorable en cuanto a tiempo se refiere.

La modalidad de lanzamiento utilizada ha sido la denominada como 'múltiple'. Que recordemos, es aquella basada en lanzamiento de todo el fichero de uri de entrada contra el IDS, para que sobre este lanzamiento conjunto tenga lugar el procesamiento de los 'logs' generados.

Fichero utilizado para las pruebas:

Fichero	Dataset	Número de uris
access_log-20170208.ltxt.uris	Biblio_Anonimizado	301608

Tabla 30 Fichero de pruebas temporales

Los tiempos obtenidos en las pruebas realizadas quedan recogidos en la tabla siguiente:

IDS	Equipos			Instancias			Tiempo	Tipo de detector
	1	5	20	1	5	10		
MLA-ModSecurityV2	X			X			2h 32' 44"	Online
	X				X		20' 3"	
	X					X	16' 6"	
		X		X			32' 2"	
		X			X		3' 59"	
		X				X	3' 6"	
			X	X			8' 16"	
MLA-ModSecurityV3	X			X			6' 20"	Offline
	X				X		2' 54"	
	X					X	2' 53"	
		X		X			1'32"	
		X			X		37"	
		X				X	27"	
			X	X			25"	
IL-ModSecurityV3	X			X			5' 36"	Offline
	X				X		2' 48"	
	X					X	2' 47"	
		X		X			1'24"	
		X			X		37"	
		X				X	25"	
			X	X			25"	
IL-Snort	X			X			1' 52"	Offline
	X				X		1' 20"	
	X					X	57"	
		X		X			38"	
		X			X		16"	
		X				X	6"	
			X	X			11"	
IL-Nemesida	X			X			1'	Offline
	X				X		29"	
	X					X	19"	
		X		X			21"	
		X			X		4"	

		X				X	5”	
			X	X			8”	
NemesidaScript	X			X			2h 19’ 7”	Online
	X				X		17’ 59”	
	X					X	14' 38"	
		X		X			29’ 26”	
		X			X		3’ 38”	
		X				X	2’ 45”	
			X	X			7’ 3”	

Tabla 31 Comparativa de tiempos Equipos-instancias

5.4.1 Estudio de tiempos según el tipo de detector

En la tabla anterior queda patente que los tiempos invertidos en las pruebas para los detectores que gozan de una implementación ‘Online’ son significativamente superiores a los tiempos registrados para los IDS de tipo ‘Offline’. Para ilustrar este hecho, representaremos los tiempos invertidos por el IDS ‘Online’ ‘MLA-ModSecurityV2’ en cada una de las pruebas Vs el IDS ‘Offline’ ‘MLA-ModSecurityV3’.

Se han escogido estos IDS porque ambos están basados en ModSecurity y su principal diferencia radica precisamente en que uno es de tipo ‘Online’ y otro de tipo ‘Offline’.

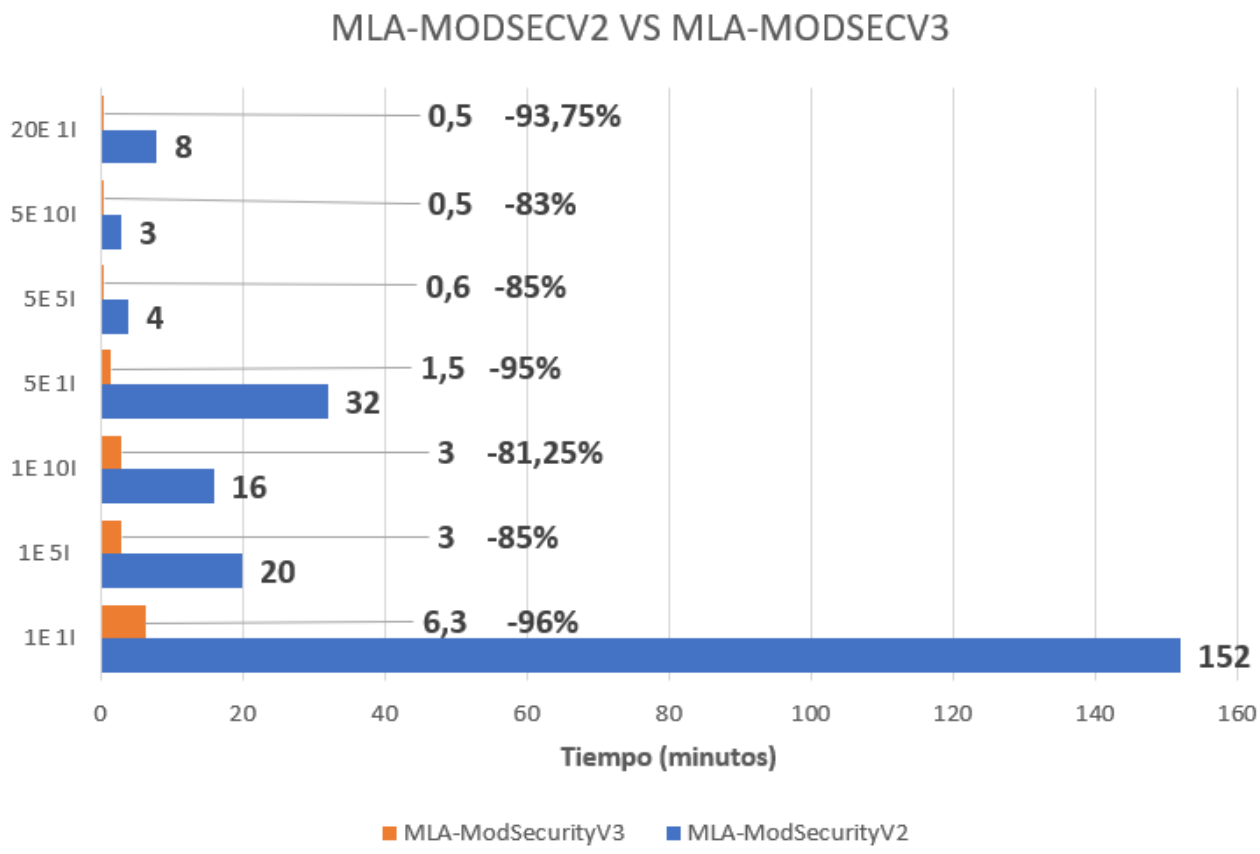


Tabla 32 Tiempos MLA-MODSECV2 VS MLA-MODSECV3

Puede verse como, para todas las pruebas realizadas, MLA-ModSecurityV3 (el cual está basado en un IDS ‘Offline’) es significativamente más rápido que MLA-ModSecurityV2 (el cual está basado en un IDS ‘Online’).

Este ejemplo va en la línea del resto de IDS estudiados y puede observarse como, **en media, los detectores basados en una metodología 'Offline' ven reducidos sus tiempos aproximadamente un 88% frente a los basados en una metodología 'Online' según las pruebas realizadas.**

Nota: en la gráfica anterior, el eje vertical representa el número de instancias y equipos presentes en la prueba. De forma que '1E 1I' representaría '1 Equipo 1 Instancia'.

5.4.2 Estudio de tiempos en función de la prueba y tipo del detector

En este apartado estudiaremos un fenómeno que varia en función de la prueba realizada y del tipo de detector sobre el que se realicen dichas pruebas.

Tomando los mismos representantes que en el anterior apartado (MLA-ModSecurityV3 Vs MLA-ModSecurityV2) observamos lo siguiente:

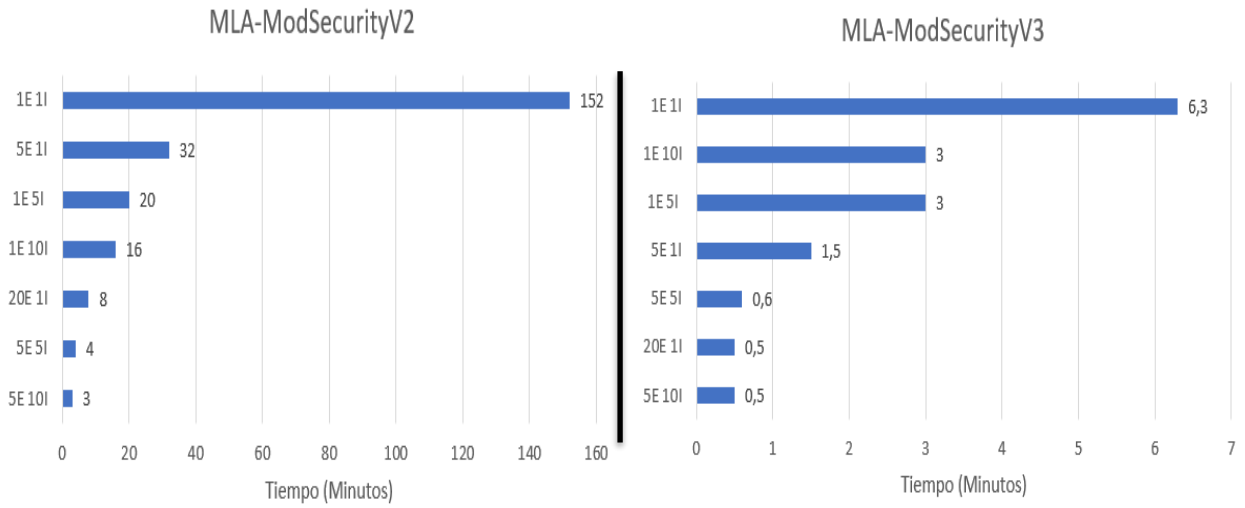


Ilustración 32 Comparativa temporal entre modos IDS Online Vs Offline

En esta imagen puede observarse un fenómeno que está presente en todos los IDS estudiados y es que, para los detectores de tipo 'Online', los tiempos disminuyen cuando aumentamos el número de instancias frente al número de equipos, es decir; si observamos la ilustración anterior podemos comprobar como el tiempo invertido en realizar la prueba para 5 equipos con una instancia por equipo, es superior al invertido cuando ejecutamos 5 instancias del IDS en un solo equipo.

Justo lo opuesto ocurre en los IDS de tipo 'Offline'. Puede observarse como claramente, el tiempo invertido en el análisis del Dataset es inferior cuando se emplean 5 equipos con una sola instancia por equipo, frente al uso de 1 equipo con 5 instancias del IDS funcionando paralelamente.

Como conclusión a lo expuesto anteriormente podría decirse que, aunque la combinación de 'n' equipos con 'm' instancias resulta beneficiosa en término temporales para ambos casos, la mejora al introducir un mayor número de equipos es más acusada en los detectores de tipo 'Offline', mientras que en los de tipo 'Online' la mejora será más acusada al introducir un mayor número de instancias por equipo.

Nota: en la gráfica anterior, el eje vertical representa el número de instancias y equipos presentes en la prueba. De forma que '1E 1I' representaría '1 Equipo 1 Instancia'.

5.4.3 Mejoras obtenidas por el uso conjunto de multiples equipos con múltiples instancias

En este apartado estudiaremos el efecto sobre los tiempos de introducir multiples equipos con multiples

instancias (5 equipos con 5 instancias del IDS por equipo), frente al uso de 1 solo equipo ejecutando una sola instancia del IDS.

En la siguiente gráfica puede apreciarse en términos porcentuales la disminución de tiempos de aplicar la solución de múltiples equipos con múltiples instancias frente a su no uso para cada IDS:

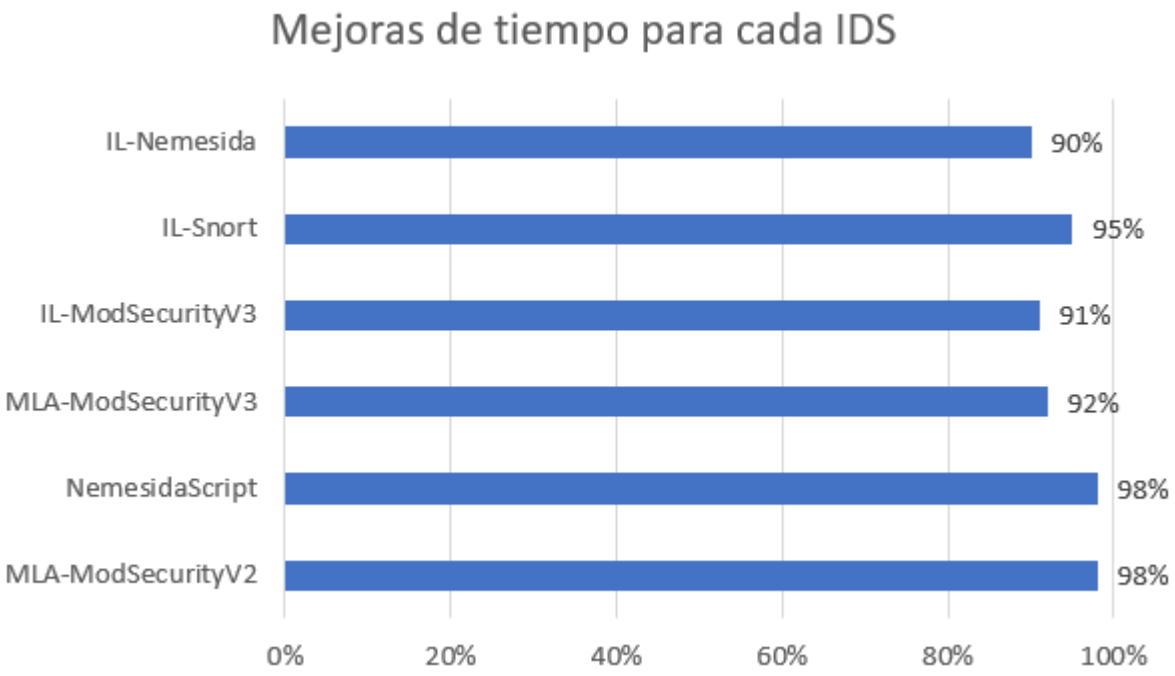


Ilustración 33 Mejoras de tiempos al aplicar la solución '5 equipos con 5 instancias'

De lo que puede deducirse que todos los IDS experimentan una mejora muy significativa de los tiempos al repartir los Datasets entre múltiples equipos corriendo múltiples instancias, siendo levemente superior la mejora en los IDS de tipo 'Online'.

5.4.4 Compartiva entre modos de lanzamiento

En este apartado veremos una pequeña prueba utilizando ahora el fichero de entrada:

Fichero	Dataset	Número de uris
0days.uri	De_ataques	420

Tabla 33 Fichero entrada prueba '1to1' vs 'multiple'

Con el que trataremos de ver las diferencias temporales existentes entre el tipo de lanzamiento '1to1' y el lanzamiento 'multiple'. La prueba se ha realizado **en un solo equipo**:

Detector	Tiempo		Tipo de detector
	1to1	Multiple	
MLA-ModSecurityV2	51”	10”	online
MLA-ModSecurityV3	1’ 3”	1”	online
IL-ModSecurityV3	51”	1”	offline
IL-Snort	36”	2”	offline
IL-Nemesida	32”	1”	offline
NemesidaScript	52”	8”	online

Tabla 34 Tiempos lanzamiento '1to1' vs ‘multiple’

En la que se puede comprobar fácilmente como el tipo de lanzamiento ‘multiple’ es mucho más rápido que el ‘1to1’. El modo ‘1to1’ solo es aconsejable en aquellos casos en los que se requiera un control totalmente preciso e inequívoco para cada uri en cada una de las etapas del proceso de detección diseñado en este trabajo.

5.4.5 Comparativa temporal de detecciones con las aplicaciones desarrolladas

Por último, estableceremos una pequeña comparativa temporal de ciertos ‘Datasets’ analizados con las aplicaciones desarrolladas en este trabajo Vs el análisis con las herramientas con las que partíamos al iniciar este proyecto.

Detector	Dataset	PL	Equipos	Instancias	Tiempos
MLA-ModSecurityV2 (antigua versión)	Biblio filtrado	4	~20	-	~25 días
MLA-ModSecurityV2 (nueva versión)	Biblio anonimizado	4	25	10	2h 6’ 52”
MLA-ModSecurityV2 (antigua versión)	INVES	4	~20	-	~26 días
MLA-ModSecurityV2 (nueva versión)	INVES	4	25	8	36’ 25”

Tabla 35 Comparativa temporal versión antigua vs nueva

Nota: esta prueba es una aproximación, ya que para la obtención de tiempos de las versiones antiguas no se usó el comando ‘time’ y se desconoce con precisión el número de equipos implicados, aunque se estima en unos 20.

La medición de tiempos se ha realizado de manera que el análisis no se ha dado por concluido hasta la finalización del último de los equipos.

Nota2: las reglas empleadas son las mismas que las usadas en todo el apartado (owasp crs 3.2.0 modificadas).

De esta forma queda patente una mejora significativa en los tiempos de ejecución al usar las aplicaciones desarrolladas en el presente trabajo.

6 CONCLUSIONES Y LINEAS DE FUTURO

6.1 Conclusiones

En este apartado detallaremos las conclusiones extraídas sobre el trabajo y los resultados obtenidos.

a. Durante el desarrollo del trabajo hemos determinado que 'Checkpoint' presenta anomalías en las detecciones que deben ser estudiadas antes de poder considerar sus detecciones como válidas.

b. En base a la experiencia obtenida, se desaconseja que el analizador del log a implementar en un futuro por un tercero que quiera hacer uso de las herramientas desarrolladas, esté codificado en 'shell-script'. Existen otros lenguajes como 'C' o 'Python' que han demostrado ser mucho más eficaces en el procesado de textos con cierta complejidad, como es el caso de los logs generados por ciertos detectores.

c. Tras el estudio sobre las alteraciones sufridas por las uris que contienen a ciertos caracteres especiales se ha concluido que: la mínima codificación necesaria para que estas puedan viajar hasta el servidor y lleguen sin ver alterada su integridad, es mediante el uso del comando 'curl' y con la codificación realizada en la 'Tabla 4'. En la que básicamente se establece la necesidad de dotar del carácter de escape '\' a la apertura y cierre tanto de llaves '{}' como de corchetes '[]', así como la codificación de los caracteres '#' y ' ' (caracter de espacio).

d. Estudiando las diferencias en las detecciones entre MLA-ModSecuriyV3 y MLA-ModSecurityV2, hemos podido concluir que generalmente la codificación de ciertos caracteres prohibidos en las uris para hacer que estas cumplan el protocolo 'http', y puedan viajar por la red, suele provocar una disminución en las detecciones. También se ha descubierto que, aunque mínimas, la pre-codificación de caracteres puede generar en ocasiones algunas detecciones adicionales.

   Con esto se concluye que los detectores de tipo 'Offline' son más fiables y ofrecen una mayor eficiencia en las detecciones que los de tipo 'Online'. Las diferencias presentes en las uris por este hecho pueden consultarse en [21].

e. En el estudio del impacto del etiquetado de 'BIBLIO' se ha comprobado que, aunque mínimos, la anonimización de las uris provoca cambios en las detecciones.

   Estos cambios pueden ser consultados en [23], y hemos podido comprobar que, salvo para Snort, la anonimización suele traducirse en un aumento neto del número de detecciones.

f. En lo que respecta al estudio de las detecciones al solapar IDS [22], hemos comprobado que es más eficiente seleccionar la acción conjunta de ciertos IDS a la hora de obtener un aumento en las detecciones, que usar indiscriminadamente todos los IDS de los que se dispongan.

   También hemos observado una mejora en la eficiencia más significativa al solapar IDS en 'Datasets de ataques' que en los 'Datasets reales'. Esto puede deberse a que básicamente las detecciones en este tipo de 'Datasets' son mínimas, y la combinación de las detecciones presentes entre varios IDS apenas suponen un aumento de la eficiencia con respecto al total de uris.

g. Hemos conseguido reducir los tiempos de manera muy significativa, pasando en algunos casos, de semanas de análisis a unas pocas horas.

h. Ha quedado patente que los detectores de tipo 'Offline' son aproximadamente un 90% más rápidos que los de tipo 'Online'.

i. Se ha podido constatar una mejora muy significativa en los tiempos al introducir en las detecciones múltiples equipos ejecutando múltiples instancias concurrentes de un mismo IDS, independientemente de la naturaleza de este.

   También en el estudio realizado hemos podido comprobar que los detectores de tipo 'Offline' experimentan una mayor mejora en los tiempos al aumentar el número de equipos, y una mejora más reducida al aumentar el número de instancias por equipo. Justo lo contrario se observa en los de tipo 'Online'. Por último, se ha comprobado que la mejora en los tiempos al introducir estas técnicas es levemente superior en los detectores de tipo 'Online' en términos relativos, aunque significativamente superior en términos absolutos.

j. Se ha cumplido con los propósitos propuestos para nuestra herramienta de:
   I. Flexibilidad: permite la adición sencilla de nuevos detectores, tanto 'offline' como 'online'.
   II. Simplificadora: al tener una estructura común a todos los IDS, la adición de cada nuevo detector solo implica la adición de una nueva 'función' de análisis de logs.
   III. Centralizada: al disponer de una configuración global para todos los detectores.
   IV. Distribuida, Multitarea y Automatizada: al ser capaz de lanzar, de forma automática, el análisis en múltiples equipos, y varias tareas por equipo, permitiendo fácilmente monitorizar el estado de cada equipo/tarea y la recogida de resultados.
   V. Sencilla: interfaz y configuración fáciles.

k. Los datos obtenidos para todos los 'Datasets' con todos los niveles de paranoia, coinciden a la perfección para los detectores 'MLA-ModSecurityV3' e 'IL-ModSecurityV3'. Esto era algo esperable y deseable puesto que ambos están basados en la misma versión de un mismo módulo de detección, y realizan la misma carga de reglas. No obstante, han sido desarrollados de manera independiente y siguiendo diferentes metodologías. Habla positivamente sobre la idoneidad de los datos obtenidos que, tras implementarse en los softwares desarrollados sus resultados coincidan.

6.2 Líneas de futuro

En este apartado se darán algunas posibles líneas de continuación del trabajo desarrollado, así como algunas propuestas de mejora:

- Volcado cómodo de los datos obtenidos en alguna plataforma (como una web), que permita la consulta de los resultados para quien pueda resultarle de interés de una manera rápida y sencilla.
- Mejoras generales de calidad de vida con funciones adicionales y menús más visualmente atractivos.
- Incorporación de nuevas funcionalidades que puedan surgir en un futuro.
- Mejoras que puedan acelerar aún más los tiempos de despliegue y ejecución.
- Encontrar las causas que implicaban un comportamiento erróneo por parte de Chekpoint e incorporarlo como detector.

ANEXO A: ESTRUCTURA DE DIRECTORIOS Y SCRIPTS DE ACD

En este anexo describiremos la estructura de directorios de la que se compone ACD y detallaremos la función de cada uno de los scripts que la componen.

La estructura de directorios es la siguiente:

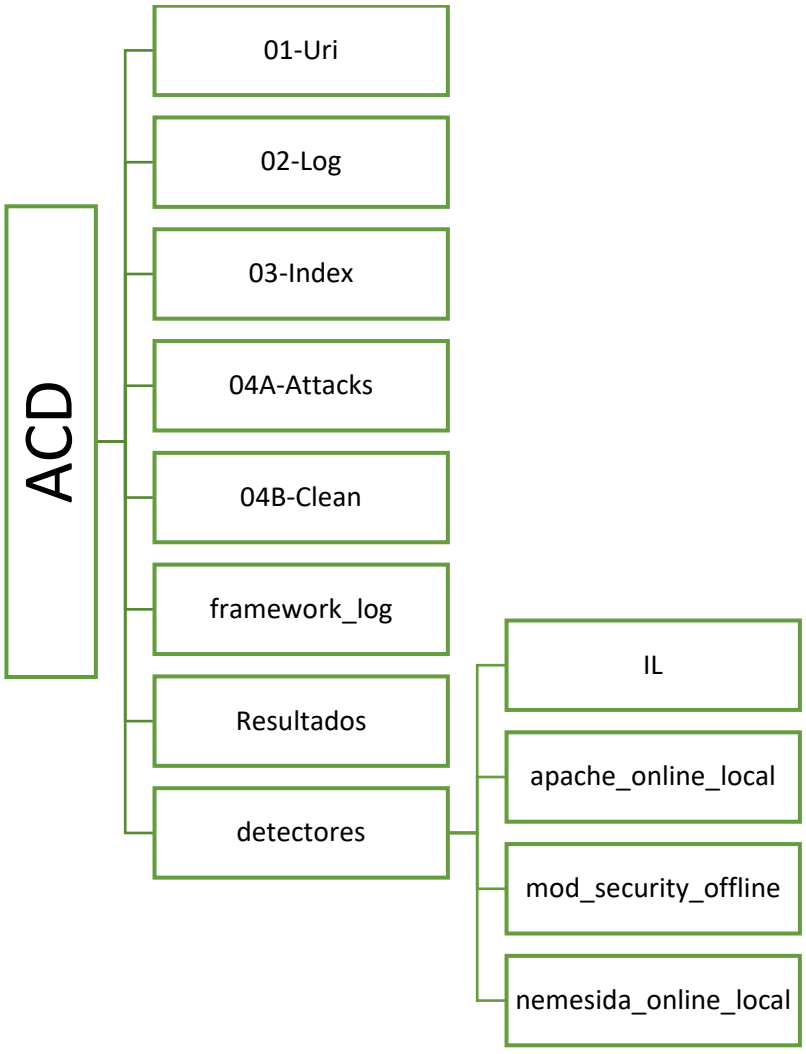


Ilustración 34 Estructura de directorios de ACD

Directorios

- **01-Uri:** directorio donde deberán situarse los ficheros de uris de entrada a analizar.

- **02-Log:** directorio donde se almacenarán los *logs* generados por el detector.
- **03-Index:** directorio donde se almacenarán los ficheros *index* generados por el analizador.
- **04A-Attacks:** directorio donde se almacenarán los ficheros de *attacks* y de *attacks extendido*.
- **04B-Clean:** directorio donde se almacenarán los ficheros con las uris detectadas como *clean.*
- **Framework_log:** directorio donde se almacenará el fichero *log* de ACD.
- **Resultados:** directorio donde se almacenan los resultados una vez finalizados el análisis.
- **Detectores:** directorio donde se almacena todo lo necesario de los detectores implementados (IL, apache_online_local, mod_security_offline y nemesida_online_local).

Scripts

- **0-Framework.sh:** script de análisis de los ficheros de uris presentes en el directorio *'01-Uri/'*. Se sirve del resto de scripts disponibles en el directorio para realizar dicho análisis.
- **1-Launcher.sh**: script encargado de realizar los lanzamientos de las uris contra los elementos de detección. En la modalidad *online* también se encarga de la codificación de los posibles caracteres inválidos existentes en las uris, para que estas cumplan el protocolo http y puedan ser lanzadas.
- **2-analyzer.py:** encargado de analizar los logs generados por ModSecurity (Apache).
- **2-analyzer_nemesida_onine.py:** encargado de analizar los logs generados por Nemesida (nwaf + nginx).
- **3-classify.py:** encargado de realizar la clasificación de las uris en *clean* y attacks.
- **3-classify_IL.py:** encargado de realizar la clasificación de las uris en *clean* y de *attacks* a partir del fichero obtenido tras la ejecución de IL en cualquiera de sus modalidades.
- **IL.sh:** encargado de realizar la invocación adecuada según la modalidad y detector de IL seleccionado en la configuración.
- **MLAv3_launcher.c:** API de lanzamiento de las uris de ModSecurityV3.

- **anade_barra.sh:** comprueba -y añade si no está presente- el carácter '/' al inicio de cada uri presente en el fichero de entrada.

- **configura_instancia.sh:** realiza las operaciones necesarias para poder tener múltiples instancias de un servidor funcionando simultáneamente.

- **detener_servidor_instancia.sh:** detiene el servidor instanciado una vez concluido el análisis.

- **install.sh:** script de instalación de *ACD* y sus detectores.

- **monitoriza-local.sh:** en modalidad *online-local* monitoriza la generación del fichero de log, de forma que cuando este está disponible, lanza un aviso para continuar con la ejecución del programa.

- **monitoriza-remoto.sh:** mismo propósito que *monitoriza-local.sh* pero en modalidad *online-remoto*.

- **rebuild_output.sh:** reconstruye la salida de forma trasparente al usuario cuando se hace uso del script *remove_repeats.sh*.

- **remoto.sh:** usado en modo *online-remoto* con tipo de lanzamiento *1to1*. Detecta la llegada del log del equipo remoto y continúa la ejecución del programa.

- **remoto-multiple.sh:** igual que *remoto.sh* pero para tipo de lanzamiento *multiple.*

- **remove_repeats.sh:** elimina las uris repetidas del fichero de entrada.

- **sendlogs.sh:** en modalidad *online-local* envía el log del equipo cliente al equipo servidor.

- **unicode.mapping:** necesario para API de ModSecurityV3.

zlimpiar.sh: script para 'limpiar' los resultados de los directorios de resultados.

ANEXO B: INSTALACIÓN Y USO DE ACD

Nota: este proyecto se encuentra disponible en [1].

Instalación

En este apartado detallaremos cómo llevar a cabo la instalación de los IDS (no obstante, en la carpeta raíz del proyecto [1] se encuentra el script de instalación que automatiza este proceso denominado 'install.sh'). También aprovecharemos para arrojar más luz sobre el procedimiento 'multi-instancia' en aquellos casos en los que se requiere la presencia de un servidor.

Todas las instalaciones han tenido lugar en CentOS 7 haciendo uso del usuario 'root'.

Instalación de MLA/IL-ModSecurityV3:

1. Para la instalación de MLA/IL-ModSecutiryV3 deberemos ejecutar los siguientes comandos:

```
yum install gcc-c++ flex bison yajl yajl-devel curl-devel curl GeoIP-devel doxygen zlib-devel pcre-devel
cd /opt/
git clone https://github.com/SpiderLabs/ModSecurity
cd ModSecurity
git checkout -b v3/master origin/v3/master
sh build.sh
git submodule init
git submodule update
./configure
yum install https://archives.fedoraproject.org/pub/archive/fedora/linux/updates/23/x86_64/b/bison-3.0.4-3.fc23.x86_64.rpm
make
make install
```

2. [MLA-ModSecurityV3] en este paso deberemos de crear el fichero 'debug.log' en el directorio correspondiente de la instalación puesto que si no lo hacemos 'MLA-ModSecurityV3' puede presentar errores. En nuestro caso:

```
mkdir -p /opt/modsecurity-debug/var/log
touch /opt/modsecurity-debug/var/log/debug.log
```

3. A continuación tendremos que mover las librerías al directorio de carga de librerías de dichos detectores, y establecer la ruta de carga de las mismas. En nuestro caso:

```
mv /usr/local/modsecurity/lib/* /usr/lib64/
LD_LIBRARY_PATH="/usr/lib64"
export LD_LIBRARY_PATH
```

Instalación de Nemesida WAF API

Hemos ejecutado los siguientes comandos:

```
rpm -Uvh https://nemesida-security.com/repo/nw/centos/nwaf-release-centos-7-1-6.noarch.rpm
yum install -y epel-release
yum install -y rabbitmq-server
rm -f /etc/machine-id
/bin/systemd-machine-id-setup
setenforce 0
rpm -Uvh http://nginx.org/packages/centos/7/x86_64/RPMS/nginx-1.18.0-1.el7.ngx.x86_64.rpm
yum install -y python36 python36-devel python36-setuptools python36-pip openssl librabbitmq libcurl-devel rabbitmq-server gcc libmaxminddb memcached
yum install -y nwaf-dyn-1.18
```

En nuestro caso hemos instalado la versión nwaf1.18.

Instalación de IL-Snort e IL-Nemesida

Para poder hacer funcionar IL-Snort e IL-Nemesida deberemos crear un enlace simbólico de la librería que originalmente se encontraba en Debian a CentOS 7. Además, tal y como ocurría en IL-ModSecurity, es necesario indicar la ruta de carga de las librerías requeridas para su funcionamiento:

```
yum install -y pcre-devel
ln -s /usr/lib64/libpcre.so.1.2.0 /usr/lib64/libpcre.so.3
LD_LIBRARY_PATH="/usr/lib64"
export LD_LIBRARY_PATH
```

Multi-instancia online

Aprovecharemos este apartado para arrojar más luz sobre cómo se ha podido llevar a cabo la posibilidad de ejecutar múltiples instancias de un servidor en un mismo equipo.

Apache

Para poder correr múltiples instancias de apache en un mismo equipo sin que estos entren en conflicto, y siguiendo la estructura de detectores utilizada en este proyecto, es necesario lo siguiente:

- Establecer nuevo *serverRoot* en fichero de configuración de apache 'httpd.conf'.
- Establecer nuevo puerto de escucha en fichero de configuración de apache 'httpd.conf'.
- Establecer nuevo puerto de escucha en fichero configuración 'ssl.conf'.

Cada una de las instancias creadas se arrancará mediante el comando:

```
httpd -f  "path_httpd.conf" -k start
```

Es importante cerciorarse de la existencia del subdirectorio 'run/' para que apache pueda escribir su 'PID' durante el arranque.

Nemesida

Para poder correr múltiples instancias de *nginx + nwaf* en un mismo equipo sin que estos entren en conflicto, y siguiendo la estructura de detectores utilizada en este proyecto, es necesario lo siguiente:

- Adaptar las rutas a la instancia en el fichero de configuración de nginx 'nginx.conf'.
- Adaptar puerto de escucha en fichero de configuración 'default.conf'.

Cada una de las instancias creadas se arrancará mediante el comando:

```
nginx  -c "path_nginx.conf"
```

Todos estos cambios se realizan de forma automática en tiempo de ejecución mediante el script *'configura_instancia.sh'*.

Nota: es importante seguir estas instrucciones siempre y cuando se emplee la estructura de directorios propuesta en este proyecto, ya que esta está pensada para minimizar los cambios necesarios en la inserción de un nuevo servidor/detector tanto *online* como *offline*.

Configuración

En este apartado ahondaremos sobre la configuración de ACD y los detectores implementados.

ACD

ACD cuenta con dos ficheros de configuración:

- **framework_config.conf:** este primer fichero de configuración tiene como propósito ser lo más escueto posible con el fin de que el usuario cuente con un número limitado de opciones para que de una forma sencilla y rápida pueda realizar la ejecución de la herramienta.

  Las opciones más destacables configurables desde este fichero son:
  - Modo de lanzamiento.
  - Activar/Desactivar eliminación de repeticiones.
  - Formato de entrada de las uris.
  - Ejecutable de API para detectores *offline*.
  - Variables de configuración relacionadas con lanzamiento en equipo remoto.

- **framework_config_interna.conf:** este fichero es mucho más completo que el anterior y permite realizar funciones más avanzadas.

  Las opciones más destacables a configurar en este fichero son:
  - Rutas de ACD y sus detectores.
  - Detector a usar de entre todos los disponibles.
  - Tipo de lanzamiento.
  - Extensiones de los ficheros.

- Puerto de escucha por defecto.

Detectores

En este apartado explicaremos cómo modificar el nivel de paranoia en ModSecurity y algunos aspectos de la configuración de los detectores implementados.

Nota: El nivel de paranoia (PL, *Paranoia Level* en inglés) en ModSecurity es un parámetro que en función de su valor realiza una carga de reglas más o menos restrictiva; esto es, a medida que aumentamos el nivel (1-4), el conjunto de reglas que pueden interceder en el proceso de detección aumenta, y por tanto, la detección se torna más estricta (y lenta al tener que procesar un mayor número de reglas).

Para modificar el nivel de paranoia (PL) de los detectores que implementan ModSecurity, deberemos editar el fichero de configuración de dicho detector. La ruta varía en función del IDS que queramos modificar:

- **apache_online_local:** detectores/apache_online_local/owasp-modsecurity-crs-3.2.0/crs-setup.conf
- **mod_security_offline:** detectores/mod_security_offline/crs-setup.conf
- **IL-modsecurity:** detectores/IL/etc/crs-setup.conf

Y deberemos modificar el campo: 'setvar:tx.paranoia_level=1' por el valor de paranoia deseado.

El acceso a los distintos ficheros de configuración de los detectores en los que podemos modificar aspectos como la ruta de los logs, cargas de reglas etc., es el siguiente:

- **apache_online_local:** detectores/apache_online_local/conf.d/mod_security.conf
- **mod_security_offline:** detectores/mod_security_offline/basic_rules.conf
- **IL-modsecurity:** detectores/IL/etc/basic_rules.conf
- **nemesida_online:** detectores/nemesida_online_local/nwaf/conf/global/nwaf.conf

IL-Nemesida e IL-Snort realizan la carga de reglas y modifican su configuración mediante argumentos en su invocación. En nuestro caso estamos ejecutando los siguientes comandos:

- IL-Snort: 'inspectorlog -l fichero.uri -t list -r snort_rules'

  Siendo 'snort_rules/' el directorio que contiene las reglas de *Snort*.

- IL-Nemesida: 'inspectorlog -l fichero.uri -t list -m nemesida-rules-bin-20220109.txt'

  Siendo 'nemesida-rules-bin-20220109.txt' el fichero de reglas de *Nemesida.*

Ejemplo de uso

En este apartado haremos un análisis a modo de ejemplo con el objetivo de poder servir de guía y/o manual para aquel usuario que quiera hacer uso de la herramienta.

Analizaremos el fichero ***0days.uri*** (420 uris) haciendo uso del detector ***mod_security_offline*** (ModSecurityV3) con PL=1 y modo de lanzamiento ***múltiple***.

AVISO: hemos seleccionado este detector para poder notificar un error detectado durante el desarrollo de las pruebas, y es que, ModSecurityV3 se encuentra en una fase alfa de desarrollo, y sus propios

creadores han reportado un *bug* de 'fuga de memoria' que se da durante el lanzamiento de las uris y carga de las reglas.

Para poder evitar desbordamientos de memoria haciendo uso de este detector, y por si puede servirle de ayuda al usuario que quiera hacer uso del IDS, hemos establecido una relación de aproximadamente 5 GB de uso de memoria RAM por cada 100k uris a analizar (este valor puede crecer un poco más cuanto mayor sea el PL) por lo que se recomienda al usuario tener en cuenta esta relación a la hora de hacer análisis. En concreto a la hora de realizar las pruebas de este proyecto hemos tratado de fragmentar algunos de los ficheros de entrada de mayor extensión, con el objetivo de no sobrepasar la memoria disponible.

La prueba se desarrollará en el equipo LT04 haciendo uso de CentOS 7.

Dicho esto, comenzamos con el análisis, los pasos a seguir son los siguientes:

1. Realizar instalación de ACD, para ello deberemos ejecutar el script **install.sh**.
2. Una vez instalado, desplazamos el fichero de entrada, que en nuestro caso es el fichero ***0days.uri*** al directorio **01-Uri/.**
3. Ahora estableceremos la configuración para realizar el análisis en las condiciones expuestas. Para ello:

   3.1. Fijamos nivel de paranoia en 1 (establecer el nivel de paranoia solo es necesario en aquellos detectores que implementen ModSecurity). Para ello, y tal y como vimos anteriormente, deberemos editar el fichero **'detectores/mod_security_offline/crs-setup.conf'** tal que así:

```
SecAction \
 "id:900000,\
  phase:1,\
  nolog,\
  pass,\
  t:none,\
  setvar:tx.paranoia_level=1"
```

   3.2. En fichero de configuración ***framework_config.conf:***

   3.2.1.**LAUNCH_MODE="multiple"**. Establece el modo de lanzamiento múltiple.

   3.2.2. **URIS_FORMAT="basic"**. Establece el formato del fichero de entrada como *basic,* esto es, las uris no presentan identificador.

   3.2.3. **API_SCRIPT="MLAv3_launcher.out"**. API de ModSecurityV3, usada en el lanzamiento de las uris.

   3.3. En fichero de configuración ***framework_config_interna.conf:***

   3.3.1. **MODSECURITY_OFFLINE=1.** Establece el detector a utilizar, el resto de detectores deben tener su valor fijado en **0**.

3.3.2. En el apartado *'extensiones'* puede modificar las extensiones de los ficheros de entrada y salida de la herramienta. En el contexto de nuestra prueba las extensiones utilizadas son las siguientes:

```
FILE_IN_EXTENSION=".uri"
LOG_EXTENSION=".log"
INDEX_EXTENSION=".index"
INFO_ATTACKS_EXTENSION="-info.attacks"
ATTACKS_EXTENSION=".attacks"
CLEAN_EXTENSION=".clean"
```

3.3.3. Si hubiésemos hecho uso de un detector *'online'* podríamos establecer el puerto por defecto de escucha del servidor mediante la variable: **DEFAULT_PORT.**

4. Procedemos al lanzamiento y análisis del fichero de entrada. Para ello, estando situado en el directorio raíz de ACD ejecutamos el script:

```
0-Framework.sh
```

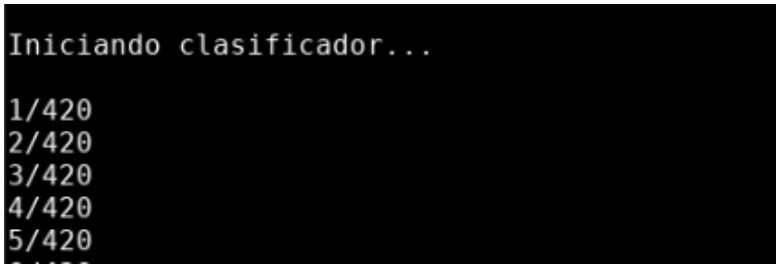
5. Por pantalla se nos mostrará el fichero de entrada, el modo de lanzamiento seleccionado y el progreso de cada una de las fases:

```
/opt/framework/01-Uri/0days.uri

Ejecutando modo de lanzamiento múltiple
```

.
.
.

```
Iniciando análisis...

1/137
2/137
3/137
4/137
5/137
```

.
.
.



6. Una vez concluido el análisis obtenemos los resultados de forma ordenada en cada uno de los directorios. Es importante mencionar en este punto que los nombres de dichos ficheros de resultados guardan relación con el nombre del fichero de entrada para que de esta forma puedan relacionarse. Lo entenderemos mejor en el punto siguiente, en el que analizaremos los resultados obtenidos.

7. Análisis de resultados:

   - **02-Log/0days.log:** presentaría un formato como el siguiente:

---siJN2HMt---A--

[07/Jul/2022:11:14:04 +0200] 1657185244 127.0.0.1 12345 127.0.0.1 80

---siJN2HMt---B--

GET /appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-O%20/tmp/dsifnfnfdwsFFS) HTTP/1.1

Host: localhost

User-Agent: Apache/2.2.15 (Red Hat) (internal dummy connection)

Accept: Yes

---siJN2HMt---H--

ModSecurity: Warning. Matched "Operator `Rx' with parameter `(?:;|\{|\|\|\|\||&|&&|\n|\r|\$\(|\$\(\(|`|\${|<\(|>\(|\(\s*\))\s*(?:{|\s*\(\s*|\w+=(?:[^\s]*|\$.*|\$.*|<.*|>.*|\'.*\'|\".*\")\s+|!\s*|\$)*\s*(?:'|\")*(?:[\?\*\[\]\(\)\-\|+\w'\"\./\\\\]+/)?[\\\\'\"]*(?: (5275 characters omitted)' against variable `ARGS:hostName' (Value: `$(/usr/bin/wget http://1.1.1.1:12345/file -O /tmp/dsifnfnfdwsFFS)') [file "detectores/mod_security_offline/rules/REQUEST-932-APPLICATION-ATTACK-RCE.conf"] [line "140"] [id "932105"] [rev ""] [msg "Remote Command Execution: Unix Command Injection"] [data "Matched Data: $(/usr/bin/wget found within ARGS:hostName: $(/usr/bin/wget http://1.1.1.1:12345/file -O /tmp/dsifnfnfdwsFFS)"] [severity "2"] [ver "OWASP_CRS/3.2.0"] [maturity "0"] [accuracy "0"] [tag "application-multi"] [tag "language-shell"] [tag "platform-unix"] [tag "attack-rce"] [tag "paranoia-level/1"] [tag "OWASP_CRS"] [tag "OWASP_CRS/WEB_ATTACK/COMMAND_INJECTION"] [tag "WASCTC/WASC-31"] [tag "OWASP_TOP_10/A1"] [tag "PCI/6.5.2"] [hostname "127.0.0.1"] [uri "/appliancews/getLicense"] [unique_id "1657185244"] [ref "o0,15v37,65"]

.

.

.

---siJN2HMt---Z--

Lo primero que cabe apuntar es que, tal y como hemos comentado en el punto anterior, el nombre del fichero de log está relacionado con la entrada. Básicamente lo que cambia es la extensión, pero el nombre perdura para poder relacionar las salidas con las entradas.

Lo siguiente que hay que poner de manifiesto es que este fichero, a diferencia del resto, depende del detector utilizado, es decir, cada detector tendrá su propio formato de log. En este caso concreto el log generado se divide en diferentes secciones, en las cuales se categoriza la información sobre el lanzamiento de la uri. De dichas secciones extraeremos la información para conformar el fichero resumen *'index'*.

- **03-Index/0days.index:** este fichero es un resumen de la información más relevante generada en el log. El formato que presenta para cada uri detectada en el log es el siguiente:

```
TimeStamp [07/Jul/2022:11:14:04 +0200]  Uri [/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-O%20/tmp/dsifnfnfdwsFFS)]  PLmin [1]  Score [20]  Nattacks [4]  [932105]  [932130]  [932150]  [932160]
```

Aunque como indicamos anteriormente, cada log es diferente y puede aportar más o menos información en función de la naturaleza del detector, existen ciertos campos en el *index* que están presentes y son comunes a todos los IDS. Estos campos son:

1. **TimeStamp:** fecha y hora de llegada de la uri al IDS.
2. **Uri:** identifica la uri sobre la que se está realizando la detección.

El resto de campos estarán presentes o cambiarán su formato en función del detector en uso. Para este caso concreto los campos adicionales son:

3. **PLmin:** propio de detectores que implementan ModSecurity. Especifica el nivel más bajo de paranoia entre las reglas que han 'saltado' durante el análisis de la uri.

4. **Score:** en ModSecurity cada regla tiene asignado un Score. El Score representado en *index* es la suma de todos los scores de reglas que han sido *violadas* por esa uri en cuestión.

   Es como una especie de indicador del nivel de peligrosidad de la uri, cuanto más alto el Score, más reglas y/o de mayor gravedad habrán sido vulneradas.

5. **Nattacks:** número de reglas vulneradas en el lanzamiento de la uri en cuestión. A continuación del número se representa el 'id' de dichas reglas.

- **04A-Attacks/:** en este directorio se alojan dos ficheros de attacks por cada fichero de entrada. Estos ficheros en nuestro caso son:

1. **0days.attacks:** en este fichero se anotan las uris detectadas como attacks junto con el ID/posición de la uri en el fichero de entrada (depende del formato de entrada). En nuestro caso como el fichero de entrada emplea el formato 'basic' utilizaríamos la posición que ocupa la uri en el fichero:

```
Packet [1]

Uri [/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-O%20/tmp/dsifnfnfdwsFFS)]
```

2. **0days-info.attacks:** fichero resumen de los resultados obtenidos del análisis. También están presentes los attacks detectados con información adicional a la mostrada en el fichero '.attacks':

```
---------- Statistics of URIs analyzed----------
[420] input, [286] clean, [134] attacks
---------- Analysis results ----------
Packet [1]

Uri [/appliancews/getLicense?hostName=$(/usr/bin/wget%20http://1.1.1.1:12345/file%20-O%20/tmp/dsifnfnfdwsFFS)]  PLmin [1]  Score [20]  Nattacks [4]  [932105]  [932130]  [932150]  [932160]
.
.
```

- **04B-Clean/0days.clean:** similar al fichero de attacks '.attacks' pero con 'uris' detectadas como 'clean' en lugar de 'attacks':

```
Packet [6]

Uri [/html/en/confAccessProt.html]
.
.
```

ANEXO C: INSERCIÓN DE UN NUEVO DETECTOR EN ACD

En este capítulo detallaremos los pasos a seguir para implementar un nuevo detector en ACD.

La implementación de un nuevo detector a nuestra herramienta viene condicionada por la naturaleza del detector, es decir, si es online (requiere de la presencia de un servidor con un software de detección) u offline (sin servidor).

- **Detector offline:** Para insertar un detector offline deberemos seguir los siguientes pasos:

    1. Introducir en el directorio raíz del proyecto el ejecutable de la API que realizará la carga de reglas contra las uris del fichero de entrada.

    2. Fijar en la variable "API_SCRIPT" del fichero de configuración *framework_config.sh* el nombre del ejecutable introducido.

    3. Insertar en el directorio raíz del proyecto el analizador del log generado por el nuevo detector. El analizador introducido debe generar el *index* siguiendo los requisitos y pautas establecidas en el capítulo de ACD de esta memoria.

    4. Establecer una variable (booleano) en el campo '#DETECTORES DISPONIBLES' del fichero de configuración *framework_config_interna.conf* para el nuevo detector.

    ```
    NUEVO_DETECTOR=1
    ```

    5. Introducir una nueva entrada en el apartado "#CONFIGURACIÓN DE LOS DETECTORES" del fichero de configuración '*framework_config_interna.conf*' para el nuevo detector fijando las variables:

    ```
    LAUNCH_TYPE="offline"
    PATH_AUDIT_LOG="ruta_del_log_generado_por_el_nuevo_detector"
    ```

    6. Crear una nueva entrada en el apartado "#SCRIPTS" del fichero de configuración '*framework_config_interna.conf*' para el nuevo analizador fijando la variable:

    ```
    ANALYZER_SCRIPT="nombre_script_analizador_introducido.py"
    ```

- **Detector online:** Para insertar un detector online deberemos seguir los siguientes pasos:

    1. Introducir en el directorio 'detectores/nombre_nuevo_detector/' el servidor con el software de detección y sus archivos de configuración. Incluir la ruta establecida para el

nuevo detector en el campo '#RUTA DETECTORES' del fichero de configuración *framework_config_interna.conf.*

2. Insertar en el directorio raíz del proyecto el analizador del log generado por el nuevo detector. El analizador introducido debe generar el *index* siguiendo los requisitos y pautas establecidas en el capítulo de ACD de esta memoria.

3. Establecer una variable (booleano) en el campo '#DETECTORES DISPONIBLES' del fichero de configuración *framework_config_interna.conf* para el nuevo detector.

```
NUEVO_DETECTOR=1
```

4. Indicar la ruta de los ficheros de configuración que deberán modificarse en tiempo de ejecución para poder albergar múltiples instancias de la dupla 'servidor + detector' en un mismo equipo en el campo '#FICHEROS Y DIRECTORIOS USADOS POR DETECTORES' del fichero de configuración *framework_config_interna.conf.*

5. Crear una nueva entrada en el apartado "#CONFIGURACIÓN DE LOS DETECTORES" del fichero de configuración '*framework_config_interna.conf*' para el nuevo detector fijando las variables:

```
LAUNCH_TYPE="online-local"
PATH_AUDIT_LOG="ruta_del_log_generado_por_el_nuevo_detector"
```

6. Introducir una nueva entrada en el apartado "#SCRIPTS" del fichero de configuración '*framework_config_interna.conf*' para el nuevo analizador fijando la variable:

```
ANALYZER_SCRIPT="nombre_script_analizador_introducido.py"
```

7. Agregar los cambios necesarios en script 'configura_instancia.sh' para que de forma automática, y con la información ya establecida en los ficheros de configuración de la herramienta realizada en los pasos anteriores, modifique los parámetros requeridos en los ficheros de configuración del servidor + software de detección, para poder correr varias instancias en un mismo equipo simultáneamente.

   Introducir comando de parada del servidor en script 'detener_servidor_instancia.sh'.

ANEXO D: ESTRUCTURA Y SCRIPTS ADT

La estructura de directorios de ADT es la siguiente:

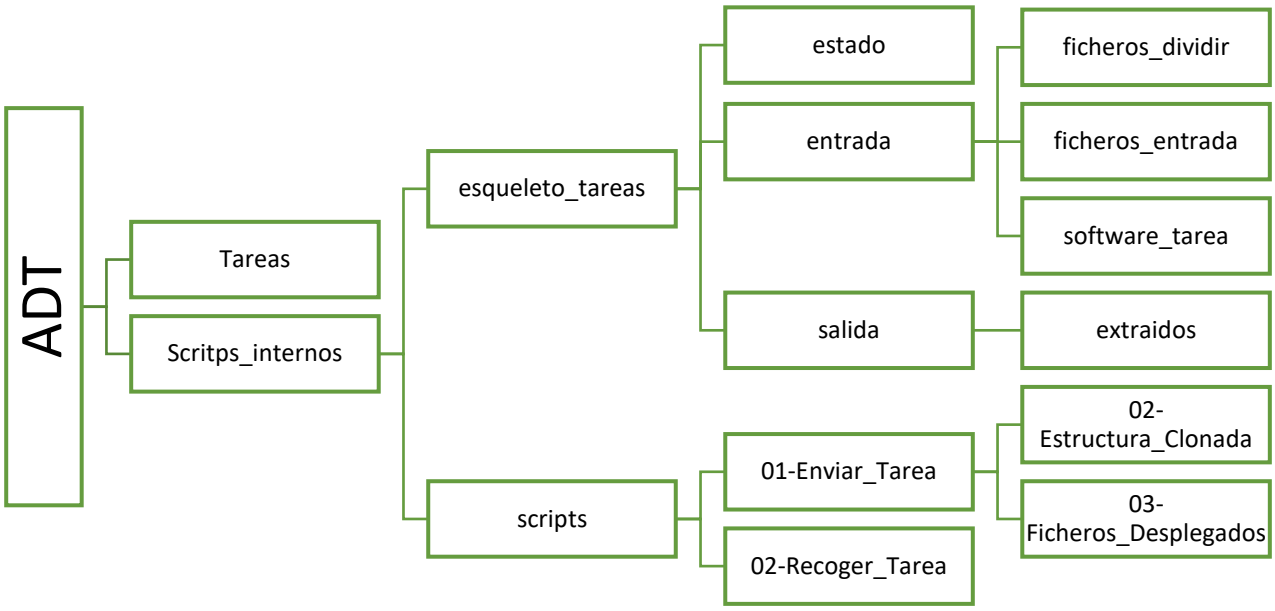


Ilustración 35 Estructura de directorios de ADT

Cuando la creación de una tarea tiene lugar, el contenido de 'Scipts_internos/esqueleto_tareas/' es transferido al directorio 'Tareas/nombre_tarea/'.

Directorios

- **Tareas/:** directorio donde se alojarán las tareas creadas.
- **Scripts_internos/:** directorio que contiene los scripts y directorios necesarios para el funcionamiento de ADT.
- **Scripts_internos /esqueleto tareas/:** directorio que contiene la estructura genérica de una tarea.
- **Tareas/nombre_tarea/estado:** directorio que contiene los ficheros con toda la información relativa al estado de la tarea.
- **Tareas/nombre_tarea/entrada:** directorio que contiene el software de la tarea y los ficheros de entrada a procesar en esta.
- **Tareas/nombre_tarea/salida:** directorio donde se almacenarán los resultados comprimidos recogidos para esa tarea. También contiene el subdirectorio descomprimido de resultados.
- **Tareas/nombre_tarea/entrada/ficheros_dividir/:** directorio donde situaremos los ficheros de entrada que queramos fragmentar.

- **Tareas/nombre_tarea/entrada/ficheros_entrada/:** directorio que contendrá los ficheros de entrada a repartir entre las instancias de la tarea.

- **Tareas/nombre_tarea/entrada/software_tarea/:** directorio donde alojar el software de la tarea a desplegar.

- **Tareas/nombre_tarea/salida/extraidos/:** directorio donde se alojará el subdirectorio de resultados extraídos de los equipos remotos.

- **Scripts_internos/scritps/01-Enviar_Tarea/:** directorio que contiene los scripts relacionados con el proceso de envío de la tarea.

- **Scripts_internos/scritps/02-Recoger_Tarea/:** directorio que contiene los scripts relacionados con el proceso de recogida de resultados.

- **Scripts_internos/scritps/01-Enviar_Tarea/02-Estructura_clonada/:** directorio que alberga tantas copias de la tarea como instancias de despliegue existan.

- **Scripts_internos/scritps/01-Enviar_Tarea/03-Ficheros_Desplegados/:** almacena las instancias comprimidas distribuidas a cada uno de los equipos designados.

Scripts

Los scripts, junto con las rutas, que forman parte del proyecto son los siguientes:

- **install.sh***:* script de instalación que contiene las dependencias necesarias para poner en funcionamiento ADT.

- **menu_global.sh:** menú con opciones que afectan a todas las tareas.

- **Scripts_internos/esqueleto_tareas/menu_tarea.sh:** menú propio de cada tarea. Las opciones disponibles solo atañen a esa tarea en específico.

- **Scripts_internos/esqueleto_tareas/pre-reparto.sh:** es un script que principalmente aporta mejoras de calidad de vida. Permite desplazar aquellos ficheros situados en el directorio 'ficheros_entrada/' que sobrepasen cierto número de líneas establecido, al directorio 'ficheros_dividir/' con objetos de que estos sean fragmentados y repartidos entre los equipos disponibles en trozos más pequeños.

- **Scripts_internos/scripts/start-listen.sh:** inicia el proceso de escucha de mensajes UDP por parte del equipo servidor. El puerto de escucha es establecido en el fichero de configuración interna de ADT.

- **Scripts_internos/scripts/stop_listen.sh:** detiene el proceso de escucha de mensajes UDP en equipo servidor.

- **Scripts_internos/scripts/envia_UDP.py:** script transferido y ejecutado en equipo cliente que realiza el envío del mensaje UDP cliente->servidor.

- **Scripts_internos/scripts/recibe_UDP.py:** script ejecutado en segundo plano en equipo servidor tras el lanzamiento de una tarea que se encarga de recepcionar los mensajes UDP provenientes del equipo cliente, responder con un ACK tras su recepción y desencadenar el proceso de recogida de resultados en el equipo tarea e instancia en cuestión.

- **Scripts_internos/scripts/monitoriza_salida.sh:** script ejecutado en segundo plano en el equipo cliente. Se encarga de monitorizar el directorio en el cual la tarea desplegada en el equipo cliente escribirá cuando haya resultados que recoger. Cuando se produce la escritura, 'monitoriza_salida.sh' invoca al script 'envia_UDP.py' con los parámetros adecuados, dichos parámetros viajarán en el mensaje UDP hasta el equipo servidor.

- **Scripts_internos/scripts/script_remoto.sh:** script desplegado en equipo cliente y encargado de realizar la ejecución de las tareas. Para ello se sirve de sesiones 'byobu'.

- **Scripts_internos/scripts/01-Enviar_Tarea/01B-clonar_estructura-Manual.sh:** encargado de realizar tantas copias de la tarea como instancias equipos e instancias de despliegue hayan configuradas.

- **Scripts_internos/scripts/01-Enviar_Tarea/02B-reparto_ficheros_Manual-Generar_Listado.sh**: realiza el reparto de los ficheros de entradas entre los equipos e instancias disponibles, existen dos modalidades de reparto: por tamaño y secuencial.

- **Scripts_internos/scripts/01-Enviar_Tarea/03-comprimir_y_enviar_ficheros.sh:** encargado de comprimir y enviar tareas a los equipos remotos.

- **Scripts_internos/scripts/01-Enviar_Tarea/04-lanzar_tareas_remotas.sh:** encargado de la ejecución de la tarea y sus instancias desplegadas en los equipos remotos.

- **Scripts_internos/scripts/01-Enviar_Tarea/checkSSH.sh:** comprueba si el acceso ssh está disponible y el tipo de acceso (por certificado o por contraseña).

- **Scripts_internos/scripts/01-Enviar_Tarea/ejecutar_comando.sh:** usado para ejecutar los comandos que figuren en un determinado fichero en un equipo remoto.

- **Scripts_internos/scripts/01-Enviar_Tarea/Fragmentador_ficheros.sh:** encargado de realizar la fragmentación de los ficheros que se encuentren en el fichero 'ficheros_dividir' en los instantes previos al reparto de los ficheros de entrada entre los equipos-instancias disponibles. El número de divisiones se establece en el fichero de configuración interna de ADT.

- **Scripts_internos/scripts/01-Enviar_Tarea/relanzamiento.sh**: script que se ejecuta cuando se detecta un relanzamiento de la tarea. Trata de recuperar los ficheros de resultados que aún no hayan sido recogidos y si no le es posible, realiza el relanzamiento de la tarea en los equipos que se encuentren disponibles con los ficheros de entrada que no han sido procesados hasta el momento.

- **Scripts_internos/scripts/01-Enviar_Tarea/zzzlimpiar.sh:** limpia los ficheros generados por la tarea en los subdirectorios de 'Scripts_internos/scripts/01-Enviar_Tarea'.

- **Scripts_internos/scripts/02-Recoger_Tarea/01-verEstado.sh:** script encargado de crear y actualizar el fichero de estado de la tarea 'listado_ficheros.txt'.

- **Scripts_internos/scripts/02-Recoger_Tarea/estado.sh:** script encargado de crear y actualizar el fichero de estado de la tarea 'estado_nombre_tarea.txt'.

- **Scripts_internos/scripts/02-Recoger_Tarea/02-recoger-resultados.sh:** script encargado de la recogida de resultados en los equipos remotos.

- **Scripts_internos/scripts/02-Recoger_Tarea/03-matar_tareas_remotas.sh:** script para detener la ejecución de una tarea en el equipo e instancia remota seleccionado.

- **Scripts_internos/scripts/02-Recoger_Tarea/04-limpiar_carpeta_equipos_remotos:** elimina los directorios desplegados para esa tarea en el equipo e instancia remoto seleccionado.

- **Scripts_internos/scripts/02-Recoger_Tarea/05-apagar-equipos-remotos.sh:** apaga los equipos seleccionados.

- **Scripts_internos/scripts/02-Recoger_Tarea/consulta.sh:** permite ver en tiempo de ejecución el estado de la tarea en ese instante. La información sobre el estado de la tarea se almacena en el directorio de 'estado' de la tarea.

- **Scripts_internos/scripts/02-Recoger_Tarea/descargar_fichero.sh:** script que permite realizar la descarga de un fichero existente en un equipo remoto.

- **Scripts_internos/scripts/02-Recoger_Tarea/Fusionador_ficheros.py:** encargado de realizar la fusión de los ficheros de resultados obtenidos a partir de la fragmentación de los ficheros de entrada.

- **Scripts_internos/scripts/02-Recoger_Tarea/limpiar_estado.sh:** limpia el directorio de 'estado/' de la tarea.

- **Scripts_internos/scripts/02-Recoger_Tarea/limpiar_tarea.sh:** limpia todos los directorios referentes a esa tarea.

- **Scripts_internos/scripts/02-Recoger_Tarea/sendmail.sh:** se encarga de la notificación vía e-mail al usuario cuando la tarea finaliza o un equipo queda fuera de servicio.

ANEXO E: INSTALACIÓN, CONFIGURACIÓN Y USO DE ADT

Instalación

Para la instalación de ADT basta con descargar el proyecto de GitHub [2].

Y ejecutar el script presente en el directorio raíz: 'install.sh'

Configuración

En este apartado trataremos de definir a partir de los ficheros de configuración de ADT las posibilidades que esta nos otorga. ADT cuenta con un fichero de configuración propio y otro específico para cada tarea.

Cloud_tarea.conf

Tareas/nombre_tarea/cloud_tarea.conf: contiene opciones específicas que solo atañen a esa tarea en específico. Las más destacable son:

- **EQUIPOS_LT:** equipos en los que desplegar la tarea.

- **REPARTO:** el reparto de los ficheros de entrada entre los diversos equipos disponibles siempre trata de ser lo más equilibrado posible con objeto de balancear la carga. Existen dos metodologías de reparto:

    - **Por tamaño:** el reparto de los ficheros se efectúa atendiendo al número de líneas presentes en cada fichero.

    - **Secuencial:** el reparto de los ficheros se efectúa por orden alfabético.

    - **PERIODO_ENTRE_RECOGIDAS:** la recogida automática de los resultados tiene lugar cada 'x' segundos. Valor 0 para desactivar el periodo entre recogidas.

    - **NUM_FICHEROS_A_RECOGER:** la recogida automática de resultados tiene lugar tras la finalización de 'x' número de ficheros (o cuando se detecte el fin de la tarea, lo que ocurra primero).

      Nota: si no se habilitan ninguna de las opciones de recogidas anteriormente mencionadas las recogidas se efectuarán cada vez que la tarea obtenga un resultado.

Cloud_config_interna.conf

Scripts_internos/scripts/cloud_config_interna.conf: opciones que afectan a todo el comportamiento de ADT las más relevantes que caben mencionar son:

-DISPONIBILIDAD DE LOS EQUIPOS: nuestra herramienta cuenta con una opción en su menú global que nos permite conocer el estado de los equipos del laboratorio. En este subapartado trataremos de ilustrar todo lo concerniente en cuanto a disponibilidad.

Nota: el conocer la disponibilidad de los equipos antes de realizar un lanzamiento puede ser muy útil a la hora de determinar qué equipos están disponibles y cuáles son compatibles con nuestra tarea.

Imaginemos que establecemos como equipos disponibles:

```
#Equipos totales
EQUIPOS_TOTALES_DISPONIBLES="1-30"
FILE_ESTADO_EQUIPOS="${DIR_CLOUD_LOCAL}estado_equipos.txt"
```

Nota: los equipos del laboratorio solo comprenden del 1-28, se incluye el 29 y el 30 con fines didácticos. También hay que tener en cuenta que el equipo 'lt04' no se escanea dado que es el equipo servidor que contiene a la propia ADT.

A continuación seleccionamos en el menú global la opción '1 Mostrar estado de los equipos':

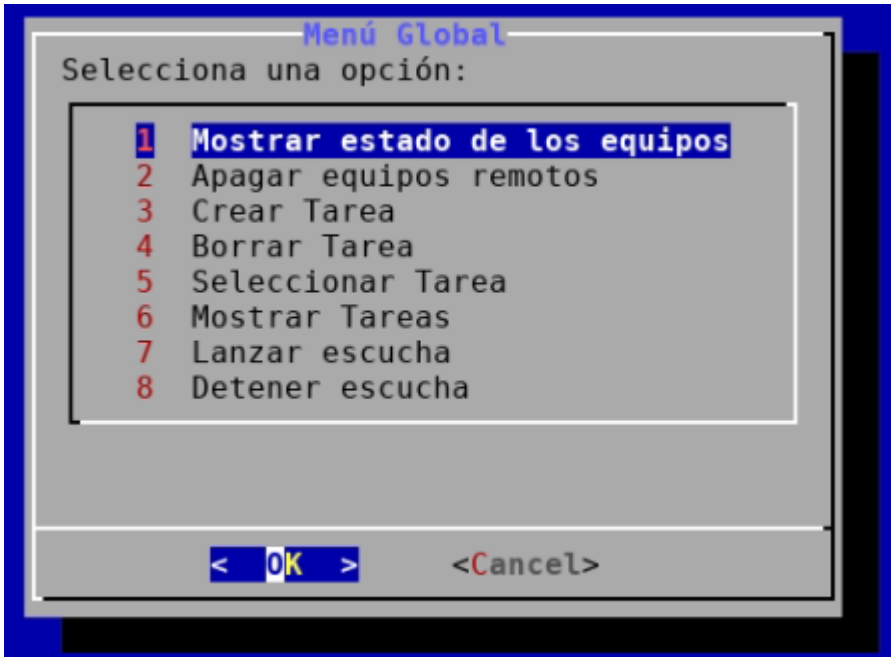


Ilustración 36 Menú global

Esto nos generará un fichero de texto en el directorio raíz de ADT que recibe por nombre 'estado_equipos.txt'.

Última actualización: Sun May 8 18:28:06 CEST 2022

Equipos analizados [29] Equipos Disponibles [21] Equipos NO disponibles [8]

Equipo lt01: OFF --- --- No disponible: apagado o sin conexión

Equipo lt02: ON Linux linux1 Disponible SSH certificado

Equipo lt03: ON Linux linux1 Disponible Sin Certificado SSH. Disponible clave

Equipo lt05: ON --- --- No disponible Sin conexión SSH (Problema de acceso)

Equipo lt06: ON --- --- No disponible Sin conexión SSH (Servicio SSH cerrado)

Equipo lt07: ON --- --- No disponible Sin conexión SSH (Problema de acceso)

Equipo lt08: ON Linux lmoviles No disponible: Versión SO incompatible SSH certificado

Equipo lt09: ON Linux linux1 Disponible SSH certificado

Equipo lt10: ON Linux linux1 Disponible SSH certificado

Equipo lt11: ON Linux linux1 Disponible SSH certificado

Equipo lt12: ON Linux linux1 Disponible SSH certificado

Equipo lt13: ON Linux linux1 Disponible SSH certificado

Equipo lt14: ON Linux linux1 Disponible SSH certificado

Equipo lt15: ON Linux linux1 Disponible SSH certificado

Equipo lt16: ON Linux linux1 Disponible SSH certificado

Equipo lt17: ON Linux linux1 Disponible SSH certificado

Equipo lt18: ON Linux linux1 Disponible SSH certificado

Equipo lt19: ON Linux linux1 Disponible SSH certificado

Equipo lt20: ON Linux linux1 Disponible SSH certificado

Equipo lt21: ON Linux lmoviles No disponible: Versión SO incompatible SSH certificado

Equipo lt22: ON Linux linux1 Disponible SSH certificado

Equipo lt23: ON Linux linux1 Disponible SSH certificado

Equipo lt24: ON Linux linux1 Disponible SSH certificado

Equipo lt25: ON Linux linux1 Disponible SSH certificado

Equipo lt26: ON Linux linux1 Disponible SSH certificado

Equipo lt27: ON Linux linux1 Disponible SSH certificado

Equipo lt28: ON Linux linux1 Disponible SSH certificado [Carpeta Tarea con poco espacio: 1194 MB < 10000 MB]

Equipo lt29: OFF --- --- No disponible: apagado o sin conexión

Equipo lt30: OFF --- --- No disponible: apagado o sin conexión

Existen varios puntos interesantes a comentar sobre el fichero generado.

El primero de estos puntos es que **todos** los ficheros de estado de ADT incorporan la fecha y hora de la última actualización del fichero de estado en cuestión.

El segundo es el resumen del sondeo de los equipos; para este caso se ha realizado el sondeo sobre 29 equipos de los cuales 21 se han detectado como disponibles y 8 como no disponibles.

El formato del fichero de estado de los equipos es el siguiente:

- Primera columna: equipo sobre el que se realiza el sondeo.
- Segunda columna: equipo encendido (ON) / apagado (OFF).
- Tercera columna: sistema operativo del equipo.
- Cuarta columna: versión del sistema operativo.
- Quinta columna: resultado de disponibilidad del equipo.
- Sexta columna: tipo de acceso ssh (por certificado o por clave).

La disponibilidad de los equipos no se rige únicamente porque el equipo se encuentre apagado o encendido, entran en juego múltiples factores. A continuación, veremos cuáles son los factores que pueden determinar que un equipo no se encuentre disponible:

- El primero de ellos es el más evidente; si un equipo se encuentra apagado figurará como 'No disponible'.

- Variable 'ANTE_CARPETA_TAREA_EXISTE'. Nos permite adoptar tres valores:

    - 0: ante carpeta de tarea ya existente en el equipo remoto consulta al usuario qué desea hacer.

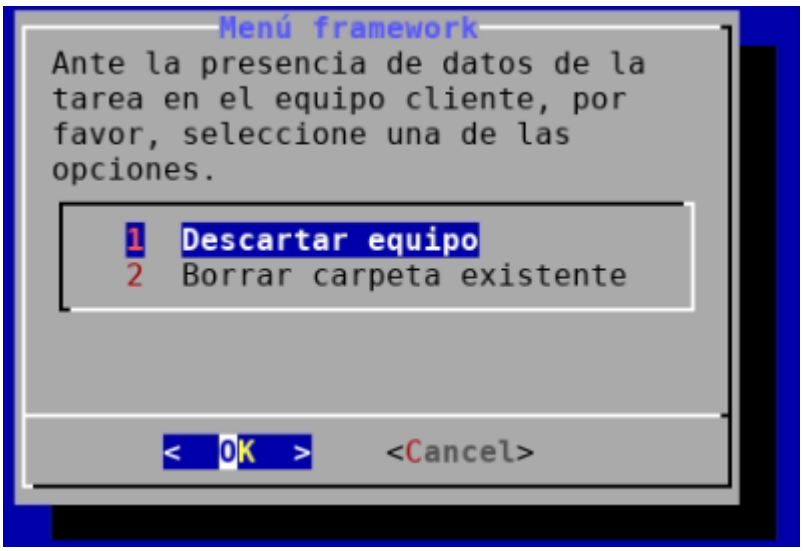


Ilustración 37 Menú ante datos ya existentes en equipo remoto

    - 1: descartar el equipo si existe directorio de la tarea, es decir, considerarlo como no disponible (esta acción se realizaría automáticamente sin establecer una consulta con el usuario).

- 2: borrar carpeta existente y continuar con el análisis (consideraría al equipo como disponible. Esta acción se realizaría automáticamente sin establecer una consulta con el usuario).

- Variable 'ANTE_ESPACIO_CARPETA_INFERIOR_UMBRAL'. Nos permite adoptar tres valores:

  - 0: ante un incumplimiento del requisito de capacidad mínimo fijado en la variable 'CARPETA_TAREA_UMBRAL_MINIMO' consulta al usuario que acción desea realizar.

Ilustración 38 Menú de gestión de la capacidad

  - 1: descartar el equipo si no se satisface la condición de capacidad, es decir, considerarlo como no disponible (esta acción se realizaría automáticamente sin establecer una consulta con el usuario).

  - 2: proseguir a pesar de no cumplir con el requisito de capacidad (consideraría al equipo como disponible. Esta acción se realizaría automáticamente sin establecer una consulta con el usuario).

- Otro aspecto que un equipo debe satisfacer para considerarse como 'disponible', es que el directorio del equipo cliente donde será transferida la tarea no sea de solo lectura. En caso de serlo, será considerado como 'no disponible' y se imprimirá el mensaje "No disponible [Carpeta Tarea SOLO LECTURA]"

- Si el puerto ssh establecido en la variable 'PUERTO_SSH' de la configuración no se encuentra a la escucha en el equipo cliente, o no se dispone de las credenciales adecuadas para acceder, el equipo será considerado como 'no disponible' y se imprimirán los mensajes "Sin conexión SSH (Servicio SSH cerrado)" o "Sin conexión SSH (Problema de acceso)" según corresponda.

  Nota: si el acceso ssh es válido, se detallará si el acceso ha tenido lugar por certificado o por contraseña tal y como se muestra en la figura del fichero 'estado_equipos.txt'.

- Por último, en las variables 'SO_COMPATIBLES' y 'TIPO_SO_COMPATIBLES', se establecen los sistemas operativos y las versiones de los mismos que serán considerados como compatibles con la tarea. Si los sistemas y sus versiones fijados en la configuración no coinciden con los obtenidos por el sondeo en los equipos clientes, el equipo será considerado como 'No disponible'.

Antes de finalizar con este punto sobre la disponibilidad, queremos poner de relieve que la captura obtenida del fichero 'estado_equipos.txt', trata de recrear un escenario que recoge la mayor casuística posible en cuanto a disponibilidad se refiere.

También es importante poner de manifiesto que, si durante el lanzamiento de una tarea algunos de los equipos seleccionados se detectan como 'no disponibles', se le dará al usuario la posibilidad de continuar el lanzamiento con los equipos disponibles restantes, o detener el análisis para seleccionar otros equipos.

Ejemplo:

Si queremos desplegar una tarea en los equipos lt02 y lt03 pero el equipo lt02 no se encuentra disponible, se nos mostrará el mensaje siguiente:

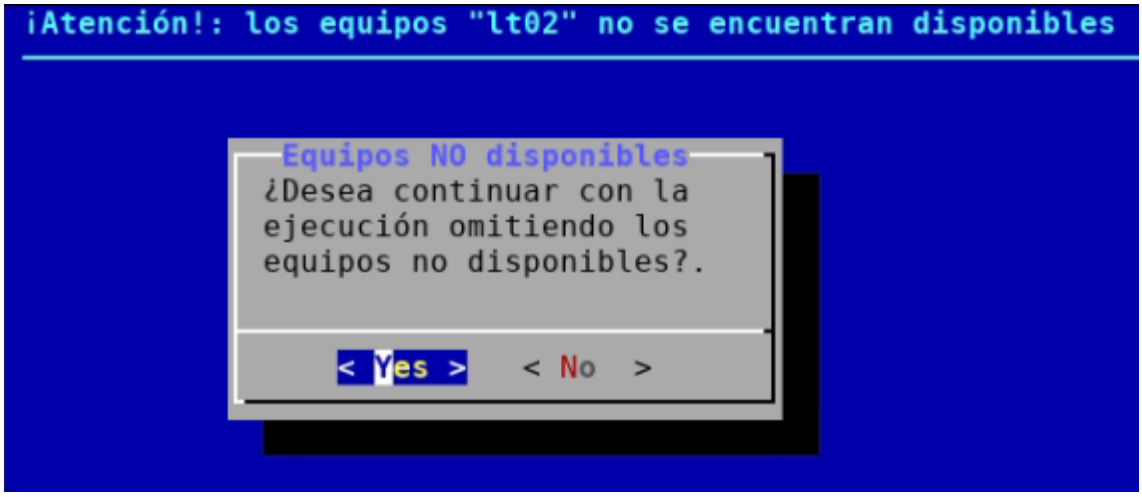


Ilustración 39 Aviso de equipos seleccionados no disponibles

Si pulsamos en 'Yes' continuaremos el lanzamiento de la tarea usando únicamente el equipo lt03, que es el único disponible. Si seleccionamos 'No', se detendrá el lanzamiento.

-**OTRAS FUNCIONALIDADES:**

- **DIVISIONES:** establece el número de partes en los que se fragmentarán los ficheros ubicados en el directorio 'entrada/ficheros_dividir/' de la tarea (hasta un máximo de 1000 divisiones).

- **N_INSTANCIA:** número de instancias de la tarea que se crearán y ejecutarán en cada equipo.

- **FAST_MODE:** si se fija a '1' algunas de las operaciones realizables desde el menú de la tarea no pedirán confirmación.

- **TIEMPO_ENTRE_REINTENTOS:** tiempo (en segundos) en los que el equipo cliente reenviará el mensaje UDP si no ha recibido ACK por parte del servidor.

- **NUM_MAX_REINTENTOS:** número máximo de intentos de reenvío que el cliente puede realizar para hacer llegar el mensaje UDP de recogida de resultados al servidor.

Prueba de uso

En este apartado realizaremos una prueba práctica del uso de ADT tratando que recoja la mayor casuística posible para mostrar cada una de sus funcionalidades. La tarea escogida para ilustrar todo esto será la también desarrollada en este proyecto, ACD.

Los pasos a seguir son los siguientes:

1. Descargamos el software del proyecto. Para ello nos situaremos en el directorio donde deseemos realizar la descarga, en nuestro caso '/opt/' y ejecutaremos lo siguiente:

```
cd /opt
git clone https://github.com/pgf53/cloud.git
```

2. A continuación sitúese en el directorio raíz del proyecto y ejecute el script de instalación:

```
cd cloud/
./install.sh
```

3. Una vez hecho esto, ya podemos utilizar *cloud*. Lo primero que haremos será crear la tarea *framework,* y para ello ejecutaremos*:*

```
./menu_global.sh
```

Cuando se nos despliegue el menú pulsamos en la opción '3 Crear Tarea':

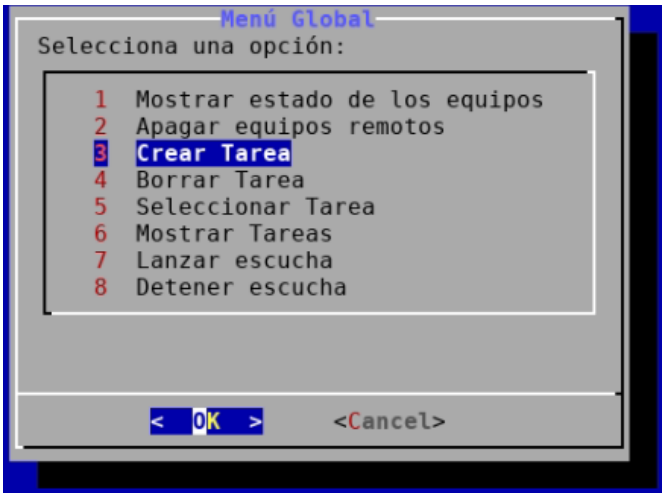


Ilustración 40 Menú global "crear tarea"

Se nos mostrará un submenú para establecer el nombre de la tarea (este campo no puede estar vacío, si no lo completamos nos saltará un mensaje de error y no se creará la tarea, lo mismo ocurre si el nombre de la tarea ya existe). En nuestro caso la llamaremos 'framework':

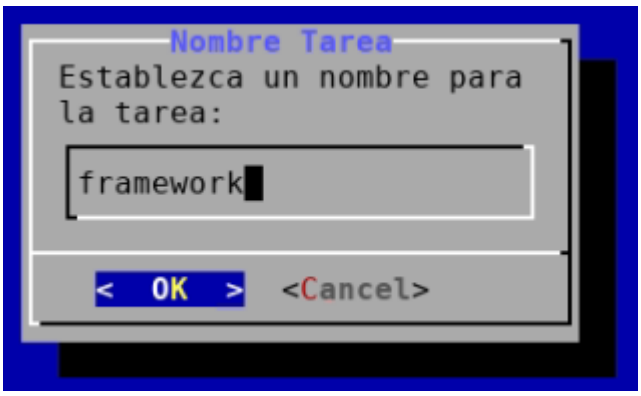


Ilustración 41 Menú de nombre

Ilustración 42 Aviso de tarea creada con éxito

Podemos comprobar las tareas creadas si desde el menú global seleccionamos la opción '6 Mostrar Tareas':

Ilustración 43 Tareas disponibles

4. Una vez tengamos la tarea creada, copiaremos el software de ACD en el directorio 'Tareas/framework/entrada/software_framework':

```
cp -rf /opt/framework/* /opt/cloud/Tareas/framework/entrada/software_framework/
```

5. Situaremos los ficheros de entrada en los directorios correspondientes, en nuestro caso, nuestro fichero de entrada es el correspondiente al dataset 'De_ataques' 'Fwaf-badqueries.uri'.

   Como tiene una extensión lo suficientemente considerable (48k líneas) lo situaremos en el directorio de división de ficheros. Los ficheros situados en este directorio serán fragmentados en tantos trozos como establezcamos en la configuración; dichos trozos serán desplazados posteriormente al directorio 'Tareas/framework/entrada/ficheros_entrada/' para ser repartidos entre los equipos e instancias disponibles.

   Si no deseamos realizar ningún tipo de división sobre alguno, o la totalidad de los ficheros de entrada, bastaría con situarlos en el directorio 'Tareas/framework/entrada/ficheros_entrada/' en lugar de 'Tareas/framework/entrada/ficheros_dividir/':

```
cp  /opt/Fwaf-badqueries.uri /opt/cloud/Tareas/framework/entrada/ficheros_dividir/
```

6. Ajustamos la configuración del lanzamiento de la tarea. Para ello establecemos las variables de los ficheros de configuración de esta forma:

   - cloud_framework.conf:

```
#### Equipos en los que desplegar la tarea (separados por espacio)
# Se admiten rangos usando guion, e.g. "03 10-15")
EQUIPOS_LT="03 05 06"
# Estado Tarea: Nombre Comando (script, ...) cuyo proceso comprobar si esta aun en ejecucion en cada equipo (grep de "ps")
PROCESO_PARA_ESTADO="./0-Framework.sh"
# SubCarpeta de la Estructura de la Tarea en la que ubicar los ficheros a analizar
SUBDIR_TAREA_ENTRADA="01-Uri/"
# SubCarpeta de la Estructura de la Tarea a recoger (vacio o "./" para todo)
SUBDIR_REMOTO_RECOGIDA="Resultados/"
REPARTO="tamaño"  #el reparto de los ficheros se efectúa atendiendo al número de líneas presentes en cada fichero.
PERIODO_ENTRE_RECOGIDAS=0

#La recogida de resultados tiene lugar cada 'x' segundos, valor 0 para desactivarlo (se recoge cada vez que termine un fichero)
NUM_FICHEROS_A_RECOGER=0

#La recogida de resultados tiene lugar tras finalizar 'x' ficheros, valor 0 para desactivarlo (se recoge cada vez que termine un fichero)
```

- cloud_config_interna.conf:

```
#SO compatibles
SO_COMPATIBLES="linux"
TIPO_SO_COMPATIBLES="linux1"
# Carpeta de despliegue del cluster de la tarea en equipo remoto
CARPETA_TAREA="/opt/"
#Número de divisiones de ficheros fragmentados.
DIVISIONES=6
#Número de instancias por equipo
N_INSTANCIA=2
#IP del equipo que implementa cloud (en nuestro caso equipo lt04)
EQUIPO_SERVIDOR="${PREFIJO_NOMBRE_EQUIPO}04"
```

Como puede deducirse de la configuración establecida, hemos seleccionado los equipos "lt03, lt05 y lt06" como equipos de despliegue de la tarea.

También podemos observar como el fichero de entrada escogido para su análisis en ACD, 'Fwaf-badqueries.uri', será fragmentado en 6 partes, y que cada una de ellas serán divididas entre los tres equipos seleccionados.

Es importante destacar que al seleccionar 'N_INSTANCIA=2' cada equipo contendrá **dos** instancias de la tarea *framework* ejecutándose simultáneamente.

Una vez hecho todo esto, realizaremos el lanzamiento de la tarea.

7. Procedemos a iniciar el lanzamiento. Para ello ejecutaremos el script de 'menu_framework.sh', este es accesible mediante la ruta: 'Tareas/framework/menu_framework.sh', o desde el menú global clicando sobre la opción '5 Seleccionar Tarea':

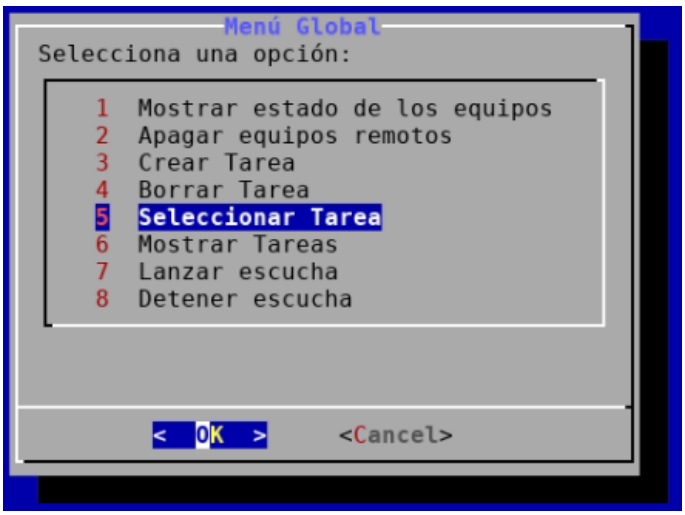


Ilustración 44 Menú de selección de tarea

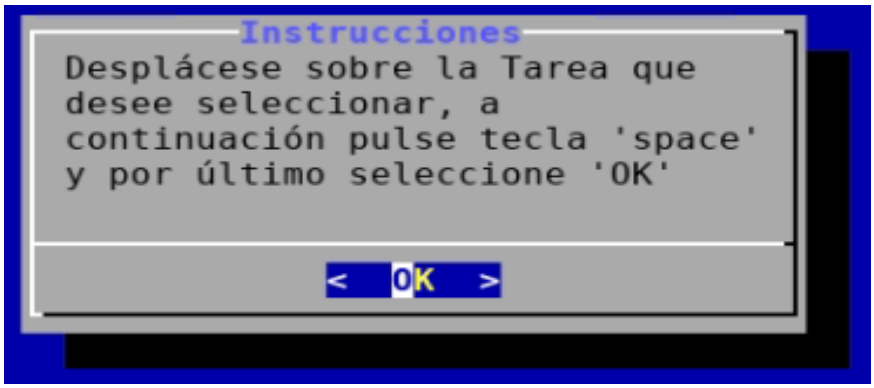


Ilustración 45 Instrucciones de selección de tarea

Ilustración 46 Selección de tareas disponibles

Ilustración 47 Tarea seleccionada

Una vez accedemos al menú de la tarea, pulsamos sobre la opción ‘1 Lanzamiento Completo’.

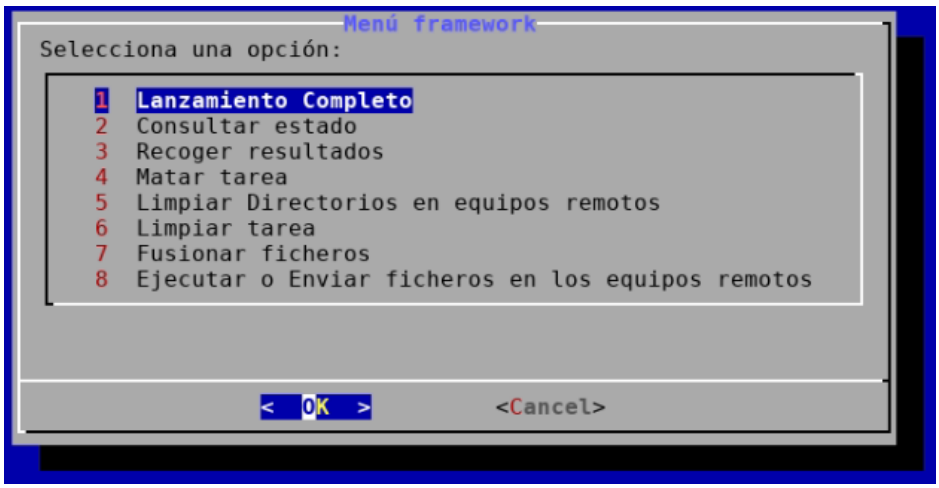


Ilustración 48 Menú de tarea "lanzamiento completo"

8. Si hemos ido siguiendo todos los pasos hasta ahora, comenzará el despliegue y ejecución de la tarea. Dicha ejecución puede desglosarse en las siguientes fases:

8.1. Fase I: comprobación de disponibilidad de los equipos seleccionados:

Ilustración 49 Estado de los equipos

A su vez, se creará en el directorio de 'estado/' de la tarea un fichero que recibe por nombre 'estado_equipos_inicial.txt' con el estado inicial de los equipos:

```
Última actualización: Sat Jul  9 16:30:41 CEST 2022


Equipos analizados [3]  Equipos Disponibles [3] Equipos NO disponibles [0]


Equipo lt03:    ON      Linux   linux1  Disponible      SSH certificado
Equipo lt05:    ON      Linux   linux1  Disponible      SSH certificado
Equipo lt06:    ON      Linux   linux1  Disponible      SSH certificado
```

A continuación se nos mostrará por pantalla un mensaje notificándonos sobre la disponibilidad de los equipos seleccionados:

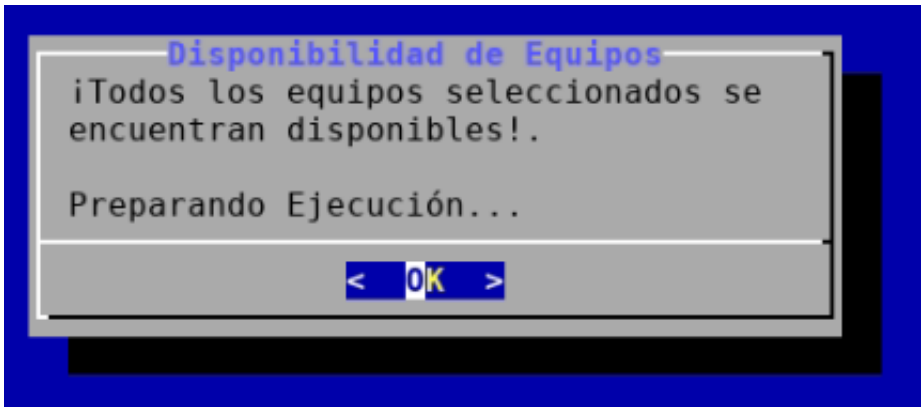


Ilustración 50 Mensaje de todos los equipos selecionados disponibles

8.2. Fase II: se procederá a la clonación de la tarea para cada uno de los equipos e instancias disponibles configurados. En nuestro caso serían seis (tres equipos con dos instancias por equipo):

8.3. Fase III [opcional]: si el directorio de la tarea 'entrada/ficheros_dividir/' contiene uno o más ficheros, realizará la división de estos de la manera más equitativa posible, en tantas partes como se hayan configurado, y desplazará dichos fragmentos al directorio 'entrada/ficheros_entrada/':

```
[root@lt04-C-L1 /opt/cloud/Tareas/framework] # ls entrada/ficheros_entrada/
Fwaf-badqueries_000.uri  Fwaf-badqueries_002.uri  Fwaf-badqueries_004.uri
Fwaf-badqueries_001.uri  Fwaf-badqueries_003.uri  Fwaf-badqueries_005.uri
```

Ilustración 51 Ficheros de entrada fragmentados

8.4. Fase IV: se inicia el reparto de los ficheros de entrada presentes en 'entrada/ficheros_entrada/' entre las diferentes copias de la tarea. En nuestro caso, y como vimos en el apartado donde detallábamos ciertos aspectos de la configuración, hemos seleccionado el reparto por 'tamaño'.

Una vez tiene lugar el reparto se nos generará en el directorio de 'estado/' de la tarea un fichero 'Reparto_Ficheros_framework.txt', con información sobre el reparto de los ficheros de entrada realizado:

```
----------Resumen del reparto----------
Equipo lt03_framework1:
Fwaf-badqueries_000.uri
Número de líneas totales asignadas: 8010
Número de Ficheros asignados: 1

Equipo lt03_framework2:
Fwaf-badqueries_001.uri
Número de líneas totales asignadas: 8010
Número de Ficheros asignados: 1

Equipo lt05_framework1:
Fwaf-badqueries_002.uri
Número de líneas totales asignadas: 8010
Número de Ficheros asignados: 1

Equipo lt05_framework2:
Fwaf-badqueries_003.uri
Número de líneas totales asignadas: 8010
Número de Ficheros asignados: 1

Equipo lt06_framework1:
Fwaf-badqueries_004.uri
Número de líneas totales asignadas: 8010
Número de Ficheros asignados: 1

Equipo lt06_framework2:
Fwaf-badqueries_005.uri
Número de líneas totales asignadas: 8015
```

Podemos apreciar que el reparto se ha efectuado de la manera más equitativa posible entre las diferentes instancias disponibles atendiendo al número de líneas.

8.5. Fase V: envío de las instancias de la tarea a los equipos remotos.

Cada una de las copias de la tarea con sus ficheros de entrada asignados se comprimen y se envían a los equipos remotos.

8.6. Fase VI: ejecución de la tarea en equipo remoto.

Una vez tenemos desplegado los ficheros comprimidos, ADT irá accediendo a cada uno de los equipos e instancias, descomprimirá el archivo con la tarea, y ejecutará esta junto con el script de monitorización del directorio de resultados que posibilita la recogida automática de los mismos.

Una vez ha comenzado la ejecución de la tarea en los equipos remotos, se creará en el directorio de 'estado/' de la tarea los siguienes ficheros:

- Listado_ficheros.txt: indica en qué equipo-instancia se encuentra cada uno de los ficheros de entrada. También muestra el progreso de los ficheros.

Última actualización: Sat Jul 9 20:25:41 CEST 2022

#########Instrucciones#######

Primera columna: nombre del fichero. Segunda columna: ¿Terminado en equipo remoto?. Tercera columna: ¿Recogido en el servidor?. Cuarta columna: equipo en el que se ha desplegado el fichero. Quinta columna: Instancia de despliegue del fichero

PROGRESO: (0/6) FICHEROS FINALIZADOS

Fwaf-badqueries_000.uri	no	no	lt03	framework1
Fwaf-badqueries_001.uri	no	no	lt03	framework2
Fwaf-badqueries_002.uri	no	no	lt05	framework1
Fwaf-badqueries_003.uri	no	no	lt05	framework2
Fwaf-badqueries_004.uri	no	no	lt06	framework1
Fwaf-badqueries_005.uri	no	no	lt06	framework2

- Estado_framework.txt: indica el progreso de la tarea en cada uno de los equipos-instancia:

```
Última actualización: Sat Jul  9 20:25:40 CEST 2022


PROGRESO: (0/6) EQUIPOS FINALIZADOS


Equipo lt03 framework1:	ON	Linux	linux1	Disponible	SSH certificado	(0/1)	Interrumpida
Equipo lt03 framework2:	ON	Linux	linux1	Disponible	SSH certificado	(0/1)	Interrumpida
Equipo lt05 framework1:	ON	Linux	linux1	Disponible	SSH certificado	(0/1)	Interrumpida
Equipo lt05 framework2:	ON	Linux	linux1	Disponible	SSH certificado	(0/1)	Interrumpida
Equipo lt06 framework1:	ON	Linux	linux1	Disponible	SSH certificado	(0/1)	Interrumpida
Equipo lt06 framework2:	ON	Linux	linux1	Disponible	SSH certificado	(0/1)	Interrumpida
```

8.7. Fase VII: recogida de resultados.

A medida que se procesan los ficheros de entrada, los resultados son recogidos de forma automática. En este punto, si lo deseamos podemos hacer uso de la opción del menú '2 consultar estado' para poder ver el estado de la tarea en ese instante. No obstante, cada vez que tiene lugar la recogida automática de un fichero, tiene lugar también una actualización automática de los ficheros de estado.

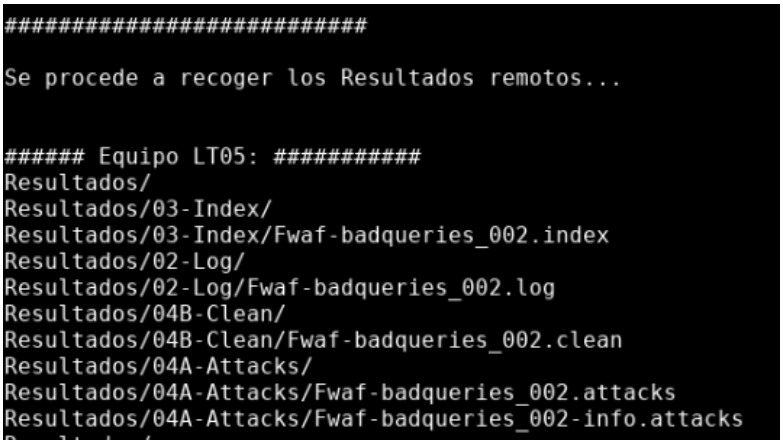


Ilustración 52 Recogida de resultados

Cuando ha tenido lugar la recogida de todos los resultados generados por las diferentes instancias de la tarea distribuida, los ficheros de estado quedan de la siguiente forma:

- Listado_ficheros.txt:

Última actualización: Sat Jul 9 20:29:47 CEST 2022

#########Instrucciones#######

Primera columna: nombre del fichero. Segunda columna: ¿Terminado en equipo remoto?. Tercera columna: ¿Recogido en el servidor?. Cuarta columna: equipo en el que se ha desplegado el fichero. Quinta columna: Instancia de despliegue del fichero

PROGRESO: (6/6) FICHEROS FINALIZADOS

Fwaf-badqueries_000.uri	si	si	lt03	framework1
Fwaf-badqueries_001.uri	si	si	lt03	framework2
Fwaf-badqueries_002.uri	si	si	lt05	framework1
Fwaf-badqueries_003.uri	si	si	lt05	framework2
Fwaf-badqueries_004.uri	si	si	lt06	framework1
Fwaf-badqueries_005.uri	si	si	lt06	framework2

- Estado_framework.txt:

```
Última actualización: Sat Jul  9 20:29:47 CEST 2022


PROGRESO: (6/6) EQUIPOS FINALIZADOS


Equipo lt03 framework1:        ON      Linux   linux1  Disponible      SSH
certificado     (1/1)   Finalizada
Equipo lt03 framework2:        ON      Linux   linux1  Disponible      SSH
certificado     (1/1)   Finalizada
Equipo lt05 framework1:        ON      Linux   linux1  Disponible      SSH
certificado     (1/1)   Finalizada
Equipo lt05 framework2:        ON      Linux   linux1  Disponible      SSH
certificado     (1/1)   Finalizada
Equipo lt06 framework1:        ON      Linux   linux1  Disponible      SSH
certificado     (1/1)   Finalizada
Equipo lt06 framework2:        ON      Linux   linux1  Disponible      SSH
certificado     (1/1)   Finalizada
```

9. Fusión de los resultados: si ya hemos obtenido todos los ficheros de resultados solo resta fusionarlos (recordemos que dividimos el fichero de entrada para repartirlos entre varios equipos con objeto de mejorar el rendimiento). Para ello pulsaremos en la opción '7 Fusionar ficheros'.

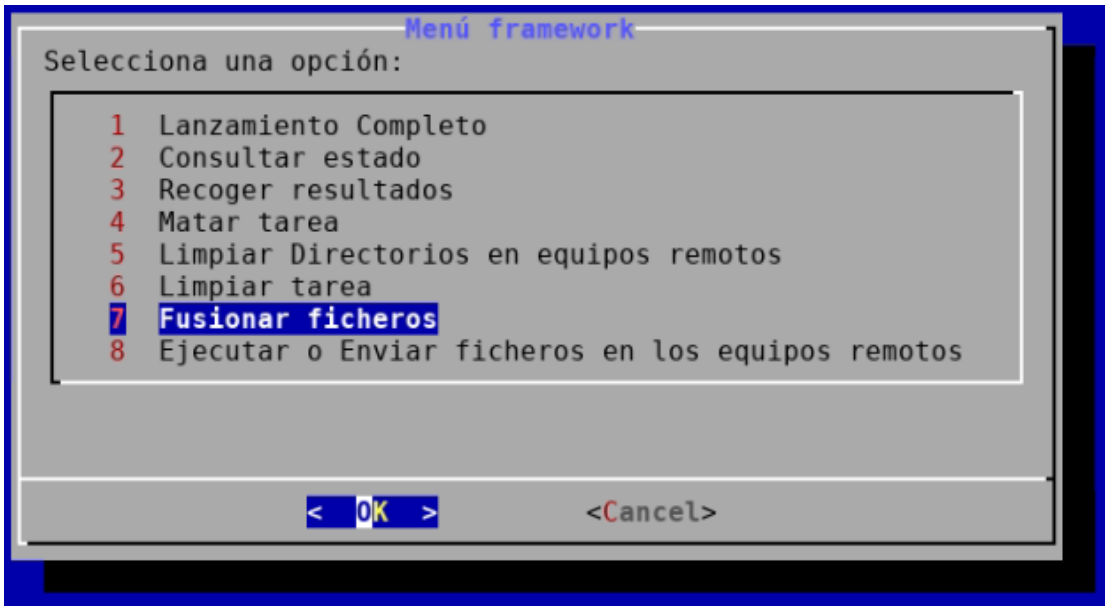


Ilustración 53 Menú de la tarea "fusionar ficheros"

De esta manera pasaremos de tener los resultados de esta forma:

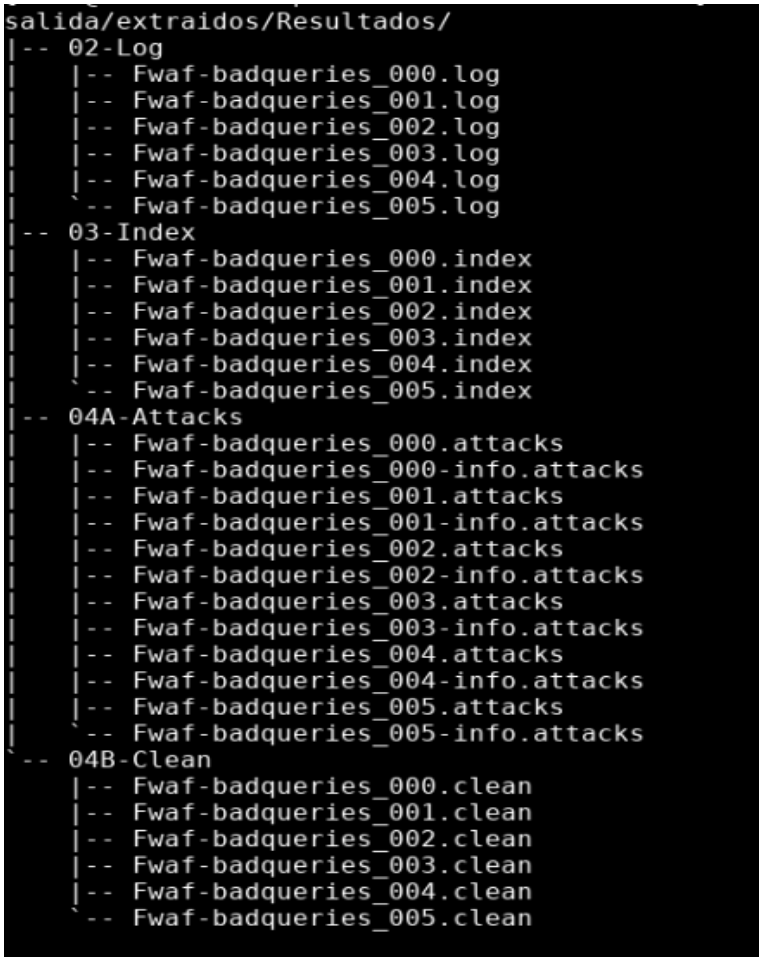


Ilustración 54 Ficheros de resultados fragmentados

A esta otra:

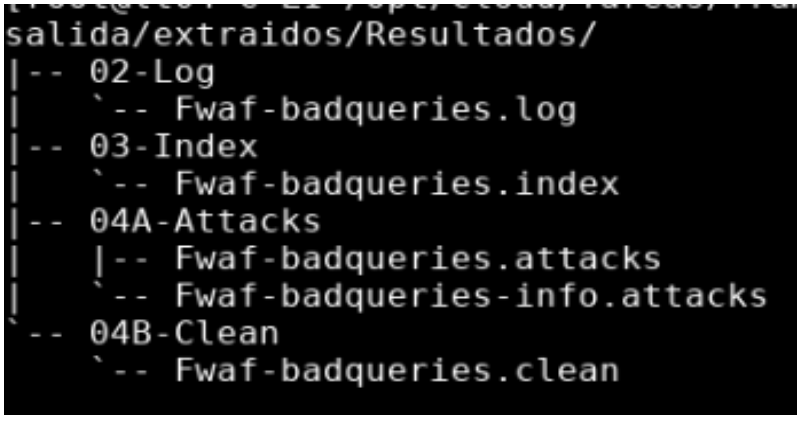


Ilustración 55 Fichero de resultados unificados

Nota: como puede deducirse de las capturas, el directorio de los ficheros de resultados descomprimidos se encuentra en el directorio ‘salida/extraidos/’ de la tarea.

De esta manera, concluiría el ejemplo de envío, ejecución y recogida automática de resultados para la tarea 'framework' desplegada. No obstante, el software también cuenta con cierta capacidad de reacción ante eventualidades que puedan surgir durante el lanzamiento/ejecución de una tarea, pudiendo efectuar un **relanzamiento** de la misma.

Explicaremos con más detalle en qué consiste el proceso de relanzamiento basándonos en el ejemplo que acabamos de presentar.

Imaginemos dos escenarios del ejemplo anterior que permitan ilustrar en qué consiste el proceso de relanzamiento.

En el primero de ellos, el mensaje UDP de la instancia *lt03-framework1* de fin de la tarea no llegan al servidor debido a una saturación en la red, sin embargo el análisis sí ha concluido.

En este escenario descrito, podemos observar que en el fichero 'listado_ficheros.txt', el fichero de entrada asociado a la instancia *lt03-framework1* aparece como no finalizado y no recogido (no se ha recibido el mensaje de finalización, por lo que para ADT no ha finalizado).

Última actualización: Sun Jul 10 11:29:24 CEST 2022

#########Instrucciones#######

Primera columna: nombre del fichero. Segunda columna: ¿Terminado en equipo remoto?. Tercera columna: ¿Recogido en el servidor?. Cuarta columna: equipo en el que se ha desplegado el fichero. Quinta columna: Instancia de despliegue del fichero

PROGRESO: (5/6) FICHEROS FINALIZADOS

Fwaf-badqueries_000.uri	no	no	lt03	framework1
Fwaf-badqueries_001.uri	si	si	lt03	framework2
Fwaf-badqueries_002.uri	si	si	lt05	framework1
Fwaf-badqueries_003.uri	si	si	lt05	framework2
Fwaf-badqueries_004.uri	si	si	lt06	framework1
Fwaf-badqueries_005.uri	si	si	lt06	framework2

Mientras que en 'estado_framework.txt' la tarea figura como 'Interrumpida'. Algo esperable, puesto que no se han obtenido los resultados y la tarea no se muestra en ejecución:

```
Última actualización: Sun Jul 10 11:29:24 CEST 2022


PROGRESO: (5/6) EQUIPOS FINALIZADOS


Equipo lt03 framework1:     ON    Linux   linux1   Disponible     SSH
certificado     (0/1)   Interrumpida
Equipo lt03 framework2:     ON    Linux   linux1   Disponible     SSH
certificado     (1/1)   Finalizada
Equipo lt05 framework1:     ON    Linux   linux1   Disponible     SSH
certificado     (1/1)   Finalizada
Equipo lt05 framework2:     ON    Linux   linux1   Disponible     SSH
certificado     (1/1)   Finalizada
Equipo lt06 framework1:     ON    Linux   linux1   Disponible     SSH
certificado     (1/1)   Finalizada
Equipo lt06 framework2:     ON    Linux   linux1   Disponible     SSH
certificado     (1/1)   Finalizada
```

Si en este momento seleccionamos la opción del menú de la tarea '1 Lanzamiento Completo', la tarea detectará que es un relanzamiento y tratará de recuperar los ficheros que no han podido obtenerse de los equipos en los que la tarea figura como 'Interrumpida'.

Como en nuestro caso la tarea sí pudo finalizar en todos los equipos, al iniciar el relanzamiento pueden obtenerse todos los ficheros de resultados, y no es necesario una segunda ejecución de la tarea.

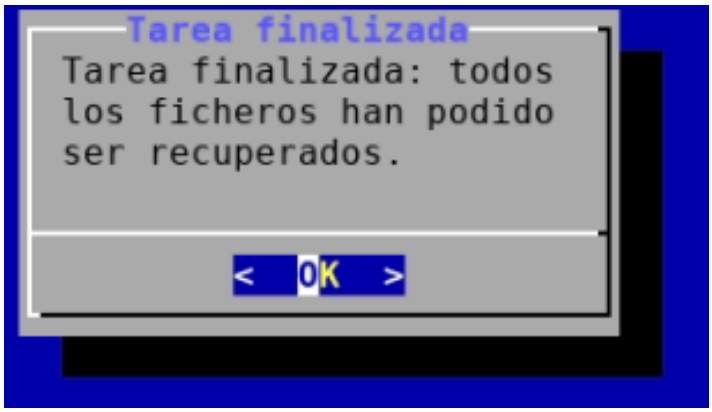


Ilustración 56 Recuperación exitosa de ficheros en relanzamiento

En el segundo escenario que nos plantearemos, el equipo lt03 se apagará durante la ejecución de la tarea. Los ficheros de estado generados serán similares a los del escenario anterior. La diferencia radica, en que en este caso los ficheros de resultados no serán recuperables puesto que no fueron generados.

Lo que hará ADT ante esta eventualidad será:

1. Tratar de recuperar los ficheros de resultados que faltan.
2. Si no es posible la recuperación, iniciará un relanzamiento de la tarea con los equipos seleccionados que figuren como disponibles con los ficheros de entrada no procesados.

   A su vez, generará un directorio con la fecha y hora actual donde almacenará los ficheros de estado de la ejecución anterior con el fin de preservarlos, y creará nuevos ficheros de estado correspondientes al nuevo despliegue y ejecución de la tarea.

Como los ficheros asignados a lt03 ‘Fwaf-badqueries_000.uri’ y ‘Fwaf-badqueries_001.uri’ no han podido procesarse puesto que el equipo lt03 ha interrumpido la ejecución de la tarea, se iniciará un relanzamiento distribuyendo esos ficheros de entrada entre los equipos que sí están disponibles.

En nuestro caso, como hemos definido dos instancias por equipo, estos dos ficheros serán ‘absorbidos’ por las dos instancias del equipo ‘lt05’:

Última actualización: Sun Jul 10 12:55:39 CEST 2022

#########Instrucciones#######

Primera columna: nombre del fichero. Segunda columna: ¿Terminado en equipo remoto?. Tercera columna: ¿Recogido en el servidor?. Cuarta columna: equipo en el que se ha desplegado el fichero. Quinta columna: Instancia de despliegue del fichero

PROGRESO: (2/2) FICHEROS FINALIZADOS

Fwaf-badqueries_000.uri	si	si	lt05	framework1
Fwaf-badqueries_001.uri	si	si	lt05	framework2

```
Última actualización: Sun Jul 10 12:55:39 CEST 2022


PROGRESO: (2/2) EQUIPOS FINALIZADOS


Equipo lt05 framework1:        ON      Linux   linux1  Disponible      SSH
certificado     (1/1)   Finalizada
Equipo lt05 framework2:        ON      Linux   linux1  Disponible      SSH
certificado     (1/1)   Finalizada
```

Para concluir, podemos comprobar cómo se ha creado un directorio con la fecha y hora actual que contiene los datos sobre el estado de la tarea de la ejecución anterior:

```
[root@lt04-C-L1 /opt/cloud/Tareas/framework] # tree estado/
estado/
|-- estado_equipos_inicial.txt
|-- estado_framework.txt
|-- listado_ficheros.txt
|-- Reparto_Ficheros_framework.txt
`-- Sun-Jul-10-12:55:02-CEST-2022
    |-- estado_equipos_inicial.txt
    |-- estado_framework.txt
    |-- listado_ficheros.txt
    `-- Reparto_Ficheros_framework.txt
```

Ilustración 57 Directorio de estado de la tarea tras relanzamiento

REFERENCIAS

[1] Código de ACD. Disponible en: https://github.com/pgf53/framework

[2] Código de ADT. Disponible en: https://github.com/pgf53/cloud

[3] Resultado de las detecciones. Disponible en: https://github.com/pgf53/tfm-detecciones

[4] Sistema de detección de intrusiones. Protección de datos, lopd. [Consultado el 14 de julio 2022] Disponible en: https://protecciondatos-lopd.com/empresas/sistema-deteccion-intrusiones-ids/

[5] IL (Inspector Log). Jesús Esteban Díaz Verdejo. Universidad de Granada.

[6] MLA (ModSecurity Log Analyer). Francisco Javier Muñoz Calle. Universidad de Sevilla.

[7] Oracle. ¿Qué es un WAF?. [Consultado el 14 de julio 2022] Disponible en: https://www.oracle.com/es/database/security/que-es-un-waf.html#:~:text=Un%20Web%20Application%20Firewall%20(WAF,HTTPS%20y%20modelos%20de%20tr%C3%A1fico

[8] Datasheet de Checkpoint 3200 Security Gateway. Checkpoint. [Consultado el 14 de julio 2022] Disponible en: https://www.checkpoint.com/downloads/products/3200-security-gateway-datasheet.pdf

[9] Documento TFM Rafael Díaz Fernández. Detección de ataques a servidores Web con WAF de Checkpoint. Universidad de Sevilla, 2020.

[10] Dinahosting. ¿Qué es apache y para qué sirve?. [Consultado el 14 de julio 2022]. Disponible en: https://dinahosting.com/ayuda/que-es-apache-y-para-que-sirve/

[11] Openwebinars. ¿Qué es C?. [Consultado el 14 de julio 2022]. Disponible en: https://openwebinars.net/blog/que-es-c/

[12] Becas-Santander. ¿Qué es Python?. [Consultado el 14 de julio 2022]. Disponible en: https://www.becas-santander.com/es/blog/python-que-es.html

[13] Trajano. Shellscript. [Consultado el 14 de julio 2022]. Disponible en: http://trajano.us.es/~fjfj/shell/shellscript.html

[14] What-is-an-intrusion-detection-system-ids [Consultado el 9 de septiembre 2022]. Disponible en: https://www.checkpoint.com/cyber-hub/network-security/what-is-an-intrusion-detection-system-ids/

[15] Waf-vs-ips-cual-es-la-diferencia [Consultado el 9 de septiembre 2022]. Disponible en: https://www.lanner-america.com/es/blog-es/waf-vs-ips-cual-es-la-diferencia/

[16] GitHub. SpiderLabs/ModSecurity. [Consultado el 14 de julio 2022]. Disponible en: https://github.com/SpiderLabs/ModSecurity/wiki/

[17] Snort. [Consultado el 14 de julio 2022]. Disponible en: https://www.snort.org/

[18] Nemesida. Nemesida-WAF. [Consultado el 14 de julio 2022]. Disponible en: https://nemesida-waf.com/

[19] Documento 'CheckScript-Funcionamiento_y_Uso.pdf'. Rafael Díaz Fernández. Universidad de Sevilla, 2020.

[20] NemesidaScript. Carlos Cagigao Bravo.

[21] Estudio MLA-ModSecurityV2 Vs MLA-ModSecurityV3. Disponible en: https://github.com/pgf53/tfm-detecciones/tree/main/estudios/MLA-ModSecurityV2_Vs_MLA-ModSecurityV3

[22] Estudio solapamiento entre IDS. Disponible en: https://github.com/pgf53/tfm-detecciones/tree/main/estudios/estudio_solapamiento_IDS

[23] Estudio de impacto de etiquetado en BIBLIO. Disponible en: https://github.com/pgf53/tfm-detecciones/tree/main/estudios/estudio_impacto_etiquetado