Universidad de la República
Facultad de Ingeniería
Instituto de Computación
Uruguay

Minería de procesos para la mejora de la seguridad de aplicaciones web

Tamara Techera
Pablo Ibáñez
Marcelo Bruno

Proyecto de Grado
Ingeniería en Computación
Universidad de la República

Montevideo, Uruguay, Diciembre de 2020

Tutores: Dr. Ing. Daniel Calegari
Dr. Ing. Gustavo Betarte
Universidad de la República

Resumen

La gran mayoría de las aplicaciones web multi-capas son diseñadas e implementadas sin tener en cuenta buenas prácticas de desarrollo seguro de código. Por lo tanto es muy frecuente que el descubrimiento y explotación de vulnerabilidades en este tipo de sistemas se den como resultado de un proceso de ensayo y error proveniente de un atacante. Es así que se vuelven necesarias las técnicas de detección y prevención de ataques. El desarrollo de este tipo de técnicas involucra procedimientos que ayudan a discernir entre el comportamiento de un usuario válido del sistema y un agente malicioso.

Al momento de implementar este tipo de técnicas, un problema no menor es la cantidad de aplicaciones ya desplegadas en las organizaciones y que deben de ser protegidas por este tipo de soluciones. Un enfoque no invasivo, es decir, que no requiere modificar la aplicación original, consiste en añadir una capa externa responsable de filtrar los datos que fluyen hacia y desde la aplicación. Concretamente, a nivel de una aplicación web, lo que está siendo extensivamente adoptado es el uso de un WAF (*Web Application Firewall*). Un WAF tiene la capacidad de registrar y analizar las peticiones HTTP a un sitio web, por lo que se podría llegar a derivar el comportamiento de un usuario a partir de la información que registra y de los eventos que genera.

El objetivo general del proyecto es explorar técnicas de Minería de Procesos para la detección de ataques en aplicaciones web, explotando las capacidades que tiene un WAF para el registro de eventos.

Una propuesta de interés se basa en la aplicación de técnicas de Minería de Procesos para la detección de ataques, a partir de identificar desviaciones en el comportamiento esperado del sistema web visto como un proceso de negocio, acorde a un modelo de comportamiento de referencia.

Para lograr el objetivo se aplicaron técnicas de Minería de Procesos a los efectos de detectar comportamientos maliciosos, previo procesamiento de la información del WAF a través de su filtrado, agrupamiento de eventos y definición de eventos críticos.

Palabras clave: Minería de procesos, Seguridad Informática, WAF, Modsecurity, Prom.

Índice general

1 Introducción

La seguridad informática [1] se refiere a los procesos y metodologías que son diseñados e implementados para proteger la información o los datos de accesos no autorizados.

En las aplicaciones web, existen diversos problemas referidos a la seguridad de los datos, como los definidos por OWASP [2]. Esta fundación mantiene una lista con las diez vulnerabilidades de seguridad más frecuentas. Frente a la aparición de una nueva vulnerabilidad es de suma importancia contar con una solución rápida y no invasiva, es decir, que no requiera cambios en la implementación de la aplicación.

En este sentido, se ha universalizado el uso de un *Web Application Firewall* [3], como implementación de la estrategia de *virtual patching* [4], la cual permite una solución no invasiva frente a nueva vulnerabilidad o posible ataque sobre un sitio web.

Por otra parte, el uso masivo de la mayoría de las aplicaciones genera grandes volúmenes de información, como registros de auditoría, registros de logs etc. Estos datos pueden ser utilizados aplicando técnicas de Minería de Procesos [5].

Concretamente, esta disciplina permite, entre otras cosas, descubrir modelos de uso bienintencionado, y posteriormente comparar estos modelos con otros usos, midiendo que tanto se corresponden. Esto puede ser de utilidad para extraer conclusiones a partir de esta información y determinar desviaciones en el comportamiento de los usuarios. El uso de las técnicas de dicha disciplina sin un preprocesamiento previo de la información, presenta algunas limitaciones que se irán presentando durante el desarrollo del informe, y se mostrarán las soluciones propuestas para sortearlas.

El objetivo de este proyecto es poner en práctica algunas de las técnicas de Minería de Procesos, explotando las capacidades que tiene un WAF para el registro de eventos de uso de una aplicación web. Se parte de un trabajo previo [6], realizando un análisis del mismo y adaptando algunas de sus particularidades al contexto de uso de un *Web Application Firewall*.

Concretamente, los objetivos del presente trabajo son:

- Obtener un conocimiento base sobre Minería de Procesos y virtual patching utilizando un WAF.
- Aplicar técnicas de Minería de Procesos existentes para la detección de vulnerabilidades en aplicaciones web.
- Extender dichas técnicas para explotar las capacidades de un WAF.
- Evaluar las técnicas aplicadas a partir de casos de estudio.

El informe está organizado en cuatro secciones principales.

En la sección 2, se presentan los conceptos teóricos, necesarios para comprender las siguientes secciones. Se muestra un breve resumen sobre seguridad informática, se describe la estrategia de *virtual patching* y se presenta información sobre Minería de Procesos y sus diferentes aplicaciones.

En la sección 3 se describe brevemente un trabajo relacionado, donde se propone utilizar las técnicas de Minería de Procesos e Ingeniería Guiada por Modelos para mejorar la Seguridad de Sistemas de Información Web. Posteriormente, se muestran las modificaciones necesarias para el presente caso. Además, se describen los problemas típicos al aplicar Minería de Procesos sin preprocesamiento adecuado y un primer acercamiento a las soluciones de estos problemas.

En la sección 4, se detallan los pasos necesarios para aplicar las técnicas de Minería de Procesos. Se presenta la herramienta que se desarrolló para realizar el procesamiento y transformación de la información.

Por último, en la sección 5 se muestran los resultados obtenidos sobre un sitio web, en particular, un sitio de *e-commerce*. Se aplican los pasos previamente propuestos y se analizan los resultados obtenidos.

2 Marco Teórico

En esta sección se presentan los conceptos fundamentales de Minería de Procesos, así como información sobre los *Web Application Firewall* (WAF) y el acercamiento a un caso particular, ModSecurity.

2.1. Seguridad en aplicaciones web

La seguridad es uno de los aspectos fundamentales a la hora de desarrollar o mantener aplicaciones en la web. La fundación OWASP [2] trabaja para mejorar la seguridad, a través de sus proyectos de software de código abierto liderados por la comunidad.

Esta fundación, mantiene una lista con los diez riesgos de seguridad más importantes en aplicaciones web [7], que se publica y actualiza cada tres años.

A modo de ejemplo, se describen dos de los tipos de ataque más comunes, que han estado en la lista desde su origen:

- **Injection.** Las fallas de inyección, como la inyección de SQL, NoSQL, OS y LDAP, ocurren cuando se envían datos que no son de confianza a un intérprete como parte de un comando o consulta. Los datos maliciosos del atacante pueden engañar al intérprete para que ejecute comandos no deseados o acceda a los datos sin la debida autorización.
- **Cross-Site Scripting XSS.** Los defectos de XSS ocurren cuando una aplicación incluye datos que no son de confianza en una nueva página web, sin la validación o el escape adecuados, o actualiza una página web existente con datos proporcionados por el usuario. Esto puede realizarse mediante una API de navegador que pueda crear HTML o JavaScript. XSS permite a los atacantes ejecutar scripts en el navegador de la víctima, y de tal forma podrían secuestrar sesiones de usuarios, desfigurar sitios web o redirigir

al usuario a sitios maliciosos.

Existen otros ataques que se generan comúnmente como pueden ser:

- *Broken Authentication*
- *Sensitive Data Exposure*
- XML *External Entities* (XXE)
- *Broken Access Control*
- *Security Misconfiguration*
- *Insecure Deserialization*
- *Using Components with Known Vulnerabilities*
- *Insufficient Logging & Monitoring*

2.1.1. Web Application Firewall

Frente a una vulnerabilidad de seguridad y desde una perspectiva puramente técnica, la estrategia número uno a seguir es corregir la vulnerabilidad identificada dentro del código fuente de la aplicación web. Este concepto está universalmente aceptado, tanto por los expertos en seguridad de aplicaciones web como por los propietarios de los sistemas. Desafortunadamente, en situaciones comerciales del mundo real, surgen muchos escenarios en los cuales actualizar el código fuente de una aplicación web no es fácil o directamente no es posible por diferentes razones, como por ejemplo, el código es legado, no se cuenta con el tiempo necesario para realizar la corrección, no se tiene acceso al código fuente, etc.

En este contexto es que surge el concepto de *virtual patching*: una capa de aplicación de políticas de seguridad que evita la explotación de una vulnerabilidad conocida. [4]

Un WAF [3] (*Web Application Firewall*) es una implementación concreta de la estrategia *virtual patching*, descrita anteriormente. Consiste en un dispositivo de *hardware* o *software* que permite proteger a los servidores de aplicaciones web de determinados ataques específicos en Internet, mediante la aplicación y mapeo de diferentes reglas relacionadas a las peticiones HTTP. Su principal diferencia con un *firewall* tradicional de capa de red es que permite filtrar contenido de aplicaciones web específicas, mientras que un *firewall* de capa de red filtra tráfico entre servidores (*endpoints*).

Los WAF permiten filtrar o bloquear tráfico HTTP desde y hacia aplicaciones web. Hay dos tipos de WAF: Los que se residen en la red y los que se basan en el servidor de aplicaciones (residen en el servidor).

La Figura 2.1 muestra la ubicación de un WAF respecto del resto de los actores, en el contexto de una arquitectura web cliente y servidor.

El funcionamiento de un WAF consiste en una combinación lógica basada en reglas, análisis sintáctico y firmas para detectar y prevenir ataques. Examina todas las peticiones realizadas a un servidor web, así como las respuestas del servidor de acuerdo con su conjunto de reglas.

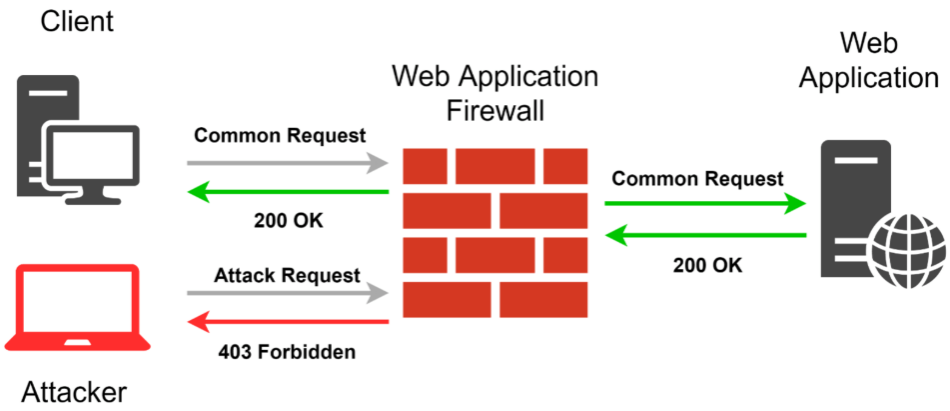

Figura 2.1: Representación de un WAF entre el cliente y servidor de una aplicación web

Si la comprobación es correcta, la petición HTTP se transfiere al sitio web para recuperar el contenido. De lo contrario, se llevan a cabo las acciones predefinidas, permitiéndose tomar distintas acciones, como registrar el comportamiento sospechoso en un archivo de log o directamente bloquear la respuesta, entre otras.

2.1.2. ModSecurity

ModSecurity [8] es un *Web Application Firewall* que ejecuta como módulo del servidor web Apache [9]. Provee mecanismos de protección de ataques a nivel de aplicación web y de monitoreo de tráfico HTTP. Está disponible como *software* libre bajo la licencia GNU *General Public License*. Esta herramienta permite ganar visibilidad dentro del tráfico HTTP(S) y provee un lenguaje de reglas y una API para implementar protecciones avanzadas. Esto significa que es posible filtrar tráfico HTTP directamente en el servidor Web, según el contenido de las peticiones de los clientes, lo cual permite detectar y bloquear ataques de tipo XSS (*Cross Site Scripting*), SQLi (*SQL injection*), etc.

El conjunto de reglas centrales de ModSecurity (CRS) de OWASP es un conjunto de reglas de detección de ataques genéricas para usar con ModSecurity o *firewalls* de aplicaciones web compatibles. El CRS tiene como objetivo proteger las aplicaciones web de una amplia gama de ataques, incluida la lista con los diez riesgos de seguridad más importantes, con un mínimo de alertas falsas.

ModSecurity hace posible el registro completo de transacciones HTTP, lo que permite registrar solicitudes y respuestas completas. Sus configuraciones también permiten tomar decisiones detalladas sobre qué se registra exactamente y cuándo, lo que garantiza que sólo se registren los datos relevantes. Como algunas de las solicitudes y/o respuestas pueden conte-

ner datos confidenciales en ciertos campos, ModSecurity puede configurarse para enmascarar estos campos antes de que se escriban en el registro de auditoría. Además de proporcionar facilidades de registro, ModSecurity permite monitorear el tráfico HTTP en tiempo real para detectar ataques. Esto permite reaccionar ante eventos sospechosos. Implementa un lenguaje de reglas flexible, diseñado especialmente para trabajar con solicitudes HTTP y funciona en diversos sistemas operativos.

ModSecurity puede actuar de inmediato para evitar que los ataques lleguen a las aplicaciones web. Hay tres enfoques de uso común:

- ***Negative security model.*** Un modelo de seguridad negativo supervisa las solicitudes en busca de anomalías, comportamiento inusual y ataques de aplicaciones web comunes. Mantiene puntuaciones de anomalía para cada solicitud, direcciones IP, sesiones de aplicación y cuentas de usuario. Las solicitudes con puntuaciones de anomalías altas se registran o rechazan por completo.
- ***Positive security model.*** Cuando se implementa un modelo de seguridad positivo, solo se aceptan las solicitudes que se sabe que son válidas, todo lo demás es rechazado. Este modelo requiere el conocimiento de las aplicaciones web que se están protegiendo. Por lo tanto, un modelo de este tipo funciona mejor con las aplicaciones que se usan mucho pero que rara vez se actualizan para minimizar el mantenimiento del modelo.
- ***Known weaknesses and vulnerabilities.*** Con ModSecurity, las aplicaciones pueden protegerse desde el exterior, sin tocar el código fuente de la aplicación (e incluso sin acceso a él).

Reglas
Para su funcionamiento, ModSecurity define un conjunto de reglas a aplicar en cada petición HTTP que se realice sobre el sitio [10]. Una regla consiste en una directiva declarada en un archivo de configuración, en la cual se especifican patrones a analizar y acciones a ejecutar por el servidor en caso de que la regla aplique.

Cada regla definida se ajusta a un mismo formato, que se muestra a continuación:

SecRule VARIABLES OPERATOR [ACTIONS]

Las tres partes tienen los siguientes significados:

- Las VARIABLES especifican qué lugares chequear en una transacción HTTP. Algunos ejemplos de variables son: ARGS (todos los argumentos, incluida la carga POST), REQUEST_METHOD (método de solicitud utilizado en la transacción), REQUEST_HEADERS (se puede usar como una colección de todos los encabezados de solicitud o para inspeccionar los encabezados específicos), etc.
- El OPERATOR especifica una expresión regular, patrón o palabra clave que se debe verificar en la(s) variable(s). Los operadores comienzan con el carácter @.

- Las ACTIONS especifican qué hacer si la regla coincide. Las acciones se definen en siete categorías: disruptivas (utilizadas para permitir que ModSecurity realice una acción, por ejemplo, permitir, bloquear, etc.), flujo (afectar el flujo, por ejemplo, omitir), metadatos (utilizados para proporcionar más información sobre las reglas), variable (utilizar cambiar y eliminar variables), registro (utilizado para influir en la forma en que se realiza el registro), acciones especiales (utilizado para proporcionar acceso a otra clase de funcionalidad) y misceláneas (contienen acciones que no pertenecen a ninguno de los otros grupos).

A modo de ejemplo, la siguiente regla se utiliza para evitar ataques XSS mediante la comprobación de un patrón <script> en los parámetros de solicitud y el encabezado. Genera un mensaje *XSS Attack* con una respuesta de estado 404:

```
SecRule ARGS|REQUEST_HEADERS "@rx <script>" id:101,
msg: 'XSS Attack',severity:ERROR,deny,status:404
```

En este ejemplo la interpretación completa de la regla es la siguiente:

VARIABLES
ARGS: Todos los argumentos de la transacción, incluyendo el POST *payload*
REQUEST_HEADERS: encabezado HTTP.

OPERATOR
"@rx <script>": la expresión regular (en este caso <script>) pasada por parámetro

ACTIONS
id, msg, severity, deny, status: estas son las acciones que serán realizadas si el patrón coincide.

- *id:101:* identificador que se asigna a la regla.
- *msg:"XSS Attack":* mensaje personalizado asignado a la regla.
- *Severity:* ERROR: severidad de la regla, que puede ser: EMERGENCY (0), ALERT (1), CRITICAL (2), ERROR (3), WARNING (4), NOTICE (5), INFO (6) y DEBUG (7).
- *deny:* detiene el procesamiento de la regla e intercepta la transacción. Es una acción disruptiva.
- *status:404:* especifica el estado HTTP de respuesta si la acción es denegada o redirigida.

Logs
Respecto a este tema, ModSecurity cuenta con una gran variedad de modos y funcionalidades. En particular, para el presente trabajo interesa el modo concurrente, en el cual por cada petición HTTP se genera un archivo de log diferente. Cada archivo de log consiste en un archivo de texto plano, dividido en distintas secciones identificadas por letras mayúsculas. [11]

Para el presente estudio, las secciones de mayor relevancia son:

- A: de esta sección, interesa obtener la dirección IP del cliente.
- B: de esta sección interesa el método HTTP (GET o POST) y el recurso de la petición.
- F: encabezados de la respuesta.
- H: de esta sección, se obtienen las posibles reglas que aplicaron el la petición.

En el caso del ejemplo anterior, para la regla definida con identificador 101, la sección H mostraría lo siguiente:

```
--69387b6c-H--
Message: Access denied with code 404 (phase 1).
Pattern match "<script>" at REQUEST_FILENAME.
[file "/etc/modsecurity/modsecurity.conf"] [line "20"]
[id "101"] [msg "Custom Rule: XSS Attack"]
[tag "Blacklist Rules"]
```

2.2. Minería de Procesos

La Minería de Procesos [5] es una disciplina relativamente joven, cuyo objetivo es utilizar los datos de eventos con diferentes finalidades, como por ejemplo identificar cuellos de botella, anticipar problemas, registrar desviaciones de políticas establecidas, recomendar medidas a tomar y/o agilizar los procesos.

La Minería de Procesos confronta los datos de eventos (es decir, el comportamiento observado) contra los modelos de procesos (hechos de forma manual o descubiertos automáticamente). Los datos de los eventos están relacionados con modelos de procesos concretos; los modelos de procesos son descubiertos a partir de datos de eventos o los datos de eventos pueden ser reproducidos en los modelos para analizar la conformidad y la *performance*.

Esta disciplina es también parte de *process science*, que es el campo que combina conocimiento de la tecnología de la información y el de la gestión de ciencias para mejorar y ejecutar procesos operativos. Por ejemplo, las técnicas de Minería de Procesos pueden ser utilizadas para descubrir modelos de procesos a partir de datos de eventos. Reproduciendo esos datos, se pueden identificar cuellos de botella y si se respeta el comportamiento preestablecido.

Las técnicas de Minería de Procesos se pueden aplicar a cualquier tipo de procesos operativos (organizaciones y sistemas), por ejemplo, para analizar los procesos de tratamiento en hospitales, mejorar los procesos de atención al cliente en una corporación, comprender el comportamiento de navegación de los clientes, etc. No se limita al descubrimiento automatizado de procesos: también se puede utilizar para verificar el cumplimiento, diagnosticar desviaciones, mejorar el desempeño, predecir tiempos de flujo y recomendar acciones.

Los datos registrados por los sistemas de información pueden ser usados para brindar una mejor vista de un proceso real, es decir, las desviaciones pueden ser analizadas y se pueden mejorar los modelos. El objetivo de la Minería de Procesos es descubrir, monitorear y mejorar procesos reales a través de la extracción de conocimiento de logs de eventos disponibles en los sistemas de hoy en día.

La Figura 2.2 muestra que la Minería de Procesos establece vínculos entre los procesos reales y sus datos por un lado, y por otro con modelos de procesos.

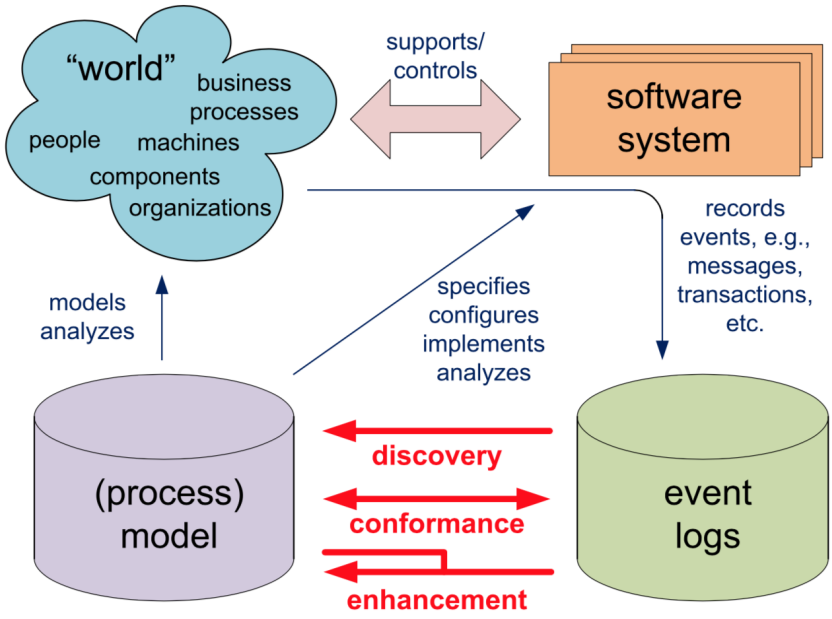

Figura 2.2: Minería de procesos, extraíado de [5]

Como se puede ver en la figura 2.2, los sistemas de *software* ayudan a controlar y dar soporte a elementos de la realidad, como pueden ser corporaciones, personas, procesos de negocio, etc. También es posible contar con modelos de procesos que analicen o representen esos elementos. Los mismos pueden utilizarse para especificar el comportamiento de los sistemas de *software*. Por otra parte, los sistemas generan logs de eventos tales como mensajes, transacciones, etc, los cuales también se corresponden con la realidad. Allí se observa cómo la Minería de Procesos puede integrarse tanto en la realidad como en los modelos de procesos, a partir del uso de sistemas de *software*. La Minería de Procesos puede utilizarse para descubrir esos modelos de la realidad a partir de los logs de eventos generados por los sistemas de software. También para detectar si el modelo se adecua a la realidad reflejada en los logs de eventos, comparando los modelos y los logs, o para mejorar los modelos a partir de los logs y modelos existentes.

Los logs de los eventos obtenidos a partir un sistema de *software*, pueden ser generados por el propio sistema, o se pueden generar estos registros sin tener que modificar los sistemas existentes, por ejemplo haciendo uso de *virtual patching*. Como se explicó en la sección anterior, consiste en agregar una capa, a nivel de capa de aplicación, a un sistema existente. Esto se implementa a través de un WAF.

Si bien los WAF implementan políticas de seguridad con el fin de evitar la explotación de vulnerabilidades, también se pueden utilizar para capturar las peticiones HTTP de los sistemas

de software con los cuales se utilice y, a partir de ellos, procesarlos y generar log de eventos con los cuales aplicar Minería de Procesos. A partir de ellos se puede, por una parte, descubrir procesos en sistemas existentes (*discovery*) para luego poder identificar si logs generados a partir del uso real del sistema se corresponden con el modelo (*conformance checking*), y así identificar comportamientos de usuarios potencialmente maliciosos. A esto, se le pueden adicionar las funcionalidades que brindan los WAF para evitar ataques. Combinando estas dos técnicas, se puede incrementar la seguridad en los sitios web, no sólo a partir de la aplicación de reglas proporcionadas por un WAF a ataques ya conocidos, sino identificando comportamientos potencialmente maliciosos a partir de modelos de procesos, utilizando al WAF como herramienta para generar logs.

Distintas aplicaciones de la Minería de Procesos
Como se mencionó anteriormente, los logs de eventos generados por los sistemas pueden ser utilizados con los siguientes tres propósitos:

Discovery. Se trata de una técnica de descubrimiento que a partir de un registro de eventos, produce un modelo sin utilizar ninguna información a priori. Un ejemplo es el algoritmo *alpha*. Este algoritmo toma un registro de eventos y produce una *Petri net* que representa el comportamiento registrado en el log.

Conformance checking. Consiste en comparar un modelo de proceso existente contra un registro de eventos del mismo proceso. Esto se puede utilizar para verificar si la realidad, según se registra en el log, se ajusta al modelo y viceversa. Por ejemplo, puede haber un modelo de proceso que indique que las órdenes de compra mayores a cierto monto requieren dos cheques. El análisis del registro de eventos mostrará si esta regla se sigue o no. Otro ejemplo es la verificación del llamado principio de "cuatro ojos" que indica que actividades particulares no deben ser ejecutadas por una misma persona. Al escanear el registro de eventos utilizando un modelo que especifica estos requisitos, se pueden descubrir posibles casos de fraude. Por lo tanto, el *conformance checking* se puede utilizar para detectar, localizar, explicar las desviaciones, y para medir la gravedad de las mismas.

Enhancement. Se busca ampliar o mejorar un modelo de proceso existente, utilizando información sobre el proceso real registrado en algún registro de eventos. Mientras que el *conformance checking* mide la alineación entre el modelo y la realidad, el *enhacement* apunta a cambiar o extender el modelo existente. Un tipo de mejora es la reparación, es decir, la modificación del modelo para reflejar mejor la realidad. Por ejemplo, si dos actividades se modelan secuencialmente pero en realidad pueden ocurrir en cualquier orden, entonces el modelo puede corregirse para reflejar esto. Otro tipo de mejora es la extensión, es decir, agregar una nueva perspectiva al modelo de proceso.

En la siguiente sección, se mostrarán las diferentes formas en las cuales se pueden modelar procesos.

2.2.1. Modelado de procesos

Para trabajar en la disciplina de Minería de Procesos [5], es esencial contar con un modelo que describa el comportamiento de los procesos que están siendo estudiados. Este modelo debe describir el orden en que las actividades que componen un proceso son ejecutadas.

Existen distintas notaciones para representar estos modelos, desde notaciones de bajo nivel como *Transition Systems*, las cuales tienen limitaciones para representar la concurrencia, hasta notaciones de alto nivel como *Petri nets*, BPMN o *Process Trees*. Estas notaciones de alto nivel son equivalentes y existen métodos para traducirlas.

BPMN (con origen de sus siglas en inglés *Business Process Modeling Notation*) es una de las notaciones más utilizadas para el modelado de procesos de negocios. Es utilizada por varias herramientas y fue estandarizada por la OMG [12].

Un *process tree* es un modelo estructurado por bloques en forma de árbol, en el que sus hojas representan las actividades, y sus nodos internos representan los operadores que describen la relación y comportamiento entre éstas.

Petri net
Es la notación más utilizada en la Minería de Procesos, ya que fue la primera en poder modelar actividades concurrentes.

Una *Petri net* está compuesta por un conjunto de *places*, un conjunto de transiciones y un conjunto de arcos dirigidos llamados *flow relation*, los cuales relacionan los *places* con las transiciones.

Formalmente una Petri Net se define de la siguiente manera:

$$N = (P, T, F)$$

Donde P es un conjunto finito de *places*, T es un conjunto finito de transiciones donde $P \cap T = \emptyset$, y $F \subset (P \times T) \cup (T \times P)$ es un conjunto de arcos dirigido.

Las transiciones pueden estar etiquetadas por su actividad correspondiente. Si una transición no se corresponde con una actividad y por lo tanto no se corresponde a un evento del log, puede ser etiquetada como (tau), llamándose transición silenciosa.

Para modelar el comportamiento del proceso se utiliza una *marked Petri net*, la cual es una *Petri net* marcada por *Petri net*. Los *tokens* indican los *places* en los que se encuentra ejecutando el proceso. Para modelar el avance de estos *tokens* a través del modelo, se utiliza la siguiente regla llamada *firing rule*:

- Antes de ser ejecutada una transición, los *places* que la preceden deben contener un *token*.

- Luego de ser ejecutada la transición, los *tokens* que se encontraban en los *places* que la precedían, son consumidos y se generan nuevos *tokens* en los *places* que suceden a la transición.

De esta forma, como se puede apreciar en la figura 2.3, de acuerdo a los arcos salientes o entrantes de los places o las transiciones, se pueden definir siguientes conceptos:

- AND-split: una transición con más de un arco saliente. Indica que las transiciones en las que se separa el flujo, pueden ejecutarse en paralelo.
- AND-join: una transición con más de un arco entrante. Indica que para proseguir con la ejecución del proceso, todos los flujos en los que se separó el proceso en el AND-split, deben haber terminado.
- XOR-split: un place con más de un arco saliente. Indica que de los distintos flujos en los que se separa el proceso, debe ejecutarse solamente uno.
- XOR-join: un place con más de un arco entrante. Indica que para proseguir con la ejecución del proceso, el flujo que se comenzó a ejecutar a partir del XOR-split, debe haber terminado.

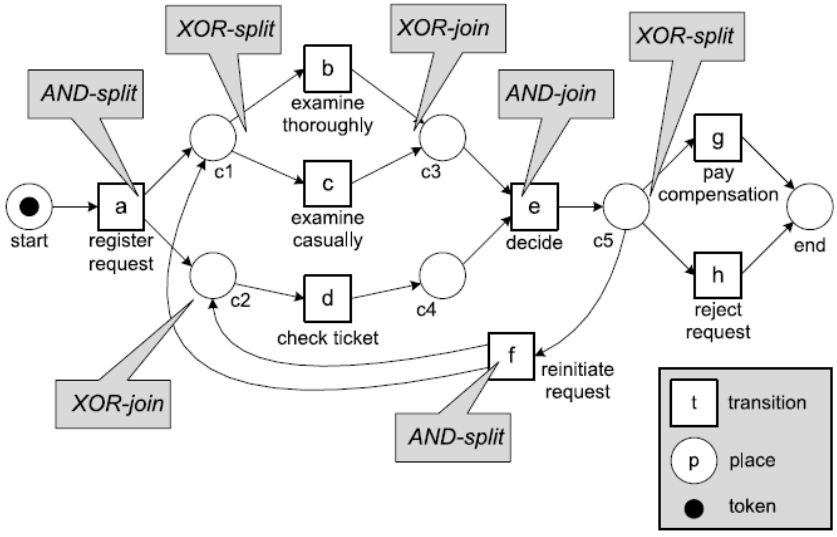

Figura 2.3: Ejemplo de una Petri net, indicando el modelado de sus XOR-split, XOR-join AND-split y AND-join, extraído de [5]

En el modelado de procesos interesa trabajar con un subgrupo de *Petri nets* llamado *workflow net*, ya que aseguran ciertas características necesarias para que el modelo sea válido.

Una *workflow net* es una *Petri net* con las siguientes características:

- Posee un único *place* de inicio llamado *start*.

- Posee un único *place* de finalización llamado *end*.
- Todos los nodos de la red pertenecen a un camino que conecta el *place* inicial *start*, con el final *end*.

Como se puede apreciar en la figura 2.3, el modelo de ejemplo cumple las condiciones de *workflow net*.

Este modelo describe el manejo de una solicitud de compensación de *tickets* de una aerolínea. Los clientes pueden solicitar la compensación por varias razones, como retrasos o cancelación de vuelos. El proceso comienza con la actividad *register request*, representada en la *Petri net* por una transición con ese nombre. Las transiciones representan actividades. Éstas están conectadas entre sí por places, los cuales modelan posibles estados del proceso. Cada transición puede ser ejecutada sólo cuando todos sus *places* de entrada contienen un *token*. Luego de ser ejecutada la transición *register request*, todos sus *places* de salida reciben un *token*. Cuando en una transición hay más de un arco saliente, se está modelando un **AND-split**, por lo que las actividades que se ejecutan luego de los *places* c1 y c2 pueden ejecutarse en paralelo. *Check ticket* modela una actividad administrativa, en el que se señala que el *ticket* está siendo analizado. En paralelo se puede seleccionar en c1 el tipo de examinación que debe hacerse al *ticket*. Según la complejidad del caso, debe examinarse el reclamo de manera detallada (transición *examine thoroughly*) o de manera sencilla (*examine casually*). De estas dos transiciones, sólo se debe ejecutar una. Cuando en un *place* hay más de un arco saliente, esto modela un **XOR-split**. Inmediatamente después, estas dos transiciones tienen arcos salientes sólo al *place* c3, el cual tiene dos arcos entrantes, esto modela un **XOR-join**. Luego de realizarse el chequeo de la solicitud, tanto de manera sencilla o detallada, se puede pasar a la transición *decide*. Esta transición, al tener más de un arco entrante, modela un **AND-join**. Para ejecutarse, deben haber *tokens* tanto en c3 y c4. Luego de ejecutada la transición *decide*, se puede pasar a las transiciones *pay compensation* (en la que el pago de compensación es aceptado), *reject request* (en la que el pago es denegado), o *reinitiate request* (en la que se inicia nuevamente el proceso).

2.2.2. Extracción de la información

Uno de los desafíos que se presentan al aplicar Minería de Procesos es la extracción de la información, muchas veces de múltiples fuentes, asegurando la calidad de los datos. [5]

Al momento de la extracción se debe determinar el alcance de la información obtenida: sólo los eventos relevantes para el proceso a ser analizado se incluirán en un registro o log de eventos. Luego de ser creado, es filtrado en un proceso iterativo. En base al resultado obtenido luego del filtrado de los datos, se aplicarán las diferentes técnicas de Minería de Procesos antes mencionadas: *discovery*, *conformance* y *enhacement*.

2.2.2.1. Logs de eventos

En la figura 2.4 se muestra una tabla de ejemplo con la información típica presente en un log de eventos utilizado para la Minería de Procesos. Dicha información se corresponde al proceso mostrado en la figura 2.5. La tabla muestra la información típica mostrada en un log de eventos a utilizar para aplicar Minería de Procesos. Se asume que un log de eventos se refiere a un mismo proceso. Cada evento del log se refiere a una instancia del proceso, o "caso". Cada solicitud se corresponde con un caso. Los eventos están relacionados a alguna actividad. En la tabla se puede ver que los eventos se refieren a actividades como por ejemplo: *register request*, *check ticket* y *reject*. Los eventos dentro del caso deben estar ordenados. Por ejemplo el evento 35654423 (la ejecución de la actividad *register request*, del caso 1) ocurre antes del evento 35654424 (la ejecución de una actividad examine *thoroughly* para el mismo caso). Sin establecer un orden en la información, no sería posible descubrir dependencias causales en los modelos de procesos. También se puede ver en la tabla, información adicional de cada evento. Por ejemplo, todos los eventos tienen un *timestamp*, es decir, información de fecha y hora como "30-12-2010:11.02". Los eventos de la tabla también se refieren a recursos, es decir, las personas que ejecutan las actividades. También los costos asociados a eventos. En el contexto de Minería de Procesos, esas propiedades son referidas como atributos.

Es común referirse a un evento por su nombre de actividad. Varios eventos podrían referirse al mismo nombre de actividad. Por ejemplo, si la misma actividad comienza dos veces en el mismo caso, en paralelo y una de ellas se completa, no se sabe cuál de las dos terminó primero. Este problema puede ser resuelto agregando información al log usando heurísticas, por ejemplo asumir que el orden es *first in first out*.

Los eventos de un caso, son representados en forma de trace, una secuencia única de eventos. Los casos contienen atributos. Cada uno tiene un atributo especial, obligatorio "trace". Un trace es una secuencia finita de eventos tal que cada evento aparece sólo una vez.

Los logs de eventos tienen un formato preestablecido y deben cumplir ciertas condiciones para poder aplicar las técnicas de Minería de Procesos. Las mismas se detallan a continuación:

- Cada log de eventos contiene información relacionada a un sólo proceso.
- Cada evento del log debe referir a una única instancia del proceso (caso).
- Los eventos pueden estar relacionados con alguna actividad. Un proceso puede verse como un conjunto de actividades.
- Los eventos de un caso deben ser ordenados.
- También puede mostrarse información adicional por evento, por ejemplo, el *timestamp*, útil para analizar las propiedades relacionadas con la *performance* o las personas (recursos) que ejecutan las actividades.

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	register request	Pete	50	...
	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	check ticket	Mike	100	...
	35654426	06-01-2011:11.18	decide	Sara	200	...
	35654427	07-01-2011:14.24	reject request	Pete	200	...
2	35654483	30-12-2010:11.32	register request	Mike	50	...
	35654485	30-12-2010:12.12	check ticket	Mike	100	...
	35654487	30-12-2010:14.16	examine casually	Pete	400	...
	35654488	05-01-2011:11.22	decide	Sara	200	...
	35654489	08-01-2011:12.05	pay compensation	Ellen	200	...
3	35654521	30-12-2010:14.32	register request	Pete	50	...
	35654522	30-12-2010:15.06	examine casually	Mike	400	...
	35654524	30-12-2010:16.34	check ticket	Ellen	100	...
	35654525	06-01-2011:09.18	decide	Sara	200	...
	35654526	06-01-2011:12.18	reinitiate request	Sara	200	...
	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	...
	35654530	08-01-2011:11.43	check ticket	Pete	100	...
	35654531	09-01-2011:09.55	decide	Sara	200	...
	35654533	15-01-2011:10.45	pay compensation	Ellen	200	...
4	35654641	06-01-2011:15.02	register request	Pete	50	...
	35654643	07-01-2011:12.06	check ticket	Mike	100	...
	35654644	08-01-2011:14.43	examine thoroughly	Sean	400	...
	35654645	09-01-2011:12.02	decide	Sara	200	...
	35654647	12-01-2011:15.44	reject request	Ellen	200	...
...	...	...	...	...	...	...

Figura 2.4: Fragmento de un log de eventos, cada línea se corresponde con un evento, extraído de [5]

2.2.2.2. Archivos XES

XES refiere a un formato estándar para guardar e intercambiar logs de eventos. En la figura 2.6, se muestra el metamodelo de XES expresado en términos de un diagrama UML. Un archivo XES consiste en una cantidad determinada de *traces* y cada *trace* contiene eventos.

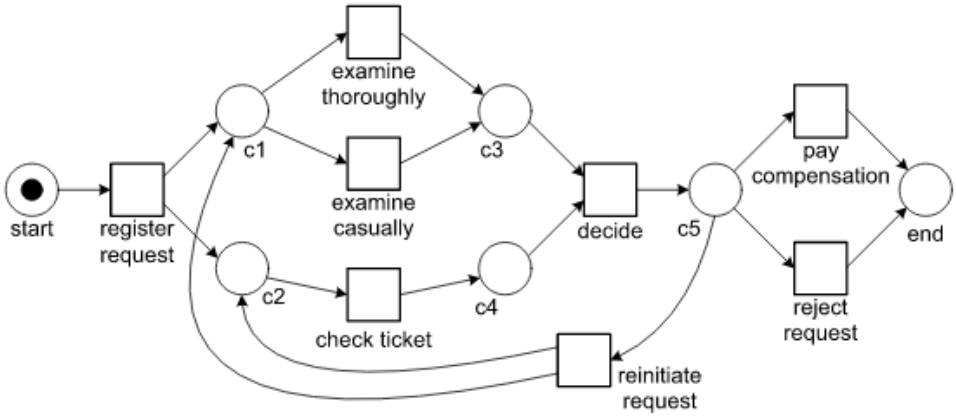

Figura 2.5: Petri net que modela el proceso de atención de solicitudes de compensación, extraído de [5]

Cada *trace* describe una lista secuencial de eventos correspondientes a un caso particular. Logs, *traces* y eventos tienen atributos, y los mismos pueden estar anidados. Hay cinco tipos principales: *String*, *Date*, *Int*, *Float* y *Boolean*. Las extensiones pueden definir nuevos atributos y un log debería declarar las extensiones usadas en él. Las extensiones brindan semántica a ciertos atributos particulares. Por ejemplo, la extensión time define un atributo *timestamp* de tipo *dateTime*. Los atributos globales son atributos declarados como obligatorios. Los atributos pueden estar a nivel de *traces* o de eventos, y pueden estar anidados. Los clasificadores de eventos son definidos para el log y asignan una etiqueta a cada evento. Cada clasificador es especificado por una lista de atributos. Si dos eventos contienen los mismos valores de esos atributos son considerados iguales para el clasificador. Pueden haber múltiples clasificadores en un mismo archivo XES.

En la figura 2.7 se muestra una serialización del log de eventos mostrado en la figura 2.4. Se pueden observar varios elementos explicados anteriormente, como por ejemplo las extensiones: *Concept*, *Time* y *Organizational*. Para cada extensión se determina un prefijo determinado, utilizado en los nombres de los atributos. Por ejemplo la extensión *time* define un atributo *timestamp*, y se guarda el *timestamp* de un evento utilizando el prefijo: *time:timestamp*.

También se especifican dos listas de atributos globales. Los *traces* tienen un atributo global: *concept:name* obligatorio para todos los *traces*. Los eventos tienen tres atributos globales: *time:timestamp*, *concept:name* y *org:resource* que son obligatorios para todos los eventos.

Tres clasificadores son definidos en el log mostrado en la figura 2.7. El clasificador *Activity* clasifica eventos en base al atributo *concept:name*. El clasificador *Resource* clasifica los eventos en base en el atributo *org:resource*. El clasificador *Both* clasifica los eventos en base a dos atributos: *concept:name* y *org:resource*. La extensión concepto define el atributo nom-

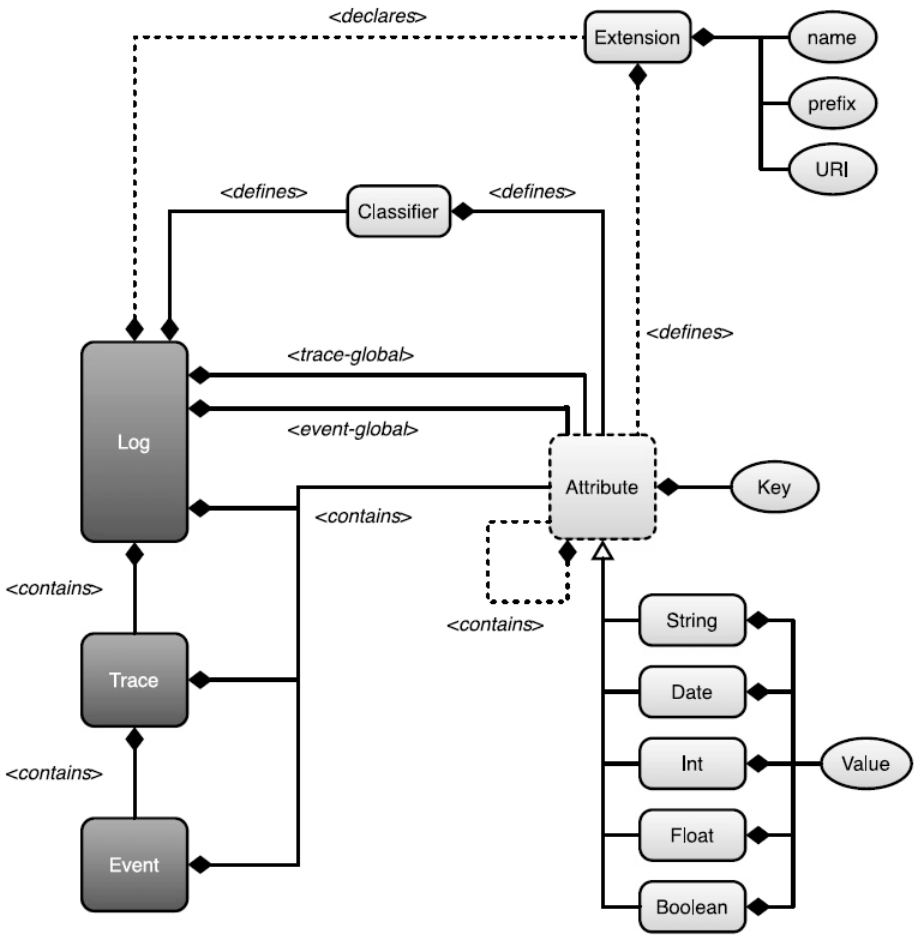

Figura 2.6: Metamodelo de un XES, extraído de [5]

bre para *traces* y eventos. En el ejemplo se puede ver que se usa *concept:name* tanto como atributo de *traces* como de eventos. En el caso de los *traces*, el atributo representa típicamente algún identificador del caso. Para los eventos, representa el nombre de la actividad. El concepto *extension* también define el atributo instancia para los eventos. Se utiliza para distinguir diferentes instancias de actividades en el mismo *trace*. La extensión ciclo de vida, define el atributo *transition* para los eventos. Algunos valores posibles para este atributo pueden ser: "*schedule*", "*start*", "*complete*", "*autoskip*" entre otros. La extensión *Organizacional* define tres atributos estándar para los eventos: *resource*, *role* y *group*. El atributo *resource* se refiere a los *resources* que son disparados o ejecutados en el evento. Los atributos rol y los grupos especifican las capacidades del *resource* y la posición del mismo en la organización.

```
<?xml version="1.0" encoding="UTF-8" ?>
   <extension name="Concept" prefix="concept" uri="http://.../concept.xesext"/>
   <extension name="Time" prefix="time" uri="http://.../time.xesext"/>
   <extension name="Organizational" prefix="org" uri="http://.../org.xesext"/>
   <global scope="trace">
      <string key="concept:name" value="name"/>
   </global>
   <global scope="event">
      <date key="time:timestamp" value="2010-12-17T20:01:02.229+02:00"/>
      <string key="concept:name" value="name"/>
      <string key="org:resource" value="resource"/>
   </global>
   <classifier name="Activity" keys="concept:name"/>
   <classifier name="Resource" keys="org:resource"/>
   <classifier name="Both" keys="concept:name org:resource"/>
   <trace>
      <string key="concept:name" value="1"/>
      <event>
         <string key="concept:name" value="register request"/>
         <string key="org:resource" value="Pete"/>
         <date key="time:timestamp" value="2010-12-30T11:02:00.000+01:00"/>
         <string key="Event_ID" value="35654423"/>
         <string key="Costs" value="50"/>
      </event>
      <event>
         <string key="concept:name" value="examine thoroughly"/>
         <string key="org:resource" value="Sue"/>
         <date key="time:timestamp" value="2010-12-31T10:06:00.000+01:00"/>
         <string key="Event_ID" value="35654424"/>
         <string key="Costs" value="400"/>
      </event>
      <event>
         <string key="concept:name" value="check ticket"/>
         <string key="org:resource" value="Mike"/>
         <date key="time:timestamp" value="2011-01-05T15:12:00.000+01:00"/>
         <string key="Event_ID" value="35654425"/>
         <string key="Costs" value="100"/>
      </event>
      <event>
         <string key="concept:name" value="decide"/>
         <string key="org:resource" value="Sara"/>
         <date key="time:timestamp" value="2011-01-06T11:18:00.000+01:00"/>
         <string key="Event_ID" value="35654426"/>
         <string key="Costs" value="200"/>
      </event>
      <event>
         <string key="concept:name" value="reject request"/>
         <string key="org:resource" value="Pete"/>
         <date key="time:timestamp" value="2011-01-07T14:24:00.000+01:00"/>
         <string key="Event_ID" value="35654427"/>
         <string key="Costs" value="200"/>
      </event>
   </trace>
   <trace>
      <string key="concept:name" value="2"/>
      <event>
         <string key="concept:name" value="register request"/>
         <string key="org:resource" value="Mike"/>
         <date key="time:timestamp" value="2010-12-30T11:32:00.000+01:00"/>
         <string key="Event_ID" value="35654483"/>
         <string key="Costs" value="50"/>
      </event>
      ...
   </trace>
   ...
</log>
```

Figura 2.7: Fragmento de un archivo XES, extraído de [5]

Por ejemplo, un evento ejecutado por el *manager* de ventas, puede tener el rol *manager* y el

grupo departamento de ventas asociado a él. La extensión *Time* define un atributo *timestamp* de tipo *dateTime*.

2.2.2.3. Construcción de logs

Al momento de crear el registro de eventos y para para asegurar la calidad y consistencia en los datos generados, algunas de las recomendaciones son:

- Seleccionar los eventos relevantes para el proceso.
- Los eventos deben estar correlacionados para formar instancias del proceso (casos)
- Los eventos deben ser ordenados usando información de *timestamp* o tener un orden explícito, por ejemplo, con una lista.
- Las referencias deberían ser estables, es decir, no relacionadas con el contexto, por ejemplo, ciertas herramientas generan diferentes logs dependiendo del idioma.
- Los valores de los atributos deberían ser lo más precisos posibles. Si un valor no tiene la precisión deseada, eso debería ser indicado explícitamente.
- Asegurar comparabilidad de los logs de eventos a través del tiempo y diferentes grupos de variantes de casos o procesos, es decir, usar los mismos criterios de registro. Si en algunos logs ciertos eventos no son registrados, se generarán diferencias que no existen en la realidad.

También es recomendable tener nombres de referencia y atributos con semánticas claras, es decir, que tengan el mismo significado para todas las personas involucradas en la creación y análisis de los datos de los eventos; así como tener una colección estructurada de nombres de referencias y atributos. Esto puede realizarse utilizando extensiones detalladas en la sección anterior. También es recomendable de ser posible, guardar información transaccional sobre el evento (*start*, *complete*, etc), realizar chequeos automatizados de consistencia y correctitud para asegurar la correctitud sintáctica del log de eventos, y asegurar la privacidad sin perder las correlaciones significativas.

2.2.3. Discovery

El modelado de procesos consiste en construir un modelo de proceso, a partir del comportamiento observado en logs de eventos. Se pueden generar modelos como por ejemplo, un BPMN, EPC, YAWL o una *Petri net*.

2.2.3.1. Algoritmo Alpha

El algoritmo Alpha (α-*algorithm*) genera una *Petri net* a partir de un log de eventos. Este algoritmo es muy popular debido a su sencillez y por haber sido uno de los primeros algoritmos en poder identificar adecuadamente procesos concurrentes.

Para generar el modelo, este algoritmo recorre el log en busca de ciertos patrones de comportamiento a partir de la relación entre eventos contiguos. Los patrones de comportamiento que pueden encontrarse son los de secuencia, XOR-split, XOR-join, AND-split y AND-join, tal como se menciona en la descripción de Petri net en la sección 2.2.1

A pesar de ser un algoritmo popular por ser una puerta de entrada a la disciplina de Minería de Procesos, en la práctica es poco utilizado debido a varias limitaciones que posee, las cuales serán mencionadas posteriormente.

2.2.3.2. Inductive Mining

Inductive Mining es una técnica de descubrimiento de procesos que a partir del log de eventos, genera un árbol de procesos, un modelo que por su estructura, garantiza poder ser transformado a una *Petri net* del tipo *workflow*, un tipo especial de *Petri net* que cuenta con un estado inicial y un estado final. Además todos los estados de esta red se encuentran en un camino desde el estado inicial al estado final. Hay distintas variantes de algoritmos de *Inductive Mining*, desde el más básico a variantes que incorporan el filtrado de comportamiento infrecuente.

El algoritmo básico de *Inductive Mining* separa iterativamente el log inicial en sublogs con eventos disjuntos, los cuales van siendo relacionados entre sí, por operadores que describen el comportamiento entre estos sublogs. El algoritmo finaliza cuando cada sublog contiene un solo evento.

Los operadores son los siguientes:

- Secuencia ($a \rightarrow b$): indica que luego del evento a ocurre el evento b.
- Mutua exclusión ($a \times b$): indica que sólo puede ocurrir el evento a o el evento b.
- Paralelo ($a||b$): indica que pueden ocurrir tanto a como b en cualquier orden.
- Loop: ($a \circlearrowleft b$) : indica que la secuencia de eventos ab puede repetirse indefinidamente.

Actualmente es una de las técnicas más utilizadas para la etapa de *discovery*, debido a su flexibilidad, garantías de comportamiento y escalabilidad.

2.2.3.3. Limitaciones de los algoritmos

Manejo de la frecuencia
Para modelar los procesos, es recomendable utilizar solo los comportamientos más frecuentes. Si un log posee secuencias de eventos que ocurren solo unas pocas veces, lo ideal es que no sean representadas en el modelo. Este comportamiento con muy baja frecuencia es llamado ruido, ya que no deja ver con claridad el comportamiento más frecuente y significativo del proceso.

Incompletitud
Se dice que un log es incompleto si el mismo contiene muy pocos eventos para permitir descubrir algunas de las estructuras de flujos de control subyacentes. Los modelos de proceso típicamente permiten muchos *traces* diferentes. Sin embargo, algunos de ellos pueden tener mucha menos probabilidad que otros, por lo cual, no es realista asumir que todos los *traces* posibles son presentados en el log. Por lo tanto, si un algoritmo asume que el único comportamiento válido es el visto en el log, seguramente tenga problemas de precisión excesiva, no pudiendo representar una gran cantidad de comportamientos válidos. Por otro lado, si la precisión del algoritmo es muy débil, genera un modelo que acepta comportamientos no válidos en la realidad.

Las versiones básicas del *Alpha* e *Inductive Mining* no trabajan con frecuencias, por lo tanto todo el comportamiento observado termina siendo modelado. Para solucionar este problema se utilizan versiones mejoradas de los algoritmos, las cuales ya tienen incorporado un filtrado de eventos poco frecuentes.

Ambos modelos tienen una noción de completitud un tanto débil. Por lo que sus modelos pueden llegar a validar algún *trace* que no sea correcto.

2.2.3.4. Criterios de calidad de los modelos

Para medir la calidad de un modelo, en general se utilizan los siguientes cuatro criterios:

- **Fitness.** El modelo permite reproducir el comportamiento visto en el log, y permite que los traces en el log puedan ser reproducidos por el modelo desde el comienzo hasta el final.
- **Simplicidad.** El modelo es capaz de reflejar el comportamiento visto en el log de la forma más simple posible.
- **Precisión.** El modelo no permite comportamientos demasiado diferentes al observado en el log.
- **Generalización.** El modelo debe permitir cierta generalización y no restringir el comportamiento sólo a lo observado en el log.

Lo ideal es que estos cuatro criterios estén balanceados. En los algoritmos de Minería de Procesos, es deseable que el modelo generado no generalice y sólo permita el comportamiento registrado en el log, pero a su vez permita modelar algo de comportamiento extra, aunque no haya indicaciones en el log que sugieran ese comportamiento adicional.

Según estos cuatro criterios, el algoritmo *alpha* tiene un problema notorio de *fitness*, ya que existen casos en los que genera un modelo incorrecto, por ejemplo no puede modelar *loops* de actividades de largo menor a 3. Si el log contiene estos loops, el algoritmo alpha genera modelos que no son workflow net, ya que poseen actividades desconectadas. También posee problemas de simplicidad, ya que muchas veces genera modelos con *places* implícitos los cuales son places redundantes que complejizan el modelo sin agregarles significado. *Induc-*

tive Mining garantiza el correcto *fitness* debido a que por su construcción produce modelos consistentes que pueden reproducir el log completo. También debido a que este algoritmo se basa en una estructura de árbol y sus actividades están estructuradas en bloque, tiende a generar un modelo simple y con una buena generalización. Donde puede llegar a presentar algún problema, es en la falta de precisión, ya que en algunos casos puede modelar comportamiento no válido en la realidad. Por sus buenos resultados, *Inductive Mining* junto con el *Heuristic mining*, *Fuzzy mining* y *Genetic Process Mining*, son las técnicas de *discovery* que se utilizan actualmente en la práctica.

2.2.4. Conformance checking

El *conformance checking* es una técnica que reproduce el log sobre el modelo, relacionando los eventos del log con las actividades del modelo, para poder encontrar puntos en común y diferencias. Los resultados del *conformance checking* son utilizados con dos objetivos diferentes.

- Descriptivo: En este caso sabemos que los logs tienen un comportamiento adecuado y se busca evaluar que tan bien el modelo describe el comportamiento de los logs. Si el valor del *fitness* es bajo, indica que hay que mejorar el modelo para que represente mejor la realidad de los logs.
- Normativo: En este caso sabemos que el modelo representa el comportamiento adecuado. Un valor bajo de *fitness* indica que en el log hay comportamiento que no cumple las normas, teniéndose que evaluar si este comportamiento es riesgoso y debería de evitarse.

El método más utilizado para *conformance checking* es Alineación.

Alineación (*alignments*)
Este método consiste en comparar los *traces* del log con un camino entre el estado inicial y final en el modelo. Para esto se alinean los eventos de cada *trace* del log a un camino en el modelo. Si se logran alinear completamente, se obtiene un valor perfecto de *fitness*. Si el *fitness* no puede alinearse completamente sobre un recorrido en el modelo, se realizan movimientos, tanto en el log como en el modelo para hacer coincidir el *trace* con el recorrido en el modelo. Cada uno de estos movimientos produce una penalización en el valor final de *fitness* del *trace*. Es posible asignar un valor de penalización diferente a cada evento, dependiendo de la importancia que este tenga. Por defecto todas las penalizaciones tienen valor 1.

El valor de *fitness* de cada *trace* se encuentra en el rango 0..1, donde 0 corresponde al peor caso de alineación posible y 1 corresponde a una alineación perfecta sin penalizaciones.

Ejemplo: dado el siguiente trace y el siguiente modelo N2, de la figura 2.8, se obtienen las siguientes 3 alineaciones óptimas que se muestran en la figura 2.9

$$o = a, d, b, e, h$$

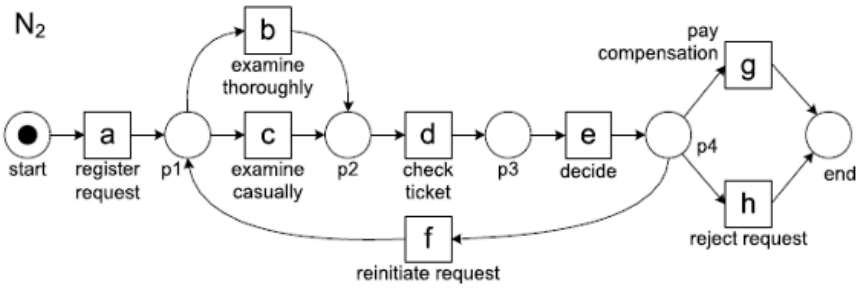

Figura 2.8: Ejemplo de *Petri net* para calcular valor de *fitness*, extraído de [5]

$$\gamma_{2a} = \begin{array}{|c|c|c|c|c|c|} a & \gg & d & b & e & h \\ a & b & d & \gg & e & h \end{array} \qquad \gamma_{2b} = \begin{array}{|c|c|c|c|c|c|} a & \gg & d & b & e & h \\ a & c & d & \gg & e & h \end{array} \qquad \gamma_{2c} = \begin{array}{|c|c|c|c|c|c|} a & d & b & \gg & e & h \\ a & \gg & b & d & e & h \end{array}$$

Figura 2.9: Ejemplo de alineaciones, extraído de [5]

Donde la fila superior corresponde a movimientos del log y la fila inferior a movimientos del modelo. Si un movimiento del modelo no puede ser igualado por un movimiento del log, en la línea superior aparece el símbolo $>>$, que corresponde a una desalineación en el log. Cuando el símbolo $>>$ aparece en la línea inferior, corresponde a una desalienación en el modelo.

Definiendo movimiento como un par (x, y) donde el primer elemento pertenece al log y el segundo elemento pertenece al modelo, los movimientos válidos (x, y) son los siguientes:

- $x = y$, y es una transición (movimiento sincrónico)
- $x =>>$, y es una transición (movimiento visible del modelo)
- $x =>>$, y es una transición silenciosa del modelo (movimiento invisible del modelo)
- $x \neq>>$, $y =>>$ (movimiento del log)

Una alineación es una secuencia de movimientos válidos en los que al remover los símbolos de desalienación $>>$, la línea superior corresponde a un trace en el log y la línea inferior corresponde a una secuencia del modelo desde el estado inicial al final.

Cálculo de *fitness* a partir de alineamientos Para calcular este valor hay que obtener el valor de la alineación óptima y el de la peor alineación posible.

La peor alineación es una en la que no hay movimientos sincrónicos. Nótese que siempre se

puede crear una alineación en la que todos los eventos del *trace* son convertidos a movimientos del log, y uno de los caminos más cortos de un estado inicial a uno final del modelo, es agregado como una secuencia de movimientos del modelo.

Esta alineación está conformada por movimientos válidos y no puede existir una alineación óptima con mayor costo que ésta.

En el *trace* de ejemplo que se está analizando, una "peor alineación" podría ser la siguiente:

$$a\ b\ d\ e\ h\ >>\ >>\ >>\ >>\ >>$$
$$>>\ >>\ >>\ >>\ >>\ a\ c\ d\ e\ h$$

Definiendo la alineación como P la de peor costo posible, y siendo O la alineación óptima, se puede definir el valor de *fitness* de un *trace* t y un modelo M como:

$$fitness(t, M) = 1 - O/P$$

Asumiendo que existe un camino desde un estado inicial a un estado final, siempre se puede obtener un valor de *fitness* entre 0 y 1.

Como se vio anteriormente, la alineación óptima de t y M tiene 2 movimientos asincrónicos. Asignando el valor 1 a cada movimiento asincrónico, se obtienen los valores O=2 y P=10 Por lo tanto:

$$fitness(t, M) = 1 - (2/10) = 0,8$$

El valor de *fitness* se puede generalizar para un log L como el promedio de los valores de *fitness* de todos sus *traces* de la siguiente manera:

$$fitness(L, M) = 1 - \frac{\sum O\ traces\ en\ L}{\sum P\ traces\ en\ L}$$

Además del valor de *fitness*, con la alineación se pueden obtener otros datos interesantes, como la identificación de eventos con ciertas características, por ejemplo los más frecuentes o los que presentan más problemas de alineación.

3 Análisis del problema

El problema a resolver en el presente trabajo es la detección de ataques maliciosos a sitios web utilizando técnicas de Minería de Procesos, con el objetivo de detectar desviaciones del normal comportamiento de un usuario sobre un sitio web. Estas desviaciones pueden ser producto de ataques o intentos de ataques sobre el sitio. En base al análisis de un trabajo previo relacionado, se adaptan algunas de sus particularidades al contexto de uso de un *Web Application Firewall*.

3.1. Trabajo relacionado

El presente proyecto parte de un análisis previo, presentado por la Universidad de Zaragoza, "Minería de Procesos e Ingeniería Guiada por Modelos para mejorar la Seguridad de Sistemas de Información Web" [6]

En ese trabajo, se propone utilizar las técnicas de Minería de Procesos e Ingeniería Guiada por Modelos para mejorar la Seguridad de Sistemas de Información Web, de forma tal que no dependa de la tecnología utilizada por las aplicaciones. El método consiste en detectar patrones de ataque identificando desviaciones del comportamiento conocido del sistema, que es representado por un "modelo normativo".

Resumidamente, el método planteado consta de los siguientes pasos:

- Paso 1: Especificación del comportamiento del sistema por medio de UML.
- Paso 2: Generación automática de un modelo formal a partir de la especificación basada en UML el cual se denominará "modelo normativo".
- Paso 3: Control y monitoreo del sistema de información web para obtener logs de datos que representan su comportamiento operativo o real.
- Paso 4: Pre-procesamiento de logs del sistema para obtener logs de eventos, que pueden

ser analizados usando técnicas de Minería de Procesos.
- Paso 5: Identificar desviaciones entre el modelo normativo y el comportamiento operativo por medio de diferentes técnicas de Minería de Procesos.

La Figura 3.1 muestra los pasos planteados.

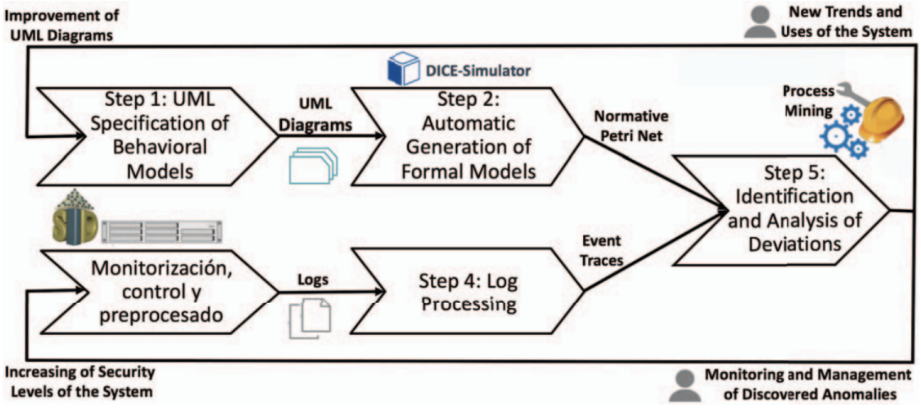

Figura 3.1: Visión general del método propuesto, extraído de [6]

3.2. Adaptación del trabajo relacionado

En esta sección se muestran las posibles adaptaciones a los pasos mencionados anteriormente, haciendo uso de un WAF para extraer información de los sistemas web.

Paso 1: Especificación del comportamiento esperado del sistema
En el trabajo descrito en la sección anterior, la especificación del comportamiento esperado del sistema se realiza a través de un diagrama UML. Otra forma de obtener esa información podría ser a través de ejecuciones válidas sobre el sistema. Las mismas, pueden ser capturadas utilizando un WAF como ModSecurity que genera logs por cada petición HTTP que se realiza al sitio en el cual se configure. A partir de logs generados por ejecuciones válidas en el sitio, es posible generar un modelo normativo. ModSecurity se configura sobre un servidor Apache, que a su vez puede apuntar como un proxy reverso al sitio web en cuestión y de esta forma, navegando sobre el sitio, se puede obtener la información necesaria. En otras secciones se presentarán más detalles sobre configuraciones específicas, pero en principio, estas configuraciones iniciales permiten no depender demasiado del caso concreto que se quiera estudiar.

Paso 2: Generación del modelo normativo
Para la obtención del modelo normativo, se pueden utilizar los logs generados a partir de comportamiento válido, como se especificó en el punto anterior.

Con herramientas de software es posible transformar los archivos generados por ModSecurity durante el uso del sistema por parte de los usuarios, a un solo log en formato XES. Este formato es aceptado por aplicaciones, como por ejemplo el *framework* ProM, que implementan algoritmos de Minería de Procesos. A partir del log XES generado, se puede obtener una *Petri net* por medio de los algoritmos de *discovery* de este tipo de aplicaciones.

Cabe aclarar que, dado que los logs iniciales son producto del recorrido de un sitio web, las diferentes actividades de la *Petri net* se corresponden con las URLs del sitio a las cuales los usuarios fueron accediendo durante el uso (adecuado) del mismo. Por este motivo, es recomendable filtrar contenidos estáticos como pueden ser solicitudes de archivos css, *javascript*, imágenes, etc, ya que no aportan información de valor en lo que se refiere al comportamiento del usuario en el sitio.

Paso 3: Obtención de logs que representen el comportamiento real del sistema
Con la infraestructura establecida, como se explicó en el paso 1, es posible recolectar información generada por ModSecurity, tanto a partir de navegaciones que realicen usuarios bien intencionados sobre el sitio, como información generada automáticamente por recorridas que pertenezcan a posibles ataques, o intentos de ataques, utilizando alguna herramienta para este fin. Es importante mencionar que cada archivo de logs contiene, entre otras cosas, información básica del pedido HTTP y posibles reglas que el WAF haya aplicado en el pedido correspondiente al archivo de log. Estas reglas son incorporadas como información en todos los logs generados en el sistema.

Paso 4: Pre-procesamiento de logs
Uno de los desafíos que se presenta es debido al nivel de granularidad de los datos, ya que las aplicaciones generan logs de muy bajo nivel de granularidad. La cantidad de eventos que se obtiene puede llegar a ser muy grande, debido a la complejidad de los sitios web y a que muchas de las URLs que se obtienen corresponden a elementos que no son relevantes para el estudio de vulnerabilidades. En este contexto, la principal estrategia para facilitar la interpretación de un log es aplicar una etapa de filtrado de los datos.

Por esta razón se recomienda aplicar filtros para excluir ciertas URLs. Por ejemplo, las URLs que terminan con ciertas extensiones, entre ellas .css, .js, .jpg. También podría ser recomendable filtrar reglas de ModSecurity que en el sitio web que se esté estudiando no impliquen una vulnerabilidad.

Otro punto a considerar, es que los logs disponibles para el estudio no contienen campos que se refieran a sesiones de usuarios. Para ello se pueden considerar diferentes heurísticas para obtener esa información a partir de los logs utilizando los valores de IP, el tiempo transcurrido o el navegador utilizado. Si el sistema permite más de un usuario, también se podría utilizar ese dato para identificar el usuario de la aplicación. Una opción consiste en considerar que si los pedidos HTTP que se registran en ModSecurity provienen de una misma IP, se trata de un

mismo usuario interactuando con el sistema en una sesión de trabajo.

Paso 5: Identificación de desviaciones entre el modelo normativo y el comportamiento del sistema utilizando técnicas de Minería de Procesos
Para identificar desviaciones entre el modelo normativo y un comportamiento dado, se aplica *conformance checking*. Para que el mismo obtenga resultados útiles, es necesario que el modelo sea estructurado.

Sin embargo, un modelo generado a partir del recorrido de un sitio web, es desestructurado. Los procesos que generan este tipo de modelos son llamados procesos "*spaghetti*" debido a su apariencia entrelazada con muchos arcos.

La figura 3.2, muestra la forma general de un modelo al que se llegaría aplicando el algoritmo *Alpha*, sobre un conjunto de logs obtenidos de un sitio, utilizando el WAF ModSecurity. Aunque se aplicaron filtros básicos, como los mencionados en el paso 4, el modelo generado es del tipo "*spaghetti*". Esto hace necesario un mayor esfuerzo en la etapa de preprocesamiento, para lograr alguna utilidad de los modelos obtenidos.

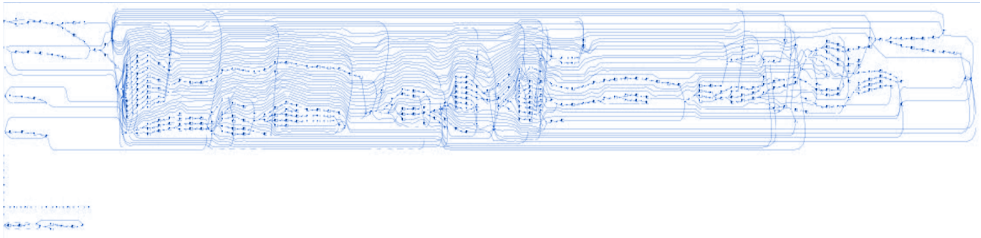

Figura 3.2: Modelo generado con el algoritmo *Alpha*, aplicado sobre navegaciones bien intencionadas sobre un sitio.

Esto se da por el formato que tiene un sitio web, y por la forma en que se genera el modelo: tomando como actividades las distintas URLs a las que acceden los usuarios en el sitio, las cuales pueden ser muy numerosas, a pesar de que se filtre el contenido estático. También afecta la estructura dada de los sitios web en general, donde se permite acceder a las mismas URLs desde diferentes puntos del sitio. Esto genera que las ejecuciones de las actividades del modelo (las URLs) no respeten un único orden fijo preestablecido o un conjunto reducido de combinaciones de actividades distintas, sino muchas combinaciones pueden darse para una misma actividad o URL, lo cual complejiza el modelo.

Por esta razón, se opta por modificar el enfoque del análisis de los procesos que existen en un sitio web, para poder aplicar las técnicas de Minería de Procesos.

3.3. Transformación de procesos desestructurados

La forma de trabajar con procesos desestructurados, es transformarlos en procesos más estructurados y simples, los cuales son llamados procesos *lasagna*. [5]

En los procesos estructurados, las actividades se repiten y tienen bien definida una entrada y una salida. Este comportamiento garantiza que se puedan utilizar todas las técnicas de Minería de Procesos. En cambio, en los procesos *spaghetti*, es difícil definir cuáles son las pre y post condiciones de cada actividad, haciendo que sólo algunas de las técnicas de Minería de Procesos puedan ser utilizadas.

No existe una definición formal que pueda caracterizar y diferenciar procesos *spaghetti* de *lasagna*, aunque existen criterios informales para identificarlos.

En casos extremos, tales como se ve en la figura 3.2, es fácil identificarlos debido a su aspecto.

En otros casos, aunque no sea tan evidente su forma de *spaghetti*, se puede detectar por el gran número de actividades que posee, y por el hecho de que la mayoría de los traces que se utilizan para generar el modelo, se corresponden con un camino diferente. Esto sugiere que el log está muy distante de ser completo.

En el caso de procesos *lasagna*, un criterio muy utilizado para detectarlos, es el de verificar que el modelo tenga un *fitness* de al menos 0,8 con respecto al log con el que fue generado, es decir que más del 80 % de los eventos del log ocurran de forma sincrónica con el modelo.

La manera más sencilla de transformar los procesos *spaghetti* en *lasagna*, es simplificándolos mediante el filtrado de eventos que no aporten información relevante al comportamiento del proceso, tal como se mencionó anteriormente.

Otra manera más avanzada de afrontar este problema, es mediante la agrupación de actividades que tengan algún patrón de comportamiento en común. Todas estas actividades pueden agruparse en una sola actividad abstracta que se conecte con el resto de actividades. De esta forma, el modelo representa un comportamiento más simple y estructurado.

El artículo *Abstractions in Process Mining: A Taxonomy of Patterns* [13] propone diferentes técnicas de abstracción, como identificar las actividades involucradas en un *loop* y agruparlas en una sola clase abstracta. Otra opción es la de identificar subprocesos o actividades con una funcionalidad en común y agruparlas también en una actividad abstracta.

Procesos críticos
Si bien aplicando los filtros anteriormente mencionados, es posible reducir el tamaño del modelo, aún así con los algoritmos de Minería de Procesos no se logra obtener resultados representativos de la realidad. Tal como se explicó anteriormente, se está analizando un sitio web en el cual para cada actividad se tiene una gran cantidad de caminos distintos que la involucran, con lo cual se dificulta extraer patrones concretos de comportamiento de los usuarios. Otro problema es la diferencia en el largo de los *traces*. Algunos usuarios podrían

navegar en el sitio durante un largo período de tiempo y otros por un período menor, lo cual generaría *traces* con largos muy diferentes. Esta posible disparidad en cuanto a la información, es un problema para este tipo de técnicas, ya que la información obtenida no permite obtener modelos que sean de utilidad para la etapa de *conformance checking*.

Por estas razones, se opta por seleccionar ciertas actividades en concreto y no el sitio en su totalidad. Este acercamiento consiste en enfocar el estudio en ciertas partes específicas de un sitio y observar qué ocurre en sus cercanías o a su alrededor. Es decir, para determinada actividad que se desee estudiar, contemplar dicha actividad y considerar algunos eventos anteriores y algunos posteriores.

Por ejemplo, si se tiene un sitio de comercio electrónico, se pueden elegir actividades como por ejemplo, la acción de agregar un elemento al carrito o de consultar un producto, en vez de tratar examinar el comportamiento de los usuarios en todo el sitio y en todas las funcionalidades.

Este acercamiento presenta dos ventajas fundamentales. La primera es que los *traces* pasan a ser de tamaño similar. Esto ayuda a un mejor desempeño de los algoritmos de descubrimiento. No se genera un modelo del sitio completo, sino modelos de funcionalidades puntuales. En el ejemplo mencionado, se obtendría el modelo normativo de la acción “agregar elemento al carrito”. La segunda es que hace posible contar con mayor cantidad de *traces* para cada funcionalidad, puesto que un mismo usuario puede ejecutarla varias veces y se considera un nuevo *trace* por cada una de estas ejecuciones. Contar con más información también es de ayuda para los algoritmos.

La elección de estas actividades o procesos a analizar, puede realizarse en base a distintos criterios. Uno de ellos podría ser en base a su criticidad, por ejemplo debido a la implicancia de transacciones de dinero o acceso a datos personales, como pueden ser el inicio de sesión en el sitio o algún proceso de compra. También se podrían elegir actividades más frecuentes o simplemente, las que resulten ser de mayor interés para investigar. De esta forma, se reduce el tamaño de los logs y de los modelos, lo cual permite la correcta aplicación de las técnicas de Minería de Procesos.

A efectos de estudiar las posibles vulnerabilidades de los sitios web, y de identificar comportamiento malicioso, se agrega la información de las reglas activadas por ModSecurity. Para que se reflejen las reglas activadas en el modelo normativo, las mismas pueden ser incorporadas como parte de los logs como eventos. Estos eventos se pueden diferenciar de los eventos relacionados a actividades, estableciendo un tipo de evento particular asociado únicamente a reglas. Cada vez que se activa una regla, se agrega un evento de tipo “regla” a continuación del evento que generó la activación. Esto genera en el modelo actividades de tipo regla, las cuales aparecerán solamente cuando alguna de las reglas sean activadas, a continuación de la actividad (URL) que generó la activación de la misma por parte del WAF.

Esta es una forma de poder agregar la información brindada por la herramienta ModSecurity al modelo.

En el siguiente capítulo se detalla cómo se pueden utilizar los logs generados por el WAF ModSecurity en un sitio web y, mediante la aplicación de técnicas de Minería de Procesos, detectar ataques al sitio.

4 Solución

En base al análisis del problema expuesto en la sección previa, a continuación se plantea una posible solución al problema planteado. Los pasos a seguir son los siguientes:

Paso 1: Obtención y generación de logs a partir de un uso válido sistema con ModSecurity.
Paso 2: Procesamiento de logs. Filtrado, agrupación e identificación de zonas críticas.
Paso 3: Aplicación del algoritmo *Inductive Mining* para generar el modelo de comportamiento del sistema.
Paso 4: Generación de logs de uso habitual del sistema y de ataques.
Paso 5: Procesamiento de logs de uso del sistema, ídem a paso 2.
Paso 6: *Conformance checking* de los logs contra el modelo normativo: Identificación y análisis de desviaciones.

La figura 4.1 muestra la relación entre los pasos planteados.

A continuación se pasará a detallar cada uno de los pasos de la solución planteada.

4.1. Obtención y generación de logs

Tal como se mencionó en la sección anterior, es necesaria la generación de logs, tanto para poder analizar el comportamiento esperado del sitio web a estudiar, como para obtener datos con los cuales comparar si el comportamiento real del sistema es el esperado o no, lo cual podría implicar la presencia de comportamiento malintencionado.

Para ello, se presenta la siguiente sección donde se especifica al detalle cómo se obtienen los logs y se presenta una herramienta, creada en el marco de este estudio, con el fin de procesar los logs, es decir, extraer la información relevante para el estudio, por medio de la aplicación

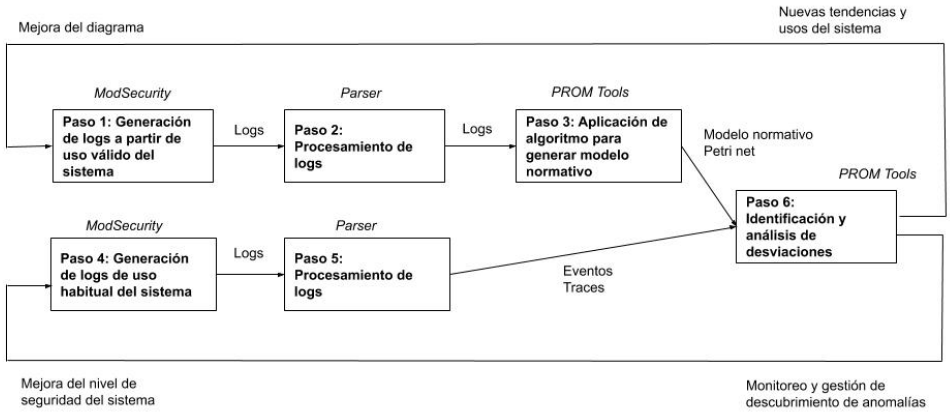

Figura 4.1: Pasos de la solución planteada.

de filtros, agrupamientos y otras técnicas que se detallarán a continuación.

El conjunto de logs utilizados para el estudio, es el generado por ModSecurity en su modo concurrente, es decir, un archivo de log por cada petición HTTP. En la sección 5, se mostrarán detalles y un ejemplo concreto de cómo se logra disponer de esta información. El propósito es describir el proceso de transformación de los logs generados por ModSecurity, a un archivo XES. El mismo es necesario para aplicar las técnicas de Minería de Procesos.

A modo de ejemplo, suponiendo un fragmento de de un archivo de log de ModSecurity, correspondiente a una petición HTTP, con esta información:

```
--4c358d0e-A--
[19/Jul/2020:17:20:08+0000] XxSAxawRAAQAAAAM9LEAAABL
   172.17.0.1 55218 172.17.0.4 80
--4c358d0e-B--
POST /catalogsearch/result/?q=aa
Host: modsecurity-magento
--4c358d0e-F--
HTTP/1.1 200 OK
--4c358d0e-H--
```

Donde la sección A muestra la IP del cliente, la B el método y el recurso asociado a la petición, F el encabezado de la respuesta y H las reglas activadas por ModSecurity (si las hubiese).

El evento correspondiente en el archivo XES es el siguiente:

```
<event>
    <string key="org:resource" value="172.17.0.1"/>
    <date key="time:timestamp" value="2020-07-19T17:20:08"/>
    <string key="concept:name" value="
       POST:catalogSearchResult"/>
    <string key="debug:originalUri" value="POST:/
       catalogsearch/result/?q=aaa"/>
    <string key="idRules" value=""/>
</event>
```

El atributo "*org:resource*" corresponde a la IP del usuario que realizó la petición. El atributo "*time:timestamp*" contiene la fecha y hora en que se realizó la solicitud. El atributo "*concept:name*" representa el recurso web específico solicitado por el usuario, lo que intenta modelar el comportamiento del mismo sobre el sitio. Su valor puede corresponderse con la URL original de la petición, o diferir por causa de una transformación de agrupamiento de eventos, que se explicará más adelante. El atributo "*debug:originalUri*" se corresponde con la URL original de la solicitud. Por último, "*idRules*" contiene un texto de todas las reglas que aplican en la petición HTTP, separadas por coma. En el ejemplo, las sección H se encuentra vacía, dado que en esta sección se generan logs con comportamiento válido que no generan la activación de reglas.

Cabe aclarar que cada evento se encuentra contenido dentro de un elemento "*trace*" en la globalidad del XES. La premisa es que cada *trace* se corresponde con una ejecución de un usuario, que se corresponde con una IP en particular, de forma de distinguir diferentes usuarios. Igualmente pueden haber varios *traces* correspondientes a una misma IP de un mismo

usuario.

Para automatizar todas estas transformaciones, fue necesario desarrollar una herramienta propia. Concretamente, ParserModSecurity [14], una aplicación de consola desarrollada en Java que se encarga de todos los procesamientos de estos archivos. Sus detalles se presentan más adelante en esta sección.

Reglas como eventos
En caso que un evento dispare una regla de ModSecurity, se agrega un evento que representa dicha ocurrencia. Para ejemplificar, continuando con el ejemplo anterior de un evento en un log XES, se tendría el valor de las reglas disparadas en el atributo "*idRule*":

```
<event>
    <string key="org:resource" value="172.17.0.1"/>
    <date key="time:timestamp" value="2020-07-19T17:20:08"/>
    <string key="concept:name" value="
       POST:catalogSearchResult"/>
    <string key="debug:originalUri" value="POST:/
       catalogsearch/result/?q=aaa"/>
    <string key="idRules" value="1,2"/>
</event>
```

Se agregan los siguientes dos eventos en forma consecutiva:

```
<event>
        <string key="org:resource" value="172.17.0.1"/>

        <date key="time:timestamp" value="2020-07-19T14:27:38
           .038-0300"/>
        <string key="concept:name" value="RULE:Id=1"/>
        <string key="debug:originalUri" value="RULE:/
           catalogsearch/result/?q=aaa"/>
        <string key="idRules" value=""/>
</event>
<event>
        <string key="org:resource" value="172.17.0.1"/>

        <date key="time:timestamp" value="2020-07-19T14:27:38
           .038-0300"/>
        <string key="concept:name" value="RULE:Id=2"/>
        <string key="debug:originalUri" value="RULE:/
           catalogsearch/result/?q=aaa"/>
```

```
        <string key="idRules" value=""/>
</event>
```

Esto tiene dos razones fundamentales:

- Poder visualizar la ocurrencia de estas reglas en los modelos generados.
- Penalizar de forma intencional la ocurrencia de una regla a la hora de ejecutar un *conformance* contra un modelo normativo. Es de esperar que un modelo generado con información de usuarios bien intencionados no contenga este tipo de actividades regla, por lo que un log que sí los contenga, obtendría valores de *fitness* más bajos en estos casos.

4.2. Procesamiento de logs y zonas críticas

Los log generados anteriormente, por medio del WAF ModSecurity, deben ser pre procesados para poder aplicar correctamente las técnicas de Minería de Procesos, ya que no toda la información generada es relevante.

Para esto se utilizaron las siguientes técnicas de preprocesamiento explicadas en la sección 3.3:

- Filtrado de registros que no son de interés.
- Unificación de registros considerándolos cajas negras. Este agrupamiento se puede realizar:
  - Detectando patrones de comportamiento (ej. loops) que luego se sustituyen por una actividad simple.
  - Tomando cierto conjunto de páginas como un único tipo de registro.
- Utilizar sólo ciertas partes de la web que sean consideradas críticas.

Filtrado de elementos estáticos
Se opta por filtrar elementos estáticos de la página tales como las imágenes, hojas de estilo css y archivos *javascript*, ya que no proporcionan información acerca del comportamiento del usuario en el sitio y están presentes en el estudio de un sitio web. Dado que el WAF captura todas las peticiones al servidor, las mismas incluyen este tipo de archivos cuya finalidad es mejorar la experiencia de usuario en cuanto a la usabilidad del sistema, pero no aportan valor a la hora de identificar procesos que modelan el uso del sistema.

Filtrado de resultados intermedios
Puede ser de interés filtrar ciertos pedidos “intermedios” al servidor capturados por ModSecurity, que no aporten valor al análisis. Un ejemplo de este caso son los campos de búsqueda, muy comunes en los sitios web que realizan pedidos AJAX [15]. El comportamiento es el siguiente: se tiene un campo de búsqueda, el cual permite realizar búsquedas en el sitio. A

medida que el usuario escribe en el campo, sea realizan llamadas AJAX donde por cada letra que se ingresa se hace un *request* al sitio, para ofrecer sugerencias al usuario. Luego, el usuario al seleccionar el botón enter, ingresa el texto completo de su búsqueda. Como cada *request* va a representar una actividad en el modelo, se generan tantos eventos como *request*. Considerar eventos y actividades diferentes por cada letra ingresada, no es recomendable, puesto que complejiza el modelo y no agrega información en cuanto al comportamiento del usuario para detectar vulnerabilidades. Sólo se generan varios *requests* para una simple búsqueda por palabra, por la tecnología que se utiliza para brindar sugerencias al usuario. Estos eventos pueden colapsarse en uno solo, registrando solamente la palabra completa final ingresada por el usuario y asociada a una única actividad de tipo búsqueda.

Filtrado de reglas
Muchas veces es de interés filtrar reglas de ModSecurity que se aplicaron a un determinado *dataset* de datos. Por ejemplo, al ingresar directamente por IP a un sitio web, al utilizar un ambiente de prueba se activa la regla 920350 Esta regla chequea que el *host header* no sea una dirección IP. Esto no es una violación al protocolo HTTP RFC, pero es un indicativo de un posible acceso automatizado. Muchos gusanos informáticos, es decir, *malware* que se replica para propagarse en otras computadoras, lo hacen escaneando bloques de direcciones IP.

Agrupamiento de eventos
Las actividades se están identificando a partir de URL, pero algunas tienen parámetros con identificadores, los cuales si no son procesadas previamente, se verán reflejadas en el modelo como actividades diferentes cuando, a efectos de este estudio, se corresponden con una misma actividad. Por ejemplo, las siguientes urls se generan a partir de la visualización de una opinión de dos productos diferentes en un sitio web:

/opinion/producto/id/699/
/opinion/producto/id/700/

Se generan dos actividades diferentes, pero la actividad de interés a modelar es la de consulta de la opinión de un producto, con lo cual, se decidió agrupar en una única actividad de visualización de la opinión de un producto:

URLs originales	URLs agrupadas a un único evento
/opinion/producto/id/699/ /opinion/producto/id/700/	opinionDeProducto

Lo mismo ocurre, por ejemplo, en acciones de visualización, o acciones sobre ítems en concreto, donde no interesa saber cuál es el ítem sino la actividad asociada al él, con lo cual se recomienda agrupar estos eventos indicando la acción independientemente del ítem.

Selección de procesos críticos
A pesar de los filtrados y agrupamientos realizados, los modelos generados a partir de esos

logs son complejos ya que, hay que recordar, que se está modelando un sitio web, el cual dispone muchos links, que pueden ser accedidos de muchas partes diferentes del sitio, lo que complejiza los modelos. En este caso, no se tiene un proceso o comportamiento predefinido de un usuario.

Dado que el foco de esta investigación es detectar evidencias de comportamiento inadecuado en el sitio, el cual potencialmente pueda tener como objetivo vulnerar el mismo, se opta por no modelarlo en su totalidad, sino porciones del mismo. Se sugiere seleccionar determinados traces que involucren actividades críticas y que puedan implicar un posible riesgo para el sistema, como por ejemplo, el login, acceso a campos de entrada o transacciones bancarias. También se pueden seleccionar actividades en base a si activaron reglas de ModSecurity.

Se genera un *trace* por ejecución de funcionalidad. Por ejemplo, para analizar el comportamiento de los usuarios al utilizar una funcionalidad, como puede ser consultar un item del sitio, se toma una cantidad fija de eventos previos y la misma cantidad fija de eventos siguientes a la ejecución de la funcionalidad antes mencionada. A esta cantidad se le denomina radio.

Por ejemplo, si se quiere obtener un *trace* asociado con radio 2, asociada a una actividad a3, y para un *trace* se tienen las siguientes actividades:

a1 a2 a4 a5 a7 **a3** a9 a10 a11 a12

Se toman sólo las actividades:

a5 a7 **a3** a9 a10

Para realizar la contabilización de actividades anteriores y posteriores no se consideran las actividades repetidas.

Es decir, si se define un radio 2 asociado a la actividad a4 para el siguiente *trace*:

a1 a2 a2 a3 **a4** a5 a5 a5 a6 a7

Se toman las actividades:

a2 a2 a3 **a4** a5 a5 a5 a6

En este caso el *trace* resultante termina teniendo un largo de 8 en vez de 5 si no se tuviesen actividades repetidas.

Esto evita la pérdida de información previa o posterior debida a los *loops*.

Tampoco se contabiliza para este radio, las reglas activadas. En tal caso si se define un radio 2 asociado a la actividad a4 para el siguiente *trace*:

a1 a2 a3 a4 a5 r1 a6 a7

Siendo r1 una regla activada, se toman las actividades:

a2 a3 a4 a5 r1 a6

La salida generada es un archivo XES por url “crítica“ elegida.

Ejemplo de generación de traces de la funcionalidad asociada a url4 a partir de un log, con radio 2

Usuario	Urls solicitadas al sitio durante la ejecución de usuarios	Generación log XES para la url 4 con radio 2
Usuario 1	url 1 url 2 url 3 **url 4** url 5	<trace> url 2 url 3 url 4 url 5 </trace>
Usuario 2	url 1 url 2 url 2 **url 4** url 5	<trace> url 1 url 2 url 2 url 4 url 5 </trace>
Usuario 3	url 1 url 2 url 3 **url 4** url 5 url 6 url 7 url 8 **url 4** url 9	<trace> url 2 url 3 url 4 url 5 url 6 </trace> <trace> url 7 url 8 url 4 url 9 </trace>

4.2.1. Herramienta de procesamiento de logs

En el marco de este proyecto, se creó una aplicación desarrollada en lenguaje Java, que transforma los logs generados por ModSecurity (*audit logs*) en archivos de log en formato XES. Esta aplicación utiliza la librería *org.jwall.web.audit* [16] para el procesamiento de los logs de ModSecurity. La aplicación está disponible en el espacio GitLab [14], junto a su instructivo de instalación y manual de usuario.

Esta aplicación recibe archivos en el formato especificado anteriormente generados por ModSecurity y los transforma a formato XES. También con ella se puede aplicar los filtros antes mencionados. Los mismos se configuran en un archivo de configuración en formato JSON donde se puede especificar:

- Filtros por regla, especificando los identificadores de las mismas.
- Filtros por extensiones de archivo, asociados a los request: como pueden ser .js, .css, .jpg
- Filtros por expresiones regulares: se pueden especificar expresiones regulares y se filtrarán los *requests* que cumplan con las mismas. Esto es especialmente útil para realizar filtrados de resultados intermedios como se explicó anteriormente en la sección "Filtrado de resultados intermedios".
- Agrupamientos de ciertas URLs, especificando a través de expresiones regulares cuáles son las URLs a transformar e indicando el texto al que se corresponderán.
- Especificar las URLs de las actividades claves a estudiar. Si corresponde a una URL que fue agrupada, se especifica el texto de la agrupación dado. También se especifica el radio a emplear.
- Se brinda la opción de activar o no la inclusión de reglas como eventos.

Por cada URL concreta a analizar se generará un archivo XES específico, recolectando información generada por ModSecurity. También la aplicación genera archivos en formato csv, que detallan:

- Las URLs a las cuales se les aplicó algún tipo de agrupamiento.
- Las URLs a las cuales no se les aplicó ningún tipo de agrupamiento.
- Información de las reglas que fueron activadas. Se genera un archivo por proceso. De cada una se muestra:
  - La URL original.
  - La URL transformada, si corresponde.
  - Método (POST/GET).
  - Identificador de la regla.
  - Mensaje genérico asociado a la regla.
  - Datos asociados a la activación de la regla.
  - Texto asociado a la regla para el *request* específico.
  - *Request body*

Se pueden ver ejemplos de esos csv generados en el anexo, en la sección A.1.

Los archivos que recibe la aplicación son archivos generados por ModSecurity. Los mismos cumplen el formato mostrado en la figura 4.2

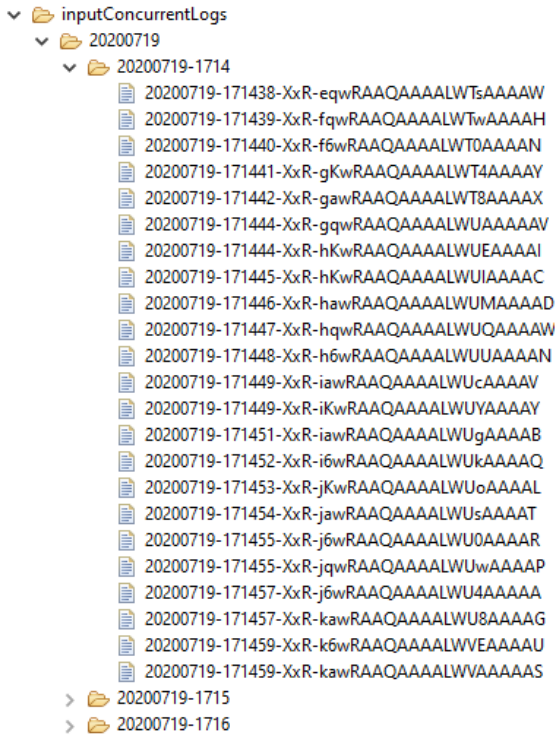

Figura 4.2: Archivos generados por ModSecurity

Como se puede ver en la figura 4.2, se muestran los archivos de entrada generados por el WAF. Se crea una carpeta por día con el siguiente formato de nombre aaaammdd, donde aaaa es el año, mm es el mes y dd es el día. Dentro de cada una se crea una carpeta por minuto con el formato aaammdd-hhmm donde hh son las horas y mm los minutos. Dentro de estas carpetas se guarda un archivo por *request*, con el formato aaaammdd-hhmmss-xxx-xxx, donde ss son segundos y xxx caracteres.

En la figura 4.3 se puede ver el archivo generado por la aplicación. El archivo contiene los eventos correspondientes a las URLs dentro del radio definido para una funcionalidad específica. Por cada ejecución de la funcionalidad a estudiar se genera un *trace*.

De cada evento se tiene:

- IP del *request*.
- *Timestamp*.
- *Concept name*, que se corresponderá con la URL si no fue agrupada o en caso de serlo,

```
<log xmlns="http://www.xes-standard.org/" openxes.version="1.0RC7" xes.features="nested-atributes" xes.version="1.0">
    <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
    <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
    <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
    <trace>
        <string key="concept:name" value="Case1"/>
        <event>
            <string key="org:resource" value="167.62.146.149"/>
            <date key="time:timestamp" value="2020-07-17T12:07:05.005-0300"/>
            <string key="concept:name" value="GET:/"/>
            <string key="debug:originalUri" value="GET:/"/>
            <string key="idRules" value=""/>
        </event>
        <event>
            <string key="org:resource" value="167.62.146.149"/>
            <date key="time:timestamp" value="2020-07-17T12:08:59.059-0300"/>
            <string key="concept:name" value="GET:menuCategory"/>
            <string key="debug:originalUri" value="GET:/men.html"/>
            <string key="idRules" value=""/>
        </event>
        <event>
            <string key="org:resource" value="167.62.146.149"/>
            <date key="time:timestamp" value="2020-07-17T12:09:15.015-0300"/>
            <string key="concept:name" value="GET:menuSubcategory"/>
            <string key="debug:originalUri" value="GET:/men/tops-men/tees-men.html"/>
            <string key="idRules" value=""/>
        </event>
        <event>
            <string key="org:resource" value="167.62.146.149"/>
            <date key="time:timestamp" value="2020-07-17T12:09:36.036-0300"/>
            <string key="concept:name" value="GET:menuSubcategory"/>
            <string key="debug:originalUri" value="GET:/men/bottoms-men/shorts-men.html"/>
            <string key="idRules" value=""/>
        </event>
        <event>
            <string key="org:resource" value="167.62.146.149"/>
            <date key="time:timestamp" value="2020-07-17T12:11:33.033-0300"/>
            <string key="concept:name" value="GET:productDetail"/>
            <string key="debug:originalUri" value="GET:/hero-hoodie.html"/>
```

Figura 4.3: Ejemplo de log XES generado por la aplicación

al texto correspondiente al agrupamiento.

- URL original, sin transformaciones.
- Reglas activadas por el evento.

Diseño de la aplicación
El proyecto se encuentra estructurado en cuatro paquetes. La figura 4.4 muestra la relación entre ellos.

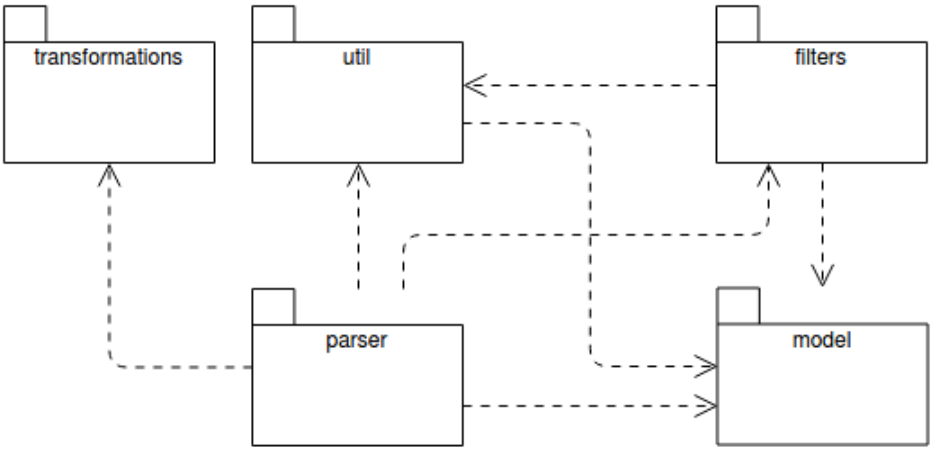

Figura 4.4: Diagrama de paquetes del ParserModSecurity

A continuación, se detalla cada uno de ellos.

Paquete Utils
Contiene clases auxiliares para el procesamiento de los archivos y carga de parámetros.

Paquete Model
Contiene clases que representan los eventos y las reglas aplicadas a cada uno de ellos.

Paquete Filters
Contiene clases que representan los filtros a aplicar sobre los eventos extraídos.

Paquete Transformations
Contiene clases que representan las transformaciones a aplicar sobre los eventos extraídos. Ahí se cargan las expresiones regulares de origen y destino definidas en el archivo de configuración.

Paquete Parser
Este paquete contiene la clase principal que se encarga de escanear un directorio con archivos de logs de ModSecurity, se encarga de filtrarlos y devolver los archivos XES correspondientes a cada proceso.

Funcionamiento general
Para cada archivo de ModSecurity identificado, se transforma su contenido en clases que representan el evento y las reglas aplicadas al mismo. Para cada evento, se aplican los filtros indicados en el archivo de configuración que se obtiene como entrada.

Finalmente, la estructura creada y filtrada de eventos y reglas, es transformada en único archivo de salida en formato XES por proceso definido en el archivo de configuración.

Aplicación de filtros
Los filtros son cargados a partir del archivo de configuración descrito en la sección anterior. Se crean instancias de una clase denominada GenericFilter abstracta. Tal como se especificó anteriormente, se tienen tres tipos de filtros diferentes. Cada uno de ellos se corresponden con las clases:
IgnoreFilter: Para el filtrado utilizando expresiones regulares.
ExtensionFilter: Para el filtrado por extensiones de archivo.
RuleFilter: Para el filtrado por reglas de ModSecurity.

En la figura 4.5 se pueden ver las clases del paquete Filters. Se tiene la clase abstracta GenericFilter, la cual implementa la operación *apply* que recibe una clase correspondiente a un evento y dependiendo de cuál se la implementación de la clase, devuelve si la clase del evento cumple o no con la condición de filtro. De esta forma, es muy sencillo agregar nuevos tipos de filtros, extendiendo de la clase GenericFilter y sin ser necesarias más modificaciones en la aplicación. Para ver más detalles sobre las clases de la aplicación, ver en el anexo, la sección A.2

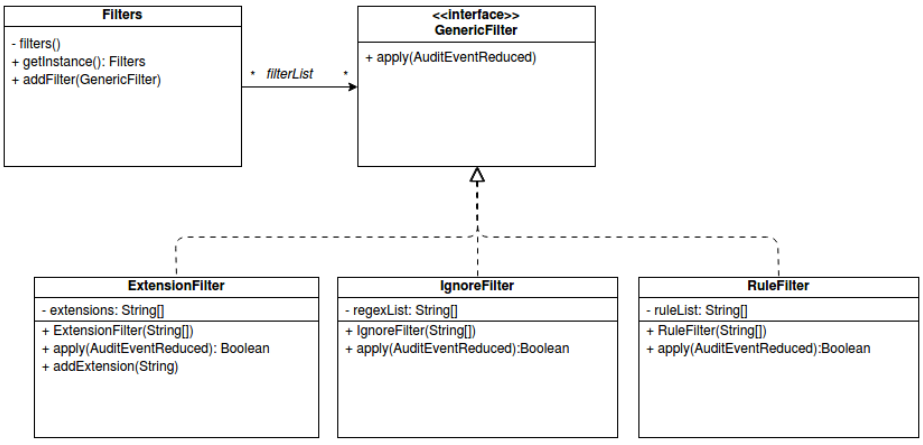

Figura 4.5: Estructura de clases del paquete Filters

4.3. Generación del modelo normativo

Se utiliza la herramienta ProM para aplicar las técnicas de Minería de Procesos, tanto para aplicar la técnica de *discovery* para obtener un modelo a partir del log, como para realizar *conformance* y reproducir los eventos del log sobre el modelo.

4.3.1. ProM

ProM (que es la abreviatura de *Process Mining framework*) es un *framework* de código abierto para algoritmos de Minería de Procesos. Es un *framework* extensible que admite una amplia variedad de técnicas de Minería de Procesos en forma de *plug-ins*. Es independiente de la plataforma, ya que está implementado en Java.

ProM 6 está dividido en partes. Primero, el *kernel* de ProM 6 se distribuye como un paquete descargable utilizando la Licencia Pública GNU (GPL) de código abierto. En segundo lugar, los *plug-ins* de ProM 6 se distribuyen como paquetes separados, normalmente utilizando la licencia de código abierto GNU *Lesser Public License* [17]. Un *plug-in* es básicamente la implementación de un algoritmo que es de alguna utilidad en el área de Minería de Procesos, donde la implementación es compatible con el *framework*. [18]

Hay cinco tipos de *plug-ins* implementados en ProM. Los mismos son:

- *Mining plug-ins*: implementan algún algoritmo, por ejemplo que construyen una Petri net basado en algún log de eventos.
- *Export plug-ins*: Implementan funcionalidades de guardado de algunos objetos, como por ejemplo, *plug-ins* que guardan *Petri nets*, *spreadsheets*, etc.
- *Import plugins*: Implementan funcionalidades de importación de objetos exportados.
- *Analysis plug-ins*: Implementan alguna propiedad de análisis sobre un resultado producto de aplicar alguna técnica de Minería de Procesos o para comparar logs contra el modelo.
- *Conversion plug-ins*: Implementan conversiones entre diferentes tipos de formatos datos. [19]

En la parte izquierda de la figura 4.6 se muestra el *workspace* del ProM 6. Se muestran los objetos cargados en la herramienta y las acciones que puede hacer el usuario al seleccionar uno de ellos. También se puede importar/exportar objetos desde/hacia un archivo. En la parte derecha de la figura, se pueden ver las acciones de visualización sobre un log. Esta vista muestra una lista de posibles acciones (*plugins*) con sus requeridos datos de entrada y salidas esperadas.

En ProM 6, los *plug-ins* pueden ser instalados dinámicamente usando el ProM Package Manager. Se trata de una herramienta separada que permite agregar o remover paquetes de la distribución de ProM instalada. Cada paquete contiene una colección de *plug-ins* y son ins-

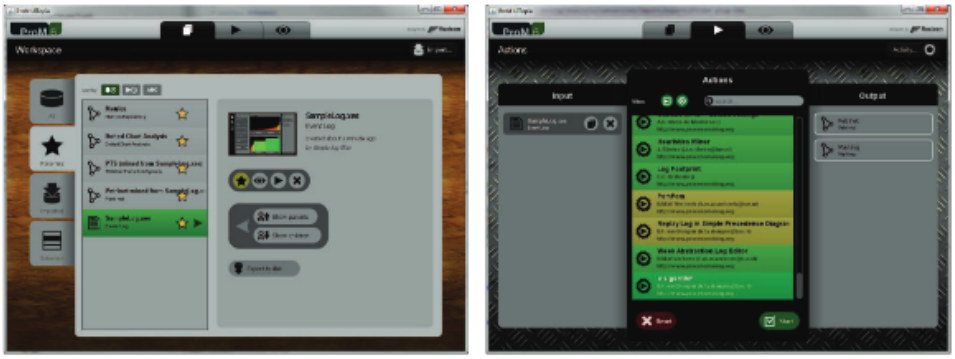

Figura 4.6: Interfaz de la aplicación ProM 6, extraído de [18]

talados si se instala el paquete. ProM contiene algunos *plug-ins* instalados por defecto y los usuarios pueden agregar más paquetes. También pueden crear sus propios *plug-ins*.

4.3.2. Generación del modelo normativo utilizando ProM

Al estar aplicando estas técnicas sobre el log de navegación de un sitio web, el cual típicamente es un proceso no estructurado, para generar el modelo se decide aplicar la variante del algoritmo *Inductive Miner Inductive Miner infrecuent* (IMF) [20], la cual es la variante ofrecida por defecto dentro del *plug-in* de *Inductive Miner* de ProM. Esta variante es recomendada porque, además de garantizar la generación de un modelo de tipo *workflow net* sin errores, aplica un filtrado de comportamiento infrecuente, el cual mejora de forma significativa su limitación para generar modelos adecuados.

Si el comportamiento infrecuente es incluido en el modelo, la simplicidad y precisión pueden verse comprometidas, generando un modelo que generaliza el comportamiento. Sin embargo, si el comportamiento infrecuente se excluye, el *fitness* puede verse afectado. Normalmente, el 80 % del comportamiento observado puede explicarse mediante un modelo que represente solo el 20 % del modelo requerido para describir todo el comportamiento. El modelo del 80 % muestra las partes principales del proceso, y para obtener ese modelo, un enfoque clásico es filtrar el log antes de descubrir el modelo. Esta versión del *Inductive Miner Infrecuent* implementa una mejora a este enfoque clásico de filtrado. Este consiste en aplicar el filtro de manera local en cada paso que da el algoritmo, sólo si en este paso el resultado obtenido es un *flower model*, es decir un modelo en el que todas sus actividades están conectadas. Este tipo de modelo no es deseable porque sólo enumera las actividades sin agregar información respecto al comportamiento, ya que todas las combinaciones posibles son aceptadas. Este enfoque de filtrado local resulta en la obtención de un modelo con mejores características y que se ejecuta en menor tiempo.

Para ejecutar este algoritmo se utiliza el *plug-in Mine Petri net with Inductive Miner* con todas sus opciones cargadas por defecto, las cuales son:

- *Variant*: *Inductive Miner - Infrecuent(IMf)*
- *Noise threshold*: 0,20
- *Event classifier*: *concept:name*

Al estar trabajando con la estrategia de modelar sólo el comportamiento cercano a ciertos eventos críticos, además de seleccionarlos, es necesario definir un valor de radio de alcance de eventos que ocurren antes y después de estos eventos críticos, el cual llamaremos r. Para esto se utilizan dos criterios. Un primer criterio conceptual en el que se evalúa que, si el r es demasiado pequeño, no se puede observar el comportamiento relevante previo y posterior al evento crítico. Si r es demasiado alto, los eventos que quedan registrados están muy lejanos al evento crítico y no aportan significado a este, pareciéndose más al log del sitio web completo, que al de un evento crítico.

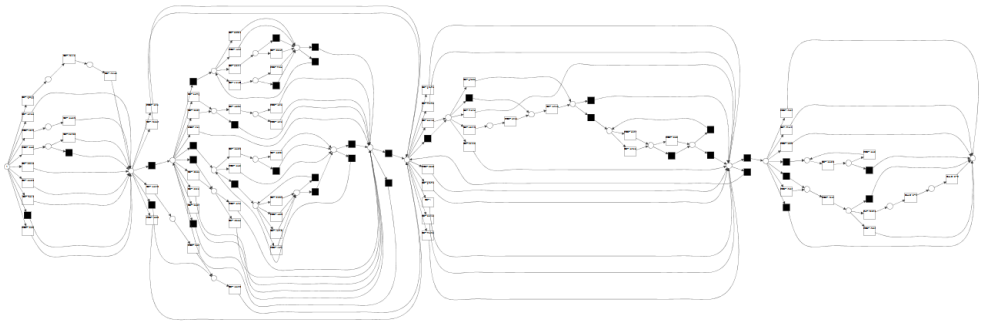

Figura 4.7: Ejemplo de una Petri net representativa de un modelo normativo obtenida aplicando Inductive Mining en ProM, a partir de un log procesado con la herramienta ParserModSecurity. Log enfocado en una funcionalidad concreta, con radio 4

Luego para elegir entre estos valores, se utiliza un criterio empírico, en que se elige el valor de r que obtenga un mejor valor de *fitness* entre el log y el modelo obtenido. Para esto, teniendo el log y la URL considerada crítica, se ejecuta el parser con los distintos valores de r. Luego para cada log XES, se genera el modelo correspondiente con el *Inductive Mining*, tal como se describió anteriormente. Con cada log y su modelo correspondiente a un mismo valor de r, se realiza un *conformance checking* con ProM . Para realizar el *conformance checking* se utiliza el *plug-in Replay a log on Petri net for conformance analysis* con sus opciones por defecto. Este *plug-in* implementa el método de *alignment* tal como se explica en la sección 2.2.4

Como entrada del *plug-in* se necesitan el log y la *Petri net* del modelo.

Al ejecutar el *plug-in* se da la opción de asociar costos especiales a ciertos movimientos no sincrónicos. Por defecto todos tienen valor 1.

De los resultados obtenidos con el *conformance checking*, se utiliza el valor promedio de *fitness*, eligiendo el modelo correspondiente al r que obtenga el mejor valor de *fitness*.

4.4. Generación de logs de uso habitual del sistema y ataques

Una vez generado el modelo normativo, se está en condiciones de compararlo contra el comportamiento real observado del sistema, y así tratar de identificar posibles desviaciones respecto al comportamiento esperado. Si se presentan desviaciones, deberán ser analizadas con mayor detenimiento, ya que podrían implicar un ataque al sistema. Con la utilización de la herramienta ModSecurity, se pueden generar logs de uso habitual del sistema por parte de los usuarios. Basta con tener configurado el WAF para realizar la captura durante el uso habitual del sistema. En esta etapa se propone generar dos nuevos conjuntos de logs para realizar *conformance checking* contra el mismo modelo y estudiar los resultados.

El primero de ellos, es un conjunto de logs obtenido a partir del uso bien intencionado del sitio, es decir, uno diferente al utilizado en la etapa de *discovery*, ya que es conveniente verificar que algunos *traces* diferentes a los que intervinieron en la generación del modelo normativo, producen valores de *fitness* alto (cercanos a 1) al realizar *conformance checking* contra el mismo.

El segundo es un conjunto de logs obtenido a partir de intentos de ataque sobre el sitio. Con este objetivo es que se propone utilizar la herramienta OWASP Zap [21]. Esta herramienta es un escáner de seguridad de código abierto y cuenta con una variedad de funcionalidades y modos de funcionamiento. En el contexto de este proyecto, se utilizará el modo de ataque activo, en el cual se especifica la URL del sitio que se desea escanear. La herramienta comienza a ejecutar peticiones maliciosas, probando distintos tipos de ataques. Las trazas del WAF obtenidas a partir de estos escaneos, posiblemente contengan reglas asociadas a los intentos de ataques.

4.5. Procesamiento de logs de uso del sistema

Los logs obtenidos en el paso previo, utilizando el WAF ModSecurity, deben ser procesados para poder aplicar Minería de Procesos sobre los mismos. Para ello, se utiliza la herramienta ParserModSecurity presentada en el paso 2, que recibe como entrada logs generados por ModSecurity, y devuelve logs de salida en formato XES, aptos para aplicar las técnicas de Minería de Procesos, aplicando filtrados y agrupamientos según la configuración que se establezca en la herramienta. Dado que el siguiente paso planteado tiene por objetivo realizar *conformance checking* de estos logs sobre el modelo normativo generado en el paso 3, para que los valores devueltos sean consistentes con el modelo, se debe utilizar la misma configuración del Parser que se utilizó para generar los logs con los que se generó el modelo

normativo. Esto para que no se generen diferencias entre los logs y el modelo, producto de haber aplicado diferentes filtrados de datos y/o agrupamientos de eventos.

4.6. Identificación y análisis de desviaciones

Luego de tener generados y procesados los logs de uso del sistema, se realiza el *conformance checking* utilizando *alignments*, empleando el modelo normativo generado en el paso 3. De esta forma se puede ver la desviación de los logs en relación al modelo.

Se debe tener en cuenta que se tendrá un log por proceso o funcionalidad definida inicialmente, lo cual implica que el *conformance* se realizará únicamente tomando en cuenta esa funcionalidad en concreto.

Para realizar el *conformance checking*, se utiliza la herramienta ProM ya presentada, cargando el modelo normativo generado en el paso 3. El *conformance* se realiza con el mismo *plug-in* y del mismo modo especificado en el paso 3.

En la herramienta, si se selecciona la opción de visualización *Project alignment to log*, se pueden ver los valores promedio, máximo, mínimo y desviación estándar asociados al *fitness*.

También se puede ver la cantidad de casos con valor de *fitness* 1, es decir, reproducibles de forma exacta en el modelo y la cantidad total de casos, así como el fitness de cada uno de los *traces*.

En la figura 4.8 se puede ver a la izquierda, información de cada *trace* del log, como puede ser valor de *fitness*, largo del *trace*, etc. En la parte central, los *alignments* de cada *trace*, donde se puede ver en verde los movimientos sincrónicos, en gris los no observados en el modelo y en amarillo los movimientos sólo vistos en el log. En la parte derecha se puede ver información general del log, como puede ser *fitness* promedio, máximo, mínimo, desviación estándar y cantidad de casos con valor 1. Estos valores pueden ser exportados a una planilla .csv para su uso posterior.

Interpretación de los resultados
Para interpretar los resultados e identificar potencial comportamiento malicioso se utilizan los siguientes datos:

- Valor de *fitness* promedio de los traces.
- Desviación estándar del *fitness* de los *traces*.
- Valor mínimo de *fitness* de un *traces*.
- Largo de los *traces*.

El primer valor a tener en cuenta es el de *fitness* promedio del log. Un valor del *fitness* promedio puede ser alto y su desviación estándar baja, e igualmente tratarse de un log que implica un ataque o intento de ataque al sitio. A continuación se exponen algunas razones por las

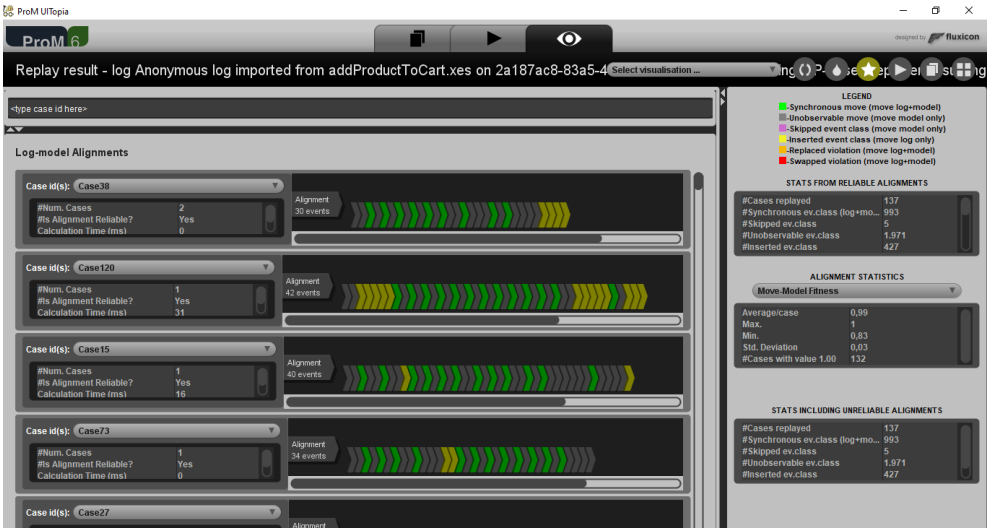

Figura 4.8: Salida obtenida de aplicar *conformance checking* de una *Petri net* contra un log.

cuales esto puede suceder.

- Muchos ataques se realizan utilizando el cuerpo del pedido HTTP, y esa información no se utiliza en la generación de los logs, por lo cual estos *requests* de ataque no son reflejados adecuadamente en los logs.
- Puede ocurrir que los ataques impliquen muchas peticiones a una misma actividad o conjunto de actividades. Eso se ve en el modelo como la ejecución de *loops*, pero éstos no son detectados por el modelo como algo incorrecto, aunque sí se puede tratar de un comportamiento sospechoso.
- Al hacer el agrupamiento de URLs, se pueden agrupar URLs maliciosas a URLs válidas y, de esa forma, esas trazas tengan su correspondencia en el modelo normativo, aunque esto no sea correcto.
- Se puede tener en un log comportamiento válido, y algunas trazas de comportamiento no válido. Al tratarse sólo de algunas, esto puede no afectar al *fitness* promedio. Esto ocurre debido a que algunos atacantes o herramientas de ataques pueden tener un comportamiento válido en el sistema. Para estos casos es necesario ver el detalle de *fitness* de cada *trace* del log, o tomar en cuenta el *fitness* mínimo del log para poder detectar el/los *traces* con comportamiento no adecuado.

El *fitness* será bajo en el caso de tratarse de un ataque donde se activen reglas de ModSecurity. Como las mismas son agregadas en los traces como eventos, el valor de *fitness* será bajo ya que en el modelo normativo no se tienen reglas modeladas. Igualmente si el *fitness* promedio es bajo, no implica necesariamente que se trate de un log con comportamiento no válido. Se puede tener un valor de *fitness* bajo debido a que el comportamiento del log no fue modelado

en el modelo normativo, pero sí se trata de un comportamiento válido.

Para interpretar los resultados obtenidos de realizar el *conformance checking* del log contra el modelo, se deben seguir los siguientes pasos:

- Si el valor del fitness mínimo (no del promedio):
  - Es bajo, entonces se debe analizar el *trace* con *fitness* mínimo y buscar si hay más *traces* con *fitness* bajo, para analizar si se tratan de casos de ataques.
  - Es alto, entonces se puede tratar de un log de ataque o no. Se debe realizar un análisis más profundo de los casos para asegurar que no se trata de un log malicioso.
- Si existen traces muy largos, se debe verificar si se debe a:
  - Una cantidad de *loops* excesiva.
  - Activación de reglas.

  En ambos casos se puede tratar de un comportamiento malicioso y se deben analizar esos casos en más profundidad.

En la siguiente sección se mostrará un caso de estudio donde se aplicarán los pasos detallados en esta sección. Se expondrá el análisis a realizar mostrando finalmente las conclusiones obtenidas, teniendo como objetivo la detección de comportamiento válido o no válido en el sistema.

5 Aplicación práctica

En esta sección, se presenta una aplicación concreta de todo el análisis y solución propuesta.

Se optó por elegir el sitio de *e-commerce* Magento [22]. Magento es una plataforma de comercio electrónico de código abierto, que ofrece un sistema de carrito de compras flexible. Está escrita en PHP y emplea el sistema de administración de base de datos relacionales MySQL o MariaDB.

Ofrece un catálogo bastante extenso de herramientas para la gestión, el *marketing* y la optimización de motores de búsqueda, por lo cual se le considera como una de las mejores plataformas *e-commerce* disponibles actualmente.

En lo que respecta a los beneficios del uso de Magento, esta es una plataforma que es fácil de instalar, además de que se pueden agregar diseños adicionales y *plugins*. Facilita la incorporación de algunas extensiones previamente diseñadas, como medios de pago, formas de envío, servicio al consumidor o diseño *responsive*.

En la figura 5.1 se puede ver una captura de la página principal del sitio.

Esta plataforma fue elegida para realizar la aplicación práctica de la solución por los siguientes motivos:

- Es un sitio relativamente simple y ampliamente utilizado.
- Ofrece un DevBox oficial [23], es decir, una versión modo desarrollo, lo que facilita la instalación de un ambiente local para pruebas.
- Esta versión de desarrollo viene precargada con un conjunto de datos, es decir categorías, productos, etc, con lo cual se simplifica la carga inicial para el análisis del sitio.
- Utilizando el DevBox, no es necesario instalar ni realizar configuraciones específicas de Apache y PHP.

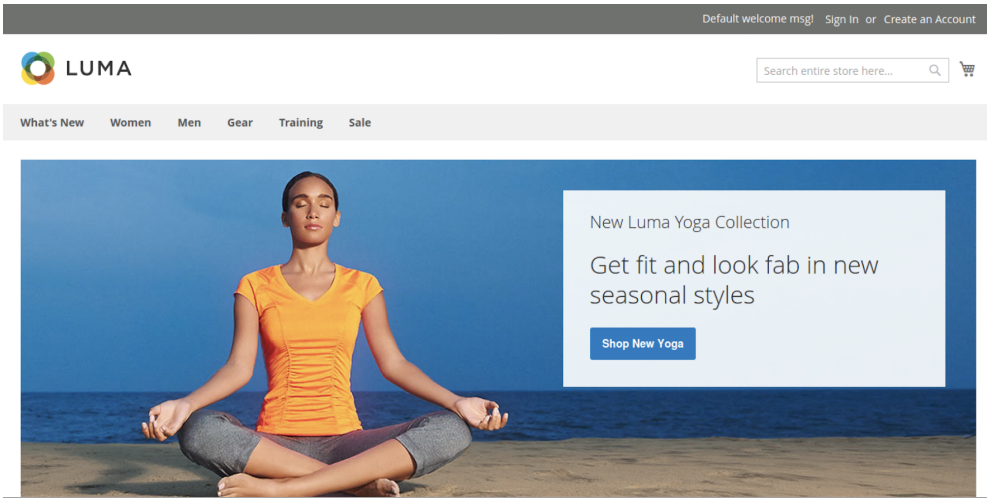

Figura 5.1: Captura de la página principal de Magento

Instalación del ambiente

Para simplificar aún más y poder replicar de manera más simple la instalación del ambiente completo, se optó por utilizar Docker [24]. Cada componente necesario se ejecutará en un contenedor de Docker. De esta forma, además de aislar los componentes, es posible replicar su instalación a partir de archivos Dockerfile. Se creó una estructura de directorios para este fin, disponible en un espacio de GitLab [25]. Siguiendo los pasos especificados en los archivos README, es posible instalar el ambiente en pocos pasos.

La figura 5.2 muestra la arquitectura del ambiente instalado para el estudio, donde cada componente especificado se corresponde con un contenedor de Docker.

El contenedor con MySql, contiene el esquema de base de datos que Magento necesita para su funcionamiento, instanciado con los datos de prueba que ofrece el DevBox. El contenedor Magento, contiene un servidor Apache, con una instalación de PHP, infraestructura básica sobre la cual ejecuta Magento. Este contenedor expone el sitio, el cual queda accesible a través de la IP local 172.17.0.3. La URL es http://172.17.0.3/. Por último, el contenedor Modsecurity contiene un servidor Apache con una instalación de ModSecurity. Este contenedor se configura como proxy reverso a la IP del contenedor Magento, por intermedio de archivos de configuración httpd. De esta forma, todas las peticiones HTTP que lleguen a http://172.17.0.4/ serán redirigidas a http://172.17.0.3/, y a su vez quedarán auditadas por el auditlog de Modsecurity

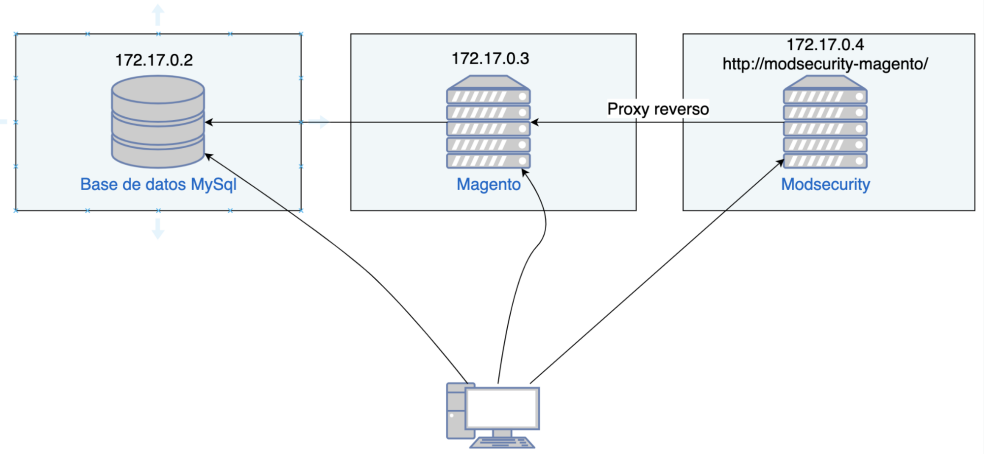

Figura 5.2: Arquitectura del ambiente

5.1. Paso 1: Obtención y generación de logs de uso válido del sistema

Para generar el modelo normativo correspondiente a un sitio web, se necesita contar con logs correspondientes a un uso del sistema que considere el apropiado. Para generar logs en el caso de estudio presentado, se publicó el sitio Magento, dejándolo accesible a través de internet, y se solicitó a un conjunto de aproximadamente 60 usuarios que navegaran por el sitio, haciendo un uso no malintencionado del mismo. Tal como se presentó al inicio la arquitectura del caso de estudio, se activó el registro de logs con ModSecurity.

5.2. Paso 2: Procesamiento de logs y zonas críticas

Los logs capturados fueron procesados con la herramienta ParserModSecurity presentada previamente, encargada de transformar los logs generados por el WAF a formato XES. Mediante la misma, se pueden configurar diferentes tipos de filtrados de los datos.

Definición de filtros
En este caso, se filtraron todas las URLs que impliquen solicitudes de archivos css, *javascript* e imágenes, correspondientes a extensiones: css, .js, .svg, .jpg, .png, .ico, .gif. Se aplicaron filtros de URLs asociadas a elementos, propios de la interfaz, que no aportan información acerca del comportamiento del usuario en el sitio, como pueden ser elementos estáticos, por ejemplo, URLs que comiencen con el siguiente formato: /pub/static/, /customer/section/ o

page_cache/block/render/.

También se aplicó un filtro de la regla 920350. Esta regla chequea que el *host header* no sea una dirección IP. Esto no es una violación HTTP RFC pero es un indicativo de un posible acceso automatizado, lo cual podría implicar un posible ataque. Puesto que la instalación del sitio utilizado se realizó en máquinas de forma local, se agregó el filtrado de esta regla. Esto implica que, si una petición activó esta regla, esta información no se agrega en los logs XES generados.

Otro filtro aplicado está relacionado con un campo de búsqueda. En el sitio Magento, se tiene un campo de búsqueda de ítems en el cual, a medida que el usuario ingresa letras en él, se realizan solicitudes AJAX que hacen peticiones al sitio, para ofrecer sugerencias al mismo, tal como se puede ver en la figura 5.3. Luego, al seleccionar el botón enter, se realiza otra petición con el el texto de entrada completo, tal como se puede ver en la figura 5.4. No es de interés reflejar en el modelo las peticiones por cada letra ingresada por el usuario, ya que no agrega información en cuanto al comportamiento del usuario para detectar vulnerabilidades. Por tal razón, se aplicó un filtro para no tener en cuenta esas URLs intermedias, filtrando las URLs que comiencen con el siguiente formato: /search/ajax/suggest/.

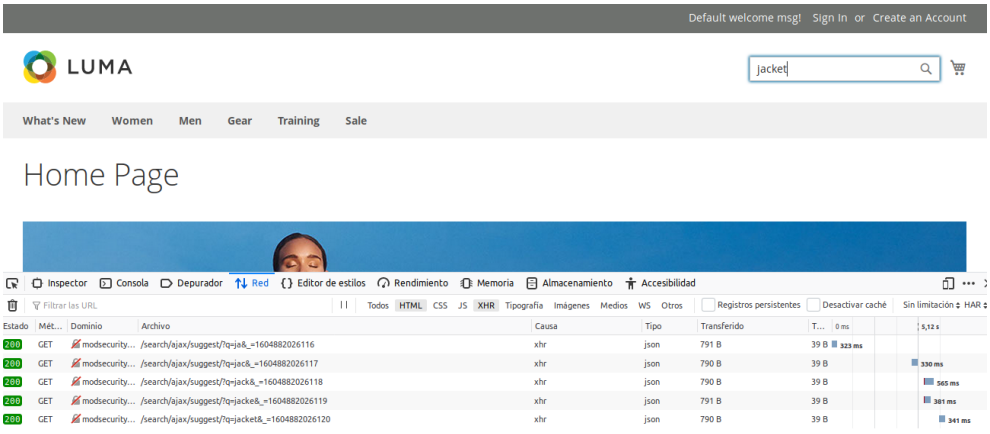

Figura 5.3: Búsqueda por palabra en el sitio Magento y las peticiones asociadas

Agrupamientos

Además del filtrado, se realizaron agrupamientos de URLs a actividades abstractas. Esto se aplicó con el objetivo de evitar modelos de tipo *spaghetti*, como fue explicado en la sección 3.3, con lo cual la técnica de *conformance checking* sobre el modelo no brindaría resultados confiables. Por tal razón, se decidió realizar agrupamientos de ciertas URLs que representan una misma acción en el sistema por parte del usuario a una sola actividad abstracta representativa de esa acción.

Por ejemplo, un agrupamiento realizado está relacionado con la consulta del menú del sitio. El sitio cuenta con un menú con categorías de productos, y dentro de cada categoría sub-

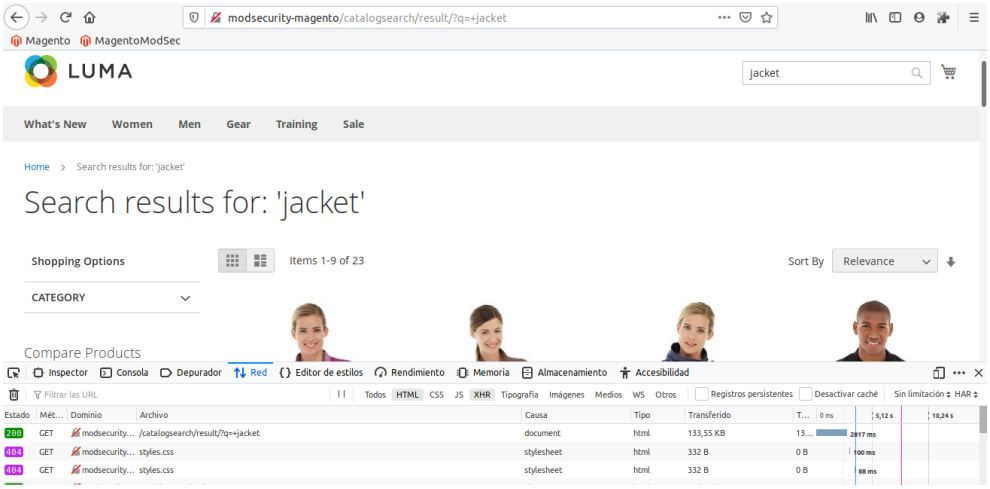

Figura 5.4: Búsqueda por palabra en el sitio Magento y petición asociadas al seleccionar enter

categorías, como se puede ver en la figura 5.5. Las URLs de acceso a las mismas fueron agrupadas a dos actividades abstractas: *menuCategory* y *menuSubcategory*, dependiendo de si la URLs implica el acceso a una categoría o a una subcategoría.

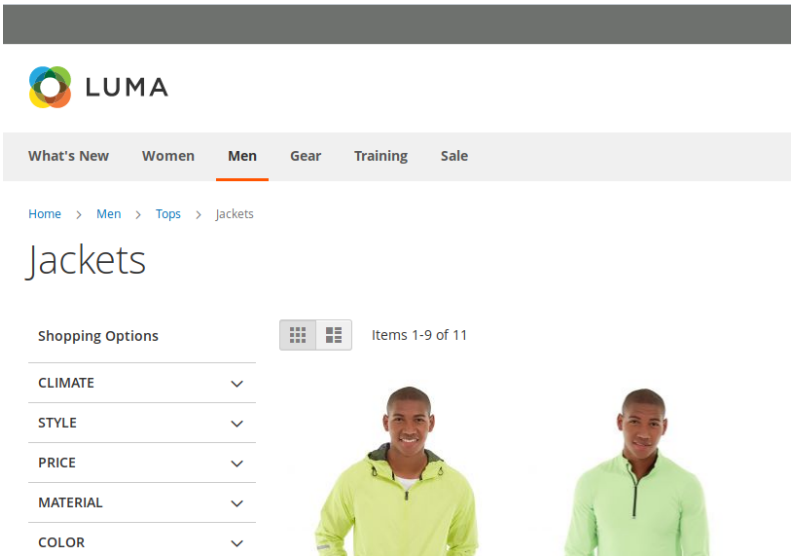

Figura 5.5: Se muestran las categorías y subcategorías presentes en el sitio Magento, por ejemplo dentro de la categoría *Men* está *Tops* y dentro de ella *Jackets*

Luego, dentro de cada subcategoría, se permiten seleccionar filtros, por ejemplo, ropa de mujer, de hombre, etc, como se puede ver en la figura 5.6. En este caso, se abstrae del filtro en sí mismo, y se agrupan estos accesos a una actividad *menuSubcategoryFiltered*.

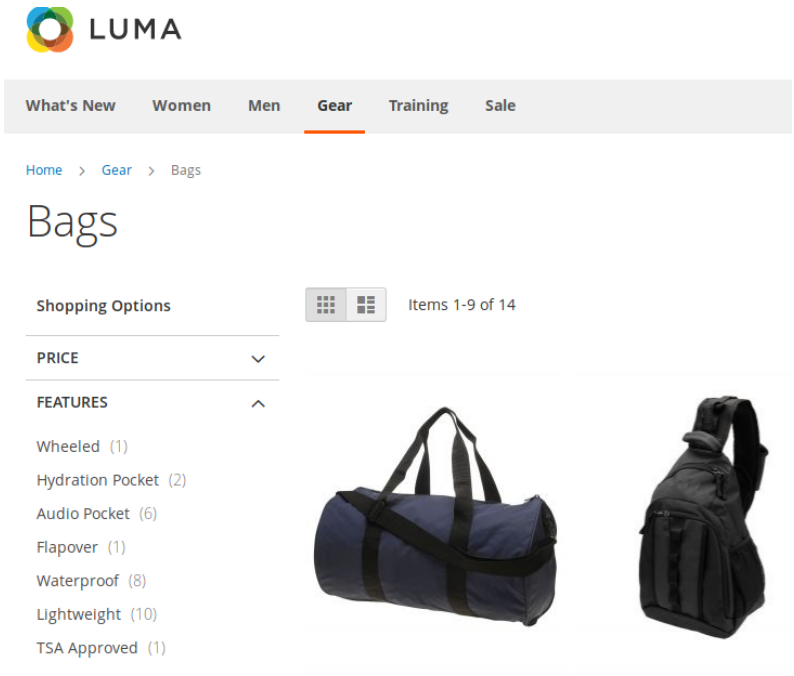

Figura 5.6: Se puede ver a la izquierda los filtros que se pueden aplicar dentro de una subcategoría, en este caso, la categoría *Gear* y subcategoría *Bags*.

Las actividades del log están asociadas a URLs, las cuales muchas de ellas contienen parámetros que representan identificadores. Estas URLs son diferentes entre sí, pero representan la misma acción en el sistema. Por ejemplo, las siguientes URLs se generan a partir de la visualización de la *review* de dos productos diferentes:

GET:/review/product/listAjax/id/699/
GET:/review/product/listAjax/id/700/

La URL es prácticamente la misma, excepto que contienen diferentes identificadores. En este caso, se generarían dos actividades diferentes en el modelo, pero ambas representan la consulta de las opiniones sobre un producto. Las dos URLs hacen referencia a una misma actividad de visualización de consulta pero de diferentes productos. Si se tuviese una actividad única por cada consulta de las opiniones de cada uno de los productos del sistema, el modelo tendría muchas actividades que además, no estarían aportando valor en cuanto al

URL	Descripción
checkout/cart	Ver carrito
/checkout/cart/add/uenc/aHs.../product/15/	Agregar un elemento al carrito
checkout/onepage/success	Compra exitosa
/catalogsearch/result/?q=jacket/>	Búsqueda de un ítem en el sitio
catalog/product_compare/add/	Agregado de ítem para comparar
catalog/product_compare/remove/	Se elimina uno de los ítems que se estaban comparando
customer/account	Información de la cuenta
customer/account/login	Login
gear/bags.html	Subsubcategoria
men/tops-men/hoodies-and-sweatshirts-men.html	Subsubcategoria
wishlist/	Consulta wishlist
wishlist/index/add	Agregado a la wishlist
women/tops-women.html?product_list_order=name	Ordenar los productos por nombre

Tabla 5.1: Ejemplos de URLs en el sitio Magento

comportamiento del usuario en el sitio. Por tal razón, se opta por agrupar estas URLs a una misma actividad en el modelo, en este caso, *reviewdeProducto*.

Lo mismo ocurre, por ejemplo, en las acciones de visualización, agregado al carrito o compra de un producto, donde no interesa saber cuál es el producto en concreto sino la actividad asociada al producto, con lo cual se optó por agrupar estos eventos como consulta o agregado al carrito de compras de un producto.

Ejemplos de URLs y descripción
En la tabla 5.1 se muestran algunas URLs del sitio y la funcionalidad o acción que representan. Esta información se utilizó para identificar los agrupamientos de las URLs.

Todos los agrupamientos aplicados en este caso de estudio y las expresiones regulares asociadas, se pueden consultar en el anexo la sección B.2

Definición de zonas críticas Además de los agrupamientos y filtrados realizados, se seleccionaron ciertas funcionalidades que se consideraron especialmente vulnerables a posibles ataques. En este caso, se utilizó la herramienta OWASP ZAP [21] para generar ataques al sitio. Los detalles de su funcionamiento se explicarán más adelante. Hubo ciertas actividades que generaron más activación de reglas de ModSecurity, las cuales se correspondieron con *addProductToCart*, asociada al agregado de un producto al carrito de compras del sitio, y *catalogSearchResult*, correspondiente a la búsqueda en el catálogo de productos. Con la ejecución realizada, se detectaron 17 reglas de ModSecurity activadas para la actividad *addProductToCart*, y 19 reglas para *catalogSearchResult*. Las mismas y su significados, pueden ser consultadas en el anexo B.1

5.3. Paso 3: Generación del modelo normativo

Se decidió estudiar las zonas críticas que rodean las actividades de *addProductToCart* y *catalogSearchResult*. Para ello fue necesario definir un radio de alcance de actividades que componen esas zonas críticas.

Se optó tomar 10 como valor máximo de radio ya que, con radios mayores, el largo de los *traces* puede alcanzar valores muy altos, sobre todo en los *traces* que incluyen *loops*, ya que las actividades repetidas no se toman en cuenta para definir el alcance del radio. Cuanto mayor tamaño tiene el *trace*, los eventos que se registran menos relación tendrán con las funcionalidades críticas.

Por ejemplo, para la actividad *addProductToCart* con radio 10, el largo máximo de un trace es de 40 eventos, bastante lejano a la media de 22, que se aproxima a la cantidad de actividades diferentes máxima que pueden existir en un log de radio 10.

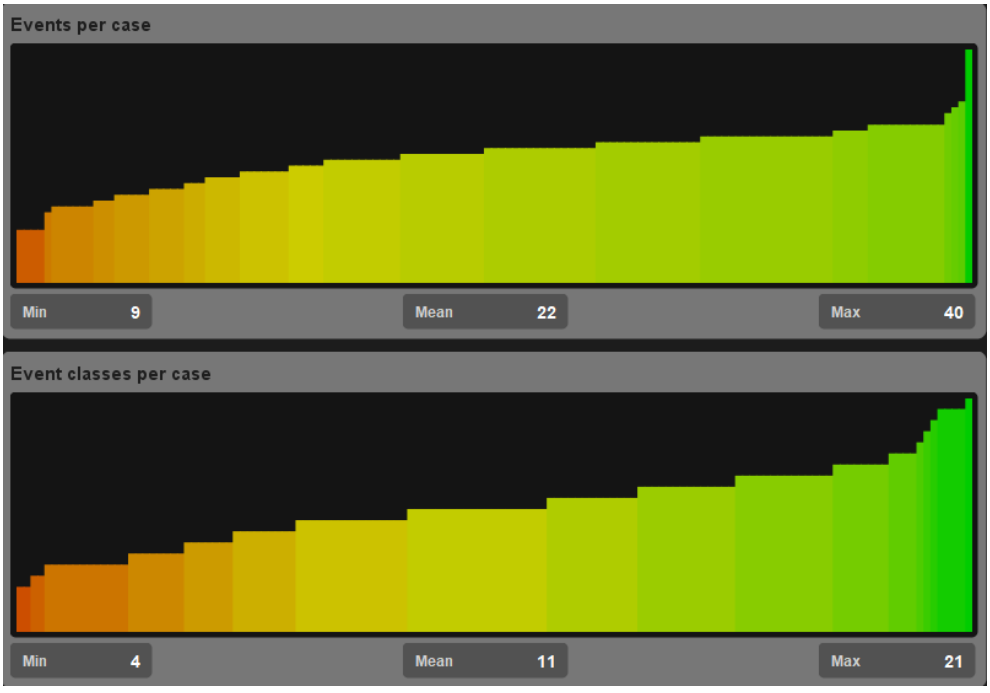

Figura 5.7: Visualización de Prom de eventos por trace (*Events per case*) y eventos únicos por *trace* (*Event clases per case*) correspondiente al log de *addProductToCart* con radio 10

Definición de radio para *addProductToCart*
En la tabla 5.2 se puede ver los largos de los traces del log utilizado para la funcionalidad *add-*

	Largo trace		
Radio	**Promedio**	**Min**	**Max**
1	3.47	3	9
2	5.81	4	11
3	8.1	5	20
4	10.36	6	26
5	12.47	7	29
6	14.42	8	21
7	16.31	9	24
8	18.16	9	36
9	19.9	9	38
10	21.55	9	40

Tabla 5.2: Largos de los *traces* del log utilizado para la funcionalidad addProductToCart según diferentes radios.

ProductToCart utilizando diferentes radios. A mayor radio, mayor largo promedio y máximo de los *traces*.

Los valores de *fitness* obtenidos en la tabla 5.3 , se obtienen como resultado del *conformance checking* entre el modelo normativo generado para cada radio, y el mismo log que lo generó, acotado a ese mismo radio. Se decide tomar en cuenta los tres valores de radio con mayor *fitness* promedio, los valores 6, 8 y 9 con valores 0.94, 0.96 y 0.94 respectivamente. Estos tres valores son muy similares y sería aceptable utilizar cualquiera de ellos. En este caso se optó por el modelo de radio 6 por tener *traces* de menor tamaño y mayor cantidad de casos con valor de *fitness* 1.

La figura 5.8, muestra el modelo *Petri Net* obtenido para el radio 6, resultado del algoritmo *Indiuctive Mining*

	Trace Fitness					
Radio	**Promedio/caso**	**Max**	**Min**	**Desviación estándar**	**#Casos con valor 1.0**	**#Casos totales**
0	0.94	1	0.50	0.14	112	137
1	0.87	1	0.40	0.17	74	137
2	0.72	1	0.38	0.15	15	137
3	0.92	1	0.56	0.11	80	137
4	0.87	1	0.50	0.13	46	137
5	0.84	1	0.50	0.14	39	137
6	0.94	1	0.57	0.08	73	137
7	0.91	1	0.47	0.10	47	137
8	0.96	1	0.59	0.07	72	137
9	0.94	1	0.57	0.08	60	137
10	0.82	1	0.45	0.15	30	137

Tabla 5.3: Largos de los *traces* del log utilizado para la funcionalidad addProductToCart según diferentes radios.

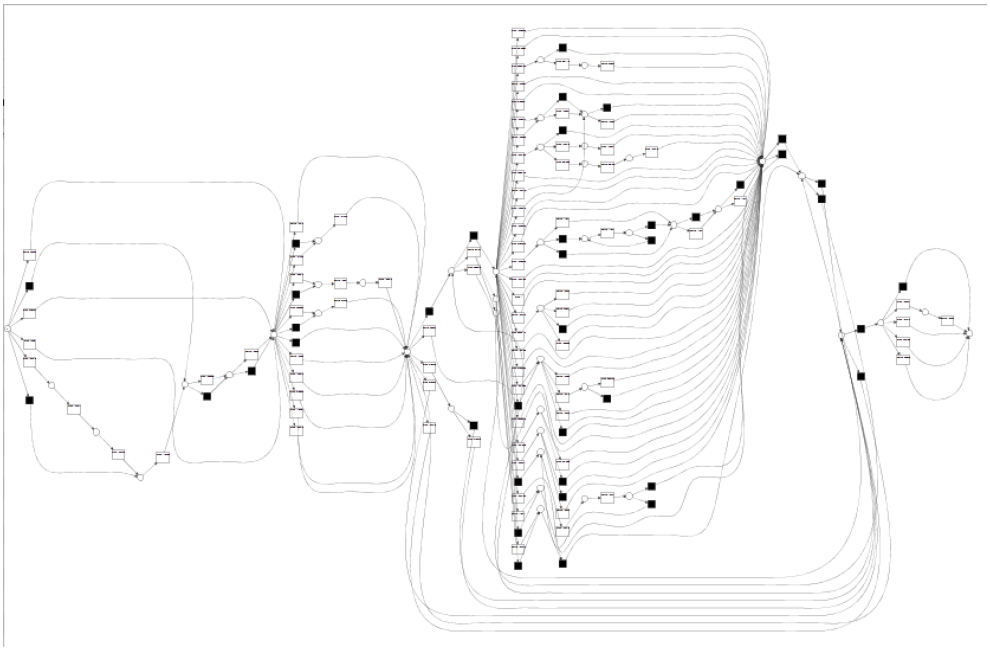

Figura 5.8: Modelo *Petri net* correspondiente al log de *addProductToCart* con radio 6

Definición de radio para *catalogSearchResult*

	Largo trace		
Radio	**Promedio**	**Min**	**Max**
1	3.86	2	8
2	6.19	3	10
3	8.7	4	12
4	10.92	5	15
5	13.4	6	18
6	15.71	8	20
7	18.08	9	22
8	20.52	10	25
9	22.63	11	27
10	24.81	12	30

Tabla 5.4: Largos de los *traces* del log utilizado para la funcionalidad *catalogSearchResult* según diferentes radios

En la tabla 5.4 se puede ver los largos de los *traces* del log utilizado para la funcionalidad *catalogSearchResult* utilizando diferentes radios. .

De acuerdo a la tabla 5.5, correspondiente a los valores de *fitness* obtenidos con diferentes radios, se decide tomar en cuenta los tres valores de radio con mayor *fitness* promedio, los valores 4, 5 y 8 con valores 0.71, 0.72 y 0.73 respectivamente. Estos tres valores son muy similares y sería aceptable utilizar cualquiera de ellos. En este caso se optó por el modelo con mayor valor de *fitness* promedio, el cual corresponde al valor de radio 8.

La figura 5.9, muestra el modelo *Petri Net* obtenido para el radio 6, resultado del algoritmo *Indiuctive Mining*

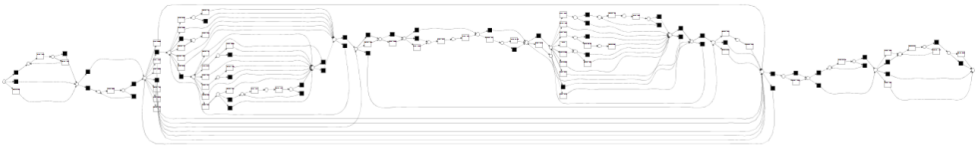

Figura 5.9: Modelo *Petri net* correspondiente al log de *catalogSearchResult* con radio 8

De acuerdo a lo explicado en la sección 3.3 acerca de la caracterización de procesos estructurados y no estructurados, si bien no hay una definición formal para diferenciarlos, una regla bastante utilizada para catalogar un proceso como *lasagna*, es la de verificar que el *conformance checking* entre el modelo y el log con el que éste se generó sea mayor a 0,8. En el caso de *addProductToCart* el valor de *fitness* es de 0,94, por lo que por esta regla se podría asumir que el modelo resultante es estructurado. En el caso del catalogSearchResult no se pudo obtener este valor, ya que el *fitness* promedio para todos los radios es menor a 0,8. Esto podría ser

	Trace Fitness					
Radio	**Promedio/caso**	**Max**	**Min**	**Desviación estándar**	**#Casos con valor 1.0**	**#Casos totales**
0	1	1	1	0.0	63	63
1	0.99	1	0.80	0.04	59	63
2	0.79	1	0.50	0.10	1	63
3	0.75	1	0.50	0.11	3	63
4	0.71	1	0.50	0.11	1	63
5	0.72	1	0.50	0.12	3	63
6	0.69	0.92	0.50	0.10	0	63
7	0.69	0.93	0.47	0.08	0	63
8	0.73	0.94	0.48	0.07	0	63
9	0.70	0.90	0.46	0.08	0	63
10	0.71	0.91	0.48	0.10	0	63

Tabla 5.5: Largos de los *traces* del log utilizado para la funcionalidad *catalogSearchResult* según diferentes radios.

debido a la poca cantidad de *traces* que se obtuvieron relacionados a esta funcionalidad, 63 casos para *catalogSearchResult* contra 137 casos para *addProductToCart*. Esto es razonable ya que la búsqueda de productos ingresando texto es sólo una de las opciones de navegación, pudiéndose llegar a ver los productos a partir de la navegación a través de los *clicks* a los links que se presentan en el sitio. Sin embargo, las actividades de *addProductToCart* son la única forma de agregar productos al carrito, por lo que terminarán siendo utilizadas por todos los usuarios que hagan compras en el sitio. También hay que notar que el acceso a la funcionalidad de *addProductToCart* está más restringida que *catalogSearchResult*, ya que para poder agregar un producto al carrito hay que cumplir ciertas precondiciones, tales como haber llegado a una página que describa el producto, luego seleccionar color, tamaño y cantidad, mientras que el acceso a las búsquedas está siempre disponible para realizarse en la esquina superior derecha del sitio. Un indicio para estimar la completitud de los datos, es la cantidad de los casos que repiten el mismo camino en el modelo. Si todos los *traces* recorren caminos diferentes, es un indicio de que todavía pueden haber más opciones de recorrido que no se han registrado en el log. Esto combinado con el hecho de tener un modelo complejo, también puede indicar que el proceso estudiado es de tipo *spaghetti*. Si por el contrario hay muchos *traces* que repiten su recorrido por el log, es un indicio de que se registró la mayoría del comportamiento, y que los próximos casos que puedan ser registrados, presentarán un comportamiento presente en el modelo.

En el caso de *addProductToCart*, de los 137 *traces*, 132 tienen un recorrido único. En tanto en *catalogSearchResult*, de los 63 *traces*, 52 tienen recorrido único. Datos que indican que se está trabajando con modelos complejos, poco estructurados y que el log está lejos de ser completo, por lo cual sería recomendable utilizar un log de mayor tamaño.

5.4. Paso 4: Generación de logs de uso habitual del sistema y de ataques

OWASP ZAP.
Para los casos de posibles ataques, se utilizó la herramienta OWASP ZAP [21] para generar solicitudes maliciosas sobre la aplicación, y se utilizó el WAF ModSecurity para almacenar las trazas.

ZAP es una herramienta utilizada para simular ataques a sitios web y encontrar vulnerabilidades que pueden ser utilizadas por atacantes externos. Algunas de sus funcionalidades son:

- Análisis de sistemas de autenticación.
- Posibilidad de lanzar varios ataques a la vez.
- Análisis pasivos.
- Análisis automáticos.
- *Plugins* para añadir más funcionalidades a la herramienta.
- Posibilidad de comprobar todas las peticiones y respuestas entre cliente y servidor.

La herramienta inicialmente adquiere la lista de posibles *request* que puede realizar (realizando un *crawling* automático a partir de la URL provista) para luego tratar generar ataques sobre las mismas que confirmen vulnerabilidades, como también verificar configuraciones correctas de los *headers* HTTP según el contenido de los mismos y del cuerpo.

En base a lo anterior, es necesario separar la ejecución de la herramienta en dos etapas. En la primera, denominada fase de reconocimiento, ZAP realiza *crawling* sobre el sitio, y guarda las URL que reconoce. En esta etapa, es importante configurar el valor *Maximum Children to Crawl*, que representa hasta qué hijo del árbol de recorrida visitar. En este caso se configuró con el valor 5, ya que por defecto tiene un valor infinito y esto puede ocasionar que la etapa se ejecute indefinidamente, sin finalizar.

En la segunda etapa, se ejecutan los intentos de ataque propiamente dichos, en base a las URLs detectadas por la herramienta en la etapa anterior.

A los efectos de seleccionar únicamente los logs de ModSecurity relativos a la etapa de ataque, es necesario identificar concretamente la hora en la que se ejecuta dicha etapa, ya que la etapa de reconocimiento también genera información que ModSecurity audita. Otra opción es borrar el contenido en el directorio de log del contenedor donde ejecuta ModSecurity una vez finalizada la etapa de reconocimiento.

La figura 5.10 muestra el progreso de los distintos tipos de ataque que la herramienta ejecuta sobre la URL especificada.

Una vez obtenidos los logs de la etapa de ataque, llamados log ZAP, los mismos fueron procesados utilizando la herramienta ParserModSecurity, con exactamente la misma configuración con la cual se generó el modelo normativo. Es decir, con los mismos filtros, agrupamientos y

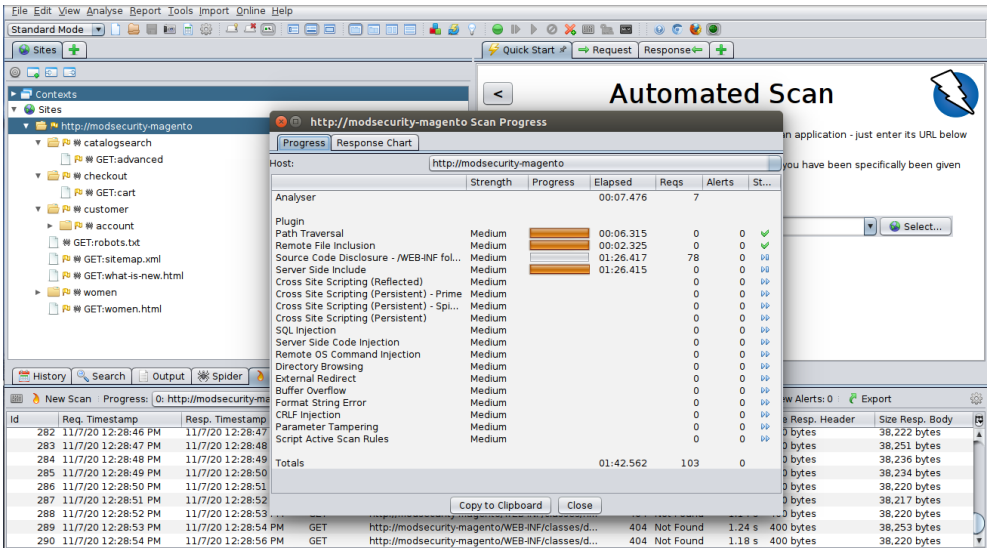

Figura 5.10: Etapa de ataque sobre la url http://modsecurity-magento/ utilizando la herramienta OWASP ZAP

radio para las zonas críticas.

El siguiente fragmento es un ejemplo de una porción de log obtenido de la fase de ataque:

```
<event>
    <string key="org:resource" value="172.17.0.1"/>
    <date key="time:timestamp" value="2020-07-19T14:27:24
        .024-0300"/>
    <string key="concept:name" value="
        POST:catalogSearchResult"/>
    <string key="debug:originalUri" value="POST:/
        catalogsearch/result/?q=ZAP"/>
    <string key="idRules" value="933160,932130"/>
</event>
<event>
    <string key="org:resource" value="172.17.0.1"/>
    <date key="time:timestamp" value="2020-07-19T14:27:24
        .024-0300"/>
    <string key="concept:name" value="RULE:Id=933160"/>
    <string key="debug:originalUri" value="RULE:/
        catalogsearch/result/?q=ZAP"/>
    <string key="idRules" value=""/>
```

```
</event>
<event>
    <string key="org:resource" value="172.17.0.1"/>
    <date key="time:timestamp" value="2020-07-19T14:27:24
       .024-0300"/>
    <string key="concept:name" value="RULE:Id=932130"/>
    <string key="debug:originalUri" value="RULE:/
       catalogsearch/result/?q=ZAP"/>
    <string key="idRules" value=""/>
</event>
```

En el primer evento, a causa de la diferencia entre el *concept:name* y el *debug:originalUri*, se puede deducir que la URL original fue agrupada. Luego se puede apreciar que el evento activó dos reglas de ModSecurity, cuyos identificadores son 933160 y 932130 respectivamente. Posterior al primer evento, se observan los dos eventos regla correspondientes.

Además de los logs generados con la herramienta OWASP ZAP, también se generaron logs realizando recorridas en el sitio de forma manual, con el objetivo de tener otro conjunto de logs diferente al utilizado para generar el modelo normativo. Se generaron dos conjuntos:

Uno de ellos, llamado Válido, se generó a partir de recorridas válidas en el sitio, es decir, que hacen un uso sencillo y usual del sitio, como ver algunos productos y realizar una compra. Estas recorridas son similares a las realizadas por la mayoría de usuarios que recorrieron el sitio a partir de las cuales se generaron logs para crear el modelo normativo. El otro conjunto, llamado Test, se generó a partir de recorridas relacionadas a un comportamiento no usual dentro del sitio, como por ejemplo navegar sin seguir el orden indicado por la interfaz de usuario, simulando ser un usuario de *testing*. Estas recorridas no implican un ataque al sitio, pero al representar un comportamiento poco probable de ser incluido en su totalidad por el modelo normativo, podrían ser identificadas como un comportamiento incorrecto y potencialmente malicioso.

Un log adicional, llamado Válido con Ataque, fue creado a partir del log Válido, agregándole además un trace de ataque extraído de los logs de ZAP. Para el caso de *addProductToCart*, este trace contiene 97 eventos, de los cuales 50 se corresponden con reglas.

Para el caso de *catalogSearchResult*, el trace adicionado contiene 18 eventos, de los cuales 9 se corresponden con reglas

5.5. Paso 5: Procesamiento de logs de uso del sistema

Luego de generar los conjuntos de logs anteriormente mencionados, los mismos fueron procesados utilizando la herramienta ParserModSecurity, utilizando exactamente la misma configuración con la cual se generó el modelo normativo.

5.6. Paso 6: Identificación y análisis de desviaciones

Una vez que se creó el modelo normativo y se generaron logs de uso del sistema, tanto representativos de uso válido como de ataques, se aplicó la técnica de *conformance checking* usando *alignments* del modelo contra los logs. Los logs utilizados fueron:

- Logs ZAP: Logs con comportamiento no válido, generados con la herramienta OWASP ZAP.
- Logs Test: Logs con comportamiento no esperado, generados manualmente.
- Logs Válidos con Ataque: Logs con comportamiento válido generados manualmente, y un caso de ataque agregado del log generado por la herramienta ZAP.
- Logs Válidos: Logs con comportamiento válido, generado de forma manual.
- Logs de Usuarios: Logs generados con varios usuarios utilizando el sitio, al dejarlo accesible a través internet.

5.6.1. Funcionalidad *addProductToCart*

Los resultados obtenidos de aplicar el *conformance checking* contra el modelo normativo de la funcionalidad *addProductToCart*, son los mostrados en la tabla 5.6

Log	**Trace Fitness**					
	Promedio/caso	**Max**	**Min**	**Desviación estándar**	**#Casos con valor 1.0**	**#Casos totales**
Válido	0,95	1,00	0,86	0,05	3	7
ZAP	0,73	1,00	0,34	0,24	18	323
Test	0,76	0,93	0,59	0,16	0	6
Válido con Ataque	0,89	1,00	0,47	0,18	3	8
Usuarios	0,95	1,00	0,57	0,07	74	137

Tabla 5.6: Valores de diferentes métricas relacionadas con el *fitness* de los distintos logs, obtenidos de aplicar *conformance checking* contra el modelo normativo

En la primera columna, se detalla qué log fue utilizado. El significado del resto de las columnas es el siguiente:

- Promedio/caso: Es el valor del *fitness* promedio por caso, en el log.
- Max: El valor máximo de *fitness* obtenido en un caso del log.
- Min: El valor mínimo de *fitness* obtenido en un caso del log.
- Desviación estándar: Desviación estándar del *fitness* de los casos del log.
- #Casos con valor 1.0: Cantidad de casos con *fitness* 1 en el log.
- #Casos totales: Cantidad de casos totales en el log.

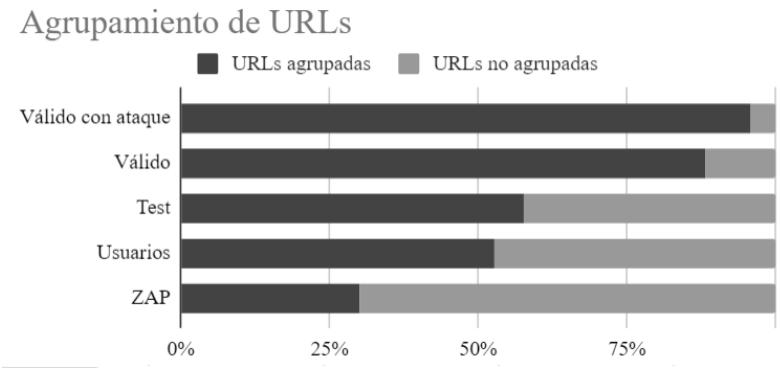

Figura 5.11: URLs agrupadas y no agrupadas de cada log

Fitness promedio

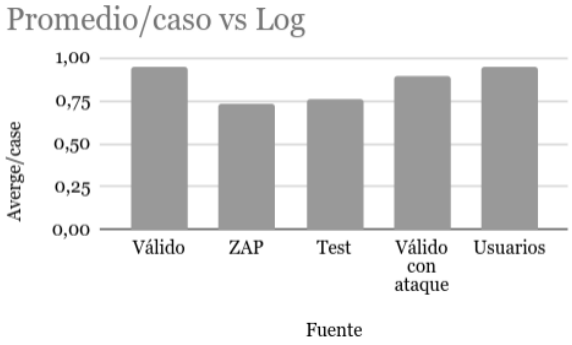

Figura 5.12: Valor promedio/caso de fitness de los diferentes logs

Como se puede observar en la figura 5.12, para los diferentes logs se obtuvo un valor de *fitness* promedio similar.

El log de ZAP tiene un buen *fitness* promedio debido a que la herramienta realiza muchos ataques en el cuerpo del pedido HTTP, y en este análisis no se está considerando esa información, con lo cual esas URLs se aceptan como válidas. También podría suceder que, debido al agrupamiento de URLs, se agrupen URLs inválidas quedándo como válidas. Por esta razón es necesario hacer una agrupación bastante restrictiva, como en este caso. Se puede ver en la figura 5.11 que el log de ZAP es el que tiene menos URLs agrupadas, y los logs Válidos tienen mayor cantidad de agrupamientos. Un factor que hace que los logs de ZAP presenten buen *fitness*, es debido a que muchas veces los ataques no implican una navegación inadecua-

da por el sitio, sino otro tipo de comportamiento que no afecta al *fitness*, tales como *loops* con una gran cantidad de iteraciones, pero no son detectados al realizar el *conformance checking* sobre el modelo como un comportamiento no válido, aunque sí podría implicar un intento de ataque al sitio. Por ejemplo, en el log de ZAP se puede ver que se agregan productos al carrito reiteradas veces, como se puede ver en la figura 5.13. En esta figura se muestran *traces* del log de ZAP, donde se pueden ver los eventos de cada uno. Los indicados en color verde se corresponden con los eventos de mayor frecuencia. En este ejemplo mostrado, coinciden con la actividad *addProductToCart*. En estos casos, no se disparan reglas de ModSecurity, y el *fitness* respecto al modelo normativo es de 0,94. Sin embargo, se realizan varios *traces* consecutivos con accesos a la misma funcionalidad de forma reiterativa, lo cual es un indicio de un posible comportamiento malicioso.

Figura 5.13: Fragmento de log de ZAP, utilizando la funcionalidad log inspector de la herramienta ProM. Los eventos en verde son los más frecuentes y los rojos los menos frecuentes.

El log Válido con Ataque presenta un *fitness* promedio alto, porque un sólo trace que no tenga comportamiento válido no afecta al promedio del *fitness*. Para detectar estos *traces* de ataques, es necesario ver en detalle cada trace del log para poder detectar si presenta un comportamiento no adecuado.

Los logs Válidos y de Usuarios tienen los promedios más altos, debido a que se corresponden con comportamiento adecuado.

El log de Test tiene menor valor de *fitness* promedio que el resto de los logs, puesto que el comportamiento que contiene es diferente al representado por el modelo.

Desviación estándar

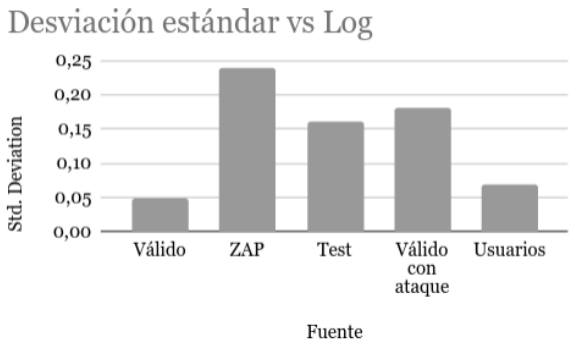

Figura 5.14: Desviación estándar del fitness de los diferentes logs

Como se puede ver en la figura 5.14, la desviación estándar se presenta más alta en el log de ZAP, ya que se tienen *traces* con grandes diferencias entre sí respecto a un comportamiento medio. Esto es porque la herramienta intenta atacar el sitio de diversas formas, y además genera la activación de diferentes reglas, lo cual acrecienta las diferencias entre los *traces*. Para el caso de los logs Válidos y de Usuarios, la desviación es mucho menor, puesto que el comportamiento de los *traces* de los logs son más similares entre sí, y en este caso, por tener *fitness* promedio alto, se adecuan más al modelo. La desviación es menor para el log Válido que para el log de Usuarios, ya que tiene sólo 7 casos y con *fitness* alto.

Casos con valor de *fitness* 1.0

Como se puede ver en la figura 5.15, el log de Usuarios fue el que obtuvo más cantidad de *traces* con *fitness* 1.0. Es el que tuvo valor más alto, puesto que con este log se generó el modelo normativo. El log de ZAP tuvo *traces* con *fitness* 1.0, pero en menor cantidad. Esto es debido a que a pesar de ser un log con ataques, algunas de sus recorridas son válidas para el modelo, aunque tenga una gran cantidad de iteraciones sobre una misma actividad y esto puede implicar un comportamiento malintencionado. El resto de los logs obtuvo menor cantidad de *traces* con *fitness* 1.0. Es difícil que los *traces* se puedan reproducir exactamente en el modelo, puesto que cumple con la propiedad de ser simple y muchos casos particulares no están registrados.

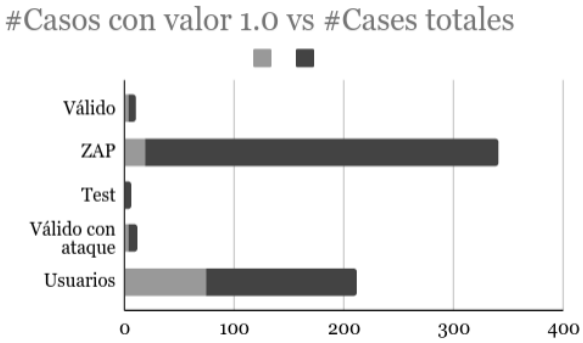

Figura 5.15: Cantidad de casos con valor de fitness 1.0 contra el total de casos de los diferentes logs

Interpretación de los resultados obtenidos

Para poder identificar si los logs presentados presentan ataques al sitio o no, se tomará en cuenta el valor del *fitness* de cada uno, los cuales fueron presentados previamente.

Como se puede ver en la tabla 5.6, el *fitness* mínimo del log Válido es alto y además, la desviación estándar es muy baja, lo cual es un indicio de que se puede tratar de un log con comportamiento válido. Además, como se muestra en la tabla 5.7, no se presentan *traces* largos que puedan llegar a implicar activación de reglas o iteraciones sobre una misma actividad.

Sin embargo, para el log de ZAP, el *fitness* mínimo es muy bajo, con lo cual se debe analizar mejor el *trace* en particular y/o los *traces* que tengan *fitness* bajo, ya que es probable que se trate de un ataque al sitio. Además su desviación estándar es más alta en comparación con el resto de los logs, lo cual es un indicio de un comportamiento variado, es decir, los valores de *fitness* no se encuentran uniformemente distribuidos respecto al promedio. Esto implica un posible comportamiento sospechoso. También se puede ver en la tabla 5.7 que el *trace* de largo máximo tiene un largo mucho mayor en comparación con los otros logs. Si se analiza el *trace* más largo, como se muestra en la figura 5.16, se puede ver que presenta muchos eventos correspondientes a reglas de ModSecurity.

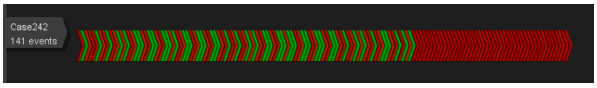

Figura 5.16: Análisis de trace de log de ZAP, donde la mayoría de los eventos en rojo se corresponden con reglas de ModSecurity

Los demás logs, de Test y Válido con Ataque, presentan *fitness* mínimo entre 0,47-0,59, por

Logs	Largo máximo de trace
Válido	17
ZAP	141
Test	17
Válido con Ataque	97
Usuarios	31

Tabla 5.7: Largo máximo de los traces de cada log

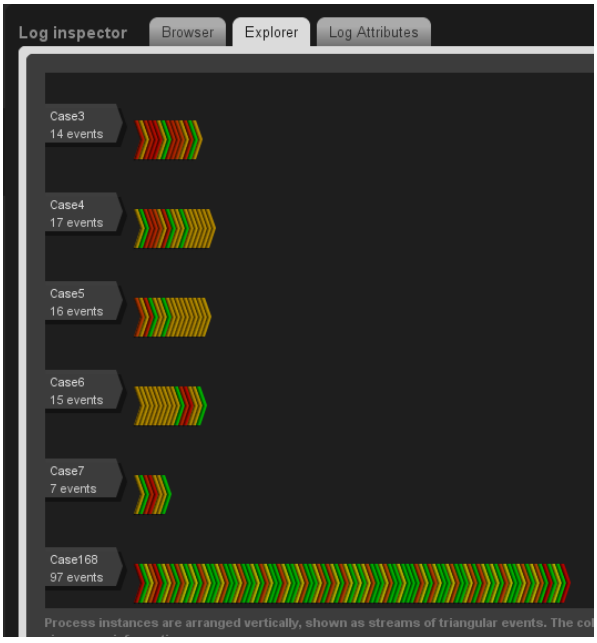

Figura 5.17: *Traces* del log Válido con Ataque utilizando la funcionalidad log inspector de ProM, donde se puede observar que existe un *trace* más largo que el resto, el cual es conveniente analizar

lo cual hay que analizar más cada uno de ellos.

El log Válido con Ataque presenta un largo máximo de *trace* mayor que los logs de Usuarios o Válidos. Además, como se definió un radio de 6, presentar un *trace* de largo 97 puede ser debido a que contiene muchas iteraciones de actividades, lo cual puede ser sospechoso, o bien se activaron reglas de ModSecurity, ya que ni los *loops* ni las reglas contabilizan para el radio. En este caso, se puede ver en la figura 5.17 que hay un único *trace* más largo que los demás. En este caso, se activaron muchas reglas de ModSecurity en un *trace*, como puede verse en la figura 5.18.

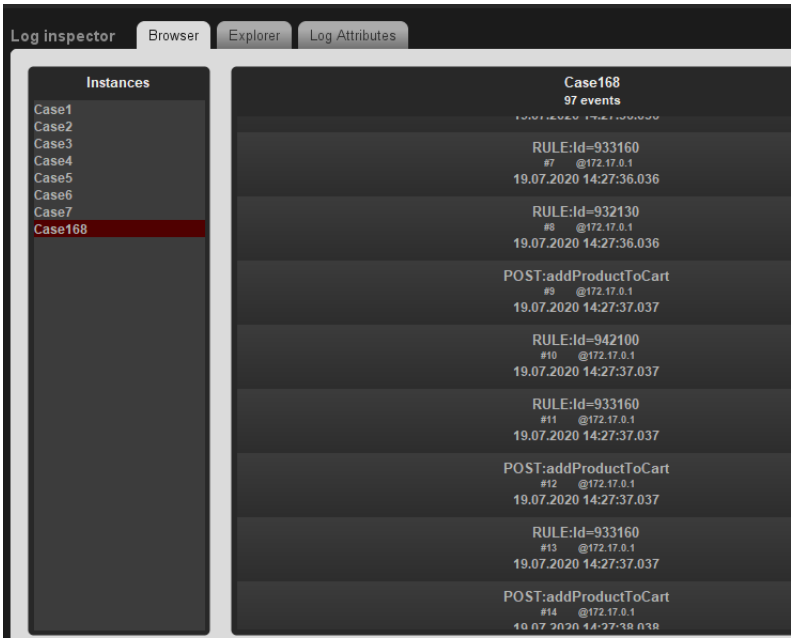

Figura 5.18: Fragmento de log Válido con Ataque, donde se puede ver el trace más largo conteniendo reglas de ModSecurity, utilizando la funcionalidad log inspector de ProM

El log de Test presenta un largo máximo de *trace* con un valor 17 lo cual indica que no hay *loops* de gran tamaño ni un *trace* que pudiese estar activando una gran cantidad de reglas, por lo cual, es posible que se trate de comportamiento no identificado en el modelo normativo. En ese caso hay que decidir si ese comportamiento será permitido o no en el sitio.

Los logs que se presentan en esta sección pueden separarse en logs que contienen ataques y logs que no contienen ataques. Los logs que no contienen ataques son Usuarios, Válido y Test. Los que poseen ataques son ZAP y Válido con Ataque. Resulta interesante identificar algún valor que diferencie estos dos grupos para lograr identificar los potenciales ataques. Como se vio, el valor de *fitness* promedio no resulta tan útil ya que pueden haber logs con mezcla de traces válidos y unos pocos traces con ataque. Incluso pueden llegar a tener un *fitness* promedio mayor al de algún log sin ataques pero con comportamiento no usual. Como ejemplo se puede ver que valor promedio de Válido con Ataque es mayor que el valor promedio de Test.

El valor de *fitness* mínimo, el grupo de logs Usuarios, Válido y Test poseen los valores 0,57, 0,86 y 0,59 respectivamente, mientras que los logs ZAP y Válido con Ataque poseen valores 0,34 y 0,47. En este caso los dos grupos pudieron diferenciarse.

5.6.2. Funcionalidad *catalogSearchResult*

Los resultados obtenidos de aplicar el *conformance checking* contra el modelo normativo de la funcionalidad *catalogSearchResult* son los mostrados en la tabla 5.8.

Tipos de logs	**Trace Fitness**					
	Promedio/caso	**Max**	**Min**	**Desviación estándar**	**#Casos con valor 1.0**	**#Casos totales**
Válido	0,84	0.91	0,75	0,06	0	8
ZAP	0,42	0,80	0,10	0,17	0	58
Test	0,51	0,70	0,33	0,11	0	12
Válido con Ataque	0,72	0,90	0,21	0,21	0	9
Usuarios	0,71	1,00	0,50	0,11	1	63

Tabla 5.8: Valores de diferentes métricas de *fitness* de los distintos logs, obtenidos de aplicar *conformance checking* contra el modelo normativo.

Fitness promedio

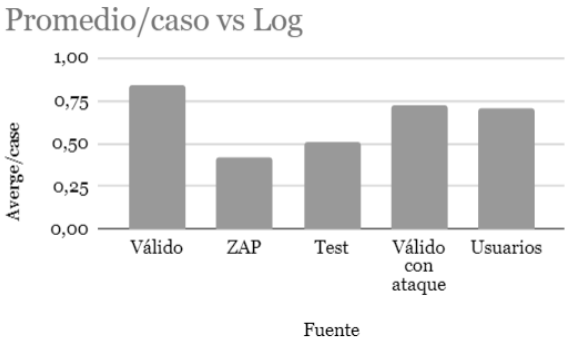

Figura 5.19: Valor promedio/caso de fitness de los diferentes logs

En la figura 5.19 se puede ver, que los logs que tienen *fitness* promedio más alto son los logs Válidos, de Usuarios y Válido con Ataque, ya que que presentan un comportamiento similar al reflejado en el modelo normativo. El log Válido con Ataque presenta buen *fitness*, porque sólo un *trace* de ataque no afecta el valor promedio. Igualmente, el *fitness* de este log es mayor que el del log de Usuarios, el cual se usó para crear el modelo. Los logs de ZAP y Test, presentan *fitness* promedio más bajo, puesto que sus *traces* no se adaptan al modelo, tanto por contener recorridas poco usuales, como por la presencia de reglas de ModSecurity.

Logs	Largo máximo de trace
Válido	21
ZAP	144
Test	21
Válido con Ataque	18
Usuarios	25

Tabla 5.9: Largo máximo de los *traces* de cada log

Desviación estándar

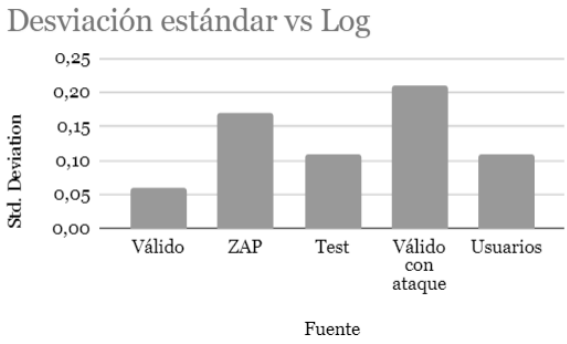

Figura 5.20: Desviación estándar del *fitness* entre los diferentes logs

Como se puede ver en la figura 5.20, la desviación estándar más alta detectada se dio con el log Válido con Ataque, ya que el *fitness* del *trace* de ataque era muy bajo. No sucedió lo mismo con el proceso *addProductToCart*, donde la desviación estándar menor se dio en el log de ZAP y no en el log Válido con Ataque. El *trace* de ataque agregado en ese caso, no obtuvo un *fitness* tan bajo como en *catalogSearchResult*. También se puede ver que la desviación estándar del log de Usuarios (con el que se creó el modelo) es igual que el log de Test, el cual tiene un comportamiento no típico. Esto indica que los *traces* del log de Usuarios tienen un comportamiento muy diverso entre sí, lo cual puede dificultar la identificación de patrones en común.

Casos con valor de fitness 1.0

Como se puede ver en la figura 5.21, sólo con el log de Usuarios se logró obtener *traces* con *fitness* 1.0. Este valor es mucho menor en comparación con lo observado en el log de *addProductToCart*.

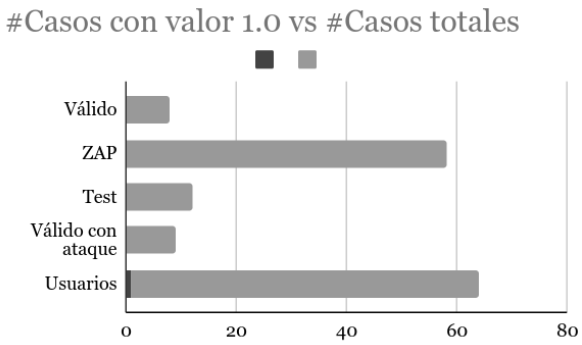

Figura 5.21: Cantidad de casos con valor de *fitness* 1.0 vs el total de casos de los diferentes logs

Interpretación de los resultados obtenidos

Al crear el modelo normativo, el *fitness* promedio de éste contra el log utilizando para su creación obtuvo valor menor a 0,8. Como bien se explicó anteriormente, con este valor no se puede asegurar que las técnicas de Minería de Procesos brinden resultados confiables. Se puede seguir el procedimiento planteado para detectar desviaciones en un sitio web, pero en procesos que impliquen una serie de pasos estructurada. En este caso, al tratarse de un campo de entrada fijo en la página, es más difícil detectar un comportamiento típico, puesto que la funcionalidad puede ser invocada desde múltiples puntos del sitio, lo cual dificulta detectar un comportamiento previo determinado. También por ser un modelo generado por un log pequeño y por ser incompleto, es decir está lejos de poder representar todas las estructuras de flujos de control subyacentes. Igualmente se puede observar que tanto los valores de *fitness* mínimo como los del largo máximo de un trace, como se puede ver en la tabla 5.9, se corresponden con los logs que contienen ataques.

6 Conclusiones

6.1. Evaluación de la solución

El objetivo de este trabajo fue analizar las herramientas que brinda la Minería de Procesos para agregar seguridad a los sitios web, a partir de la identificación del comportamiento de los usuarios en el sitio, y con el uso de un WAF.

Primeramente, se realizó un estudio de un trabajo previo y de las herramientas que ofrece la Minería de Procesos para descubrir, mejorar modelos y realizar chequeo de conformidad de datos contra modelos preestablecidos.

La conclusión que se obtiene de este estudio es que la aplicación de la Minería de Procesos puede ser especialmente útil para identificar cuál es el comportamiento habitual de los usuarios del sitio. Al tener este conocimiento, es posible explotarlo tanto para identificar posibles vulnerabilidades del sitio, o extraer información acerca del comportamiento desconocido de los usuarios. Esto podría ser utilizado para datos estadísticos, o para poder adaptar el diseño del sitio con el objetivo de que los usuarios lo utilicen de la forma deseada. También podría ser útil, por ejemplo, para decidir en qué áreas agregar más seguridad o que sección destacar más.

También en este estudio se adaptó un trabajo previo, ajustándolo al contexto de uso de un WAF. En ese trabajo el modelo normativo era realizado con UML, pero en este caso se realiza automáticamente utilizando la técnica de *discovery* de Minería de Procesos, y haciendo uso de los logs generados por ModSecurity. También se incorporaron las reglas de la herramienta a los logs, para luego poder hacer *conformance checking* de logs sobre el modelo generado.

Las limitantes identificadas en lo que respecta a la solución propuesta, se pasarán a detallar a continuación:

- No todos los ataques pueden ser identificados a partir de las URLs a las que acceden los usuarios. Un ejemplo visto son los ataques que son realizados en los cuerpos de los mensajes HTTP, los cuales presentan URLs válidas y se corresponden en el modelo con actividades válidas. Estos ataques pueden ser indirectamente detectados agregando las reglas activadas de Modsecurity, como eventos del log.

- Las iteraciones sobre las actividades que son realizadas muchas veces no son detectados analizando el *fitness* de los modelos, cuando éstos pueden implicar un intento de ataque. Por esta razón, se considera en el análisis de logs, el largo de los *loops* puesto que es una forma de detectar ese comportamiento sospechosos.

- No es útil para funcionalidades en un sitio web que son accesibles desde muchos puntos del mismo, puesto que es más difícil extraer un patrón de comportamiento habitual y generar un modelo que no sea del tipo *spaghetti*. Por esta razón, antes de dejar expuesta una funcionalidad desde muchos puntos, se debe tener en cuenta el balance entre la usabilidad y la seguridad, puesto que es más difícil controlar una funcionalidad que sea demasiado accesible.

- Sin un preprocesamiento adecuado, se obtienen modelos que aportan poca información a la hora de utilizarlos como referencia, y esto implica un conocimiento de la realidad o aplicación concreta que se quiere estudiar.

Dadas las limitaciones antes expuestas, el agregado de reglas de ModSecurity como eventos a los logs es fundamental para penalizar los casos donde el modelo no puede identificar cuerpos de mensajes HTTP con contenido malicioso. Es la forma de combinar la información brindada por ModSecurity a través de sus reglas con el modelo.

Los resultados obtenidos producto de la aplicación de la solución planteada, en una primera instancia, no son determinantes para identificar si un log contiene ataques o no, pero sí brinda una aproximación de lo que podría contener. Si los logs no se adecuan al modelo, se requiere un análisis más profundo para identificar si se trata de un ataque o un comportamiento no especificado. Si se adecúan, es más probable que se trate de un comportamiento adecuado. Es probable y no seguro, dadas las limitaciones antes presentadas. Es de suma importancia realizar un correcto agrupamiento de las URLs en la etapa de preprocesamiento de logs, dado que esto puede ser un punto crítico que determinará la detección o no de determinados casos de ataques, dependiendo de si estos son agrupados correcta o incorrectamente. El agrupamiento de URLs, puede ser un problema, ya que se podría estar perdiendo, en algunos casos, información sobre determinados ataques. Por otra parte, al agrupar de menos se continuaría teniendo el problema de los modelos *spaghetti*.

6.2. Trabajo a futuro

En esta subsección se presentarán los posibles trabajos a futuro a realizar, a partir del estudio realizado:

- La solución y aplicación práctica planteadas en este trabajo, es un análisis post mortem de lo que sucedió con el sitio web estudiado, utilizando logs generados con el uso del sistema. Otra aplicación de interés, sería mantener un modelo normativo previamente generado, y disparar algún tipo de alerta en base a un umbral de *fitness* definido como mínimo aceptable o al detectar determinada cantidad de iteraciones sobre una misma funcionalidad. También se podría utilizar la técnica de *enhancement* para retroalimentar el mencionado modelo normativo y descubrir nuevo comportamiento válido en base a nuevos registros de log generados por el uso del sitio. En el presente trabajo, el modelo normativo quedó estático con la información obtenida en una etapa inicial de definición de lo que sería el comportamiento válido.

- Otro estudio posible podría ser continuar con la aplicación práctica definiendo otros procesos críticos, y analizar los resultados que se obtienen. Por ejemplo, podría considerarse el inicio de sesión, u otros procesos que involucren una sesión, o un usuario autenticado, que por simplicidad fueron omitidos de este estudio.

- Respecto a los elementos a analizar, éstos pueden ampliarse y considerar, no solamente las URLs de las peticiones HTTP, sino también los cuerpos de las mismas para identificar posible comportamiento malicioso.

- En cuanto a la incorporación de reglas en el ModSecurity, se podrían tomar en cuenta otras opciones de visualización de las mismas: modificar alguna interfaz o crear algún *plug-in* para ver esa información al momento de generar el modelo normativo en ProM.

- En consideración a la definición del modelo, otra forma de abordar el problema podría ser definir el proceso a la inversa, es decir, en vez de generar un modelo normativo a partir de comportamiento válido, realizarlo a partir de ciertos ataques.

- Otro aspecto que es bastante común cuando se trata de minería de sistemas de información, es que las actividades no están predefinidas. Si se ve un modelo de proceso como una máquina de estados, se puede definir el estado de diversas formas, por ejemplo, la acción de un usuario, la página web a la que se accede, las reglas de ModSecurity que se activan, las transiciones que van desde zonas protegidas a zonas libres y viceversa. Entonces, una estrategia podría ser, dependiendo de lo que se busca, cambiar el punto de vista del log. En este caso se podría tomar los identificadores de las reglas de ModSecurity que se activan y utilizarlas como actividad. Tal vez los modelos sean más pequeños y permitan detectar patrones de activación de reglas en presencia de atacantes. Si se piensa en otras perspectivas y en la estrategia de agrupación de casos, se podría plantear que un caso no es el comportamiento de un usuario, sino el comportamiento de todos los usuarios en un día específico. Entonces, se tendría un caso por

día con un comportamiento usual que se puede comparar con los días subsiguientes y detectar variaciones en el comportamiento día a día.

Como se expuso anteriormente, aún hay mucho por investigar sobre la aplicación de Minería de Procesos para agregar seguridad a los sitios web. La seguridad de las aplicaciones es un área de suma importancia actualmente, donde muchos elementos de la realidad se basan en sistemas de información. El aporte de este trabajo es sólo un puntapié para continuar investigando y así mejorar la integridad y privacidad de los datos, aunque como bien se mencionó, también esto podría tener otro tipo de aplicaciones, como el análisis del comportamiento de los usuarios, entre otras.

Bibliografía

[1] SANS Institute. *Sitio oficial*. Disponible en: https://www.sans.org/information-security. Última consulta: 20/11/2020.

[2] Fundación OWASP. *Sitio oficial*. Disponible en: https://owasp.org/. Última consulta: 20/11/2020.

[3] Fundación OWASP. *Web Applicattion Firewall*. Disponible en: https://owasp.org/www-community/Web_Application_Firewall. Última consulta: 20/11/2020.

[4] Fundación OWASP. *Virtual Patching*. Disponible en: http://owasp.org/www-community/Virtual_Patching_Best_Practices. Última consulta: 20/11/2020.

[5] Aalst Wil M. P. van der. *Process Mining : Data Science in Action.* Vol. Second edition. Springer, 2016. ISBN: 9783662498507.

[6] Simona Bernardi, Raúl Piracés Alastuey y Raquel Trillo-Lado. «Using Process Mining and Model-driven Engineering to Enhance Security of Web Information Systems». En: *EuroS&P Workshops* (2017).

[7] Fundación OWASP. *OWASP Top Ten*. Disponible en: https://owasp.org/www-project-top-ten/. Última consulta: 20/11/2020.

[8] ModSecurity. *Open source web Application Firewall*. Disponible en: https://modsecurity.org/. Última consulta: 20/11/2020.

[9] Apache. *Sitio oficial*. Disponible en: https://httpd.apache.org/. Última consulta: 20/11/2020.

[10] SpiderLabs. *ModSecurity Wiki*. Disponible en: http://github.com/SpiderLabs/ModSecurity/wiki/. Última consulta: 20/11/2020.

[11] SpiderLabs. *ModSecurity Wiki*. Disponible en: https://github.com/SpiderLabs/ModSecurity/wiki/ModSecurity-2-Data-Formats. Última consulta: 20/14/2020.

[12] OMG. *Sitio oficial*. Disponible en: https://www.omg.org/. Última consulta: 20/11/2020.

[13] R.P. Jagadeesh Chandra Bose y Wil M.P. van der Aalst. *Abstractions in Process Mining: A Taxonomy of Patterns*. Inf. téc. Department of Mathematics y Computer Science, University of Technology, Eindhoven, The Netherlandsm Philips Healthcare, Veenpluis 5-6, Best, The Netherlands, 2017.

[14] Tamara Techera, Pablo Ibañez y Marcelo Bruno. *Repositorio ParserModSecurity*. Disponible en: https://gitlab.fing.edu.uy/marcelo.bruno/ParserModSecurity. Última consulta: 20/11/2020.

[15] Jesse James Garrett. *Ajax: A New Approach to Web Applications*. Inf. téc. AdaptivePath, 2005.

[16] Christian Bockermann. *Librería jwall audit*. Repositorio de código disponible en: https://bitbucket.org/cbockermann/org.jwall.web.audit/src/master/. Última consulta: 20/11/2020.

[17] *ProM Tools*. Disponible en: http://www.promtools.org/. Última consulta: 20/11/2020.

[18] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, W.M.P. van der Aalst. «The ProM Framework: A New Era in Process Mining Tool Support». En: *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings* (2005).

[19] W.M.P. van der Aalst, B.F. van Dongen, C. Günther, A. Rozinat, H.M.W. Verbeek, A.J.M.M. Weijters. *ProM: The Process Mining Toolkit*. Inf. téc. Department of Mathematics, Computer Science, Eindhoven University of Technology, Department of Industrial Engineering e Innovation Sciences, Eindhoven University of Technology, 2009.

[20] Sander J.J. Leemans, Dirk Fahland, Wil M.P. van der Aalst. *Discovering Block-Structured Process Models From Event Logs Containing Infrequent Behaviour*. Inf. téc. Eindhoven University of Technology, the Netherlands, 2009.

[21] Fundación OWASP. *OWASP Zap*. Disponible en: https://www.zaproxy.org/. Última consulta: 20/11/2020.

[22] Magento. *Sitio ofical*. Disponible en: https://magento.com/. Última consulta: 20/11/2020.

[23] Magento. *Magento DevBox*. Disponible en: https://devdocs.magento.com/guides/v2.3/extension-dev-guide/build/optimal-dev-environment.html. Última consulta: 20/11/2020.

[24] Docker. *Sitio oficial*. Disponible en: https://www.docker.com/. Última consulta: 20/11/2020.

[25] Tamara Techera, Pablo Ibañez y Marcelo Bruno. *Espacion GitLab*. Disponible en: https://gitlab.fing.edu.uy/marcelo.bruno/ParserModSecurity/-/wikis/home. Última consulta: 20/11/2020.

A ParserModSecurity

A.1. Archivos de salida CSV

Archivo de URLs transformadas

originalUrl	transformedUrl
/catalogsearch/result/?q=thishouldnotexistandhopefullyitwillnot	catalogSearchResult
/catalogsearch/result/?q=thishouldnotexistandhopefullyitwillnot	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/742/	addProductToCart
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/catalogsearch/result/?q=ZAP	catalogSearchResult
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/742/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/755/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/742/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/755/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/742/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/755/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/755/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/742/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/755/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/742/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/742/	addProductToCart
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/755/	addProductToCart
[illegible]	[illegible]

Figura A.1: Archivo que indica las URLs transformadas en una ejecución del Parser

Archivo de URLs no transformadas

UrlNotTransformed
/473288803632568536
/catalogsearch/7588092141987338570
/checkout/5137030121945907106
/checkout/cart/849597891555475555
/checkout/cart/add/2513377506180444548
/checkout/cart/
/checkout/cart/add/5534231380451680912
/checkout/cart/
/checkout/cart/add/uenc/8548778114984863769
/checkout/cart/
/checkout/cart/add/uenc/6729122757689946398
/checkout/cart/
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/3458079489047894500
/checkout/cart/
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/5016338408729080160
/checkout/cart/
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/3990640112931225115
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL2JvdHRvbXMtbWVuL3BhbnRzLW1lbi5odG1s/product/487348531068170953
/checkout/cart/
/checkout/cart/
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL3Nob3J0cy1tZW4uaHRtbA
/checkout/cart/add/uenc/aHR0cDovLzE3Mi4xNy4wLjMvbWVuL3Nob3J0cy1tZW4uaHRtbA
/checkout/cart/

Figura A.2: Archivo que muestra las URLs que no fueron transformadas en una ejecución del Parser

Archivo de URLs asociadas a reglas

transformedUrl	originalUrl	method	ruleId	ruleMsg	ruleData	ruleText	requestBody
catalogSearchResult	/catalogsearch/result/?q=ZAP	POST	942190	Detects MSSQL code execution and in	Matched Data: \x22!n found within ARGS:pro	Warning. Pattern match "(	product=ZAP+%251
catalogSearchResult	/catalogsearch/result/?q=ZAP	POST	942190	Detects MSSQL code execution and in	Matched Data: \x22!n found within ARGS:pro	Warning. Pattern match "(	product=ZAP+%251
addProductToCart	/checkout/cart/add/uenc/aH	POST	942190	Detects MSSQL code execution and in	Matched Data: \x22!n found within ARGS:pro	Warning. Pattern match "(	product=ZAP+%251
addProductToCart	/checkout/cart/add/uenc/aH	POST	942190	Detects MSSQL code execution and in	Matched Data: \x22!n found within ARGS:pro	Warning. Pattern match "(	product=ZAP+%251
addProductToCart	/checkout/cart/add/uenc/aH	POST	942190	Detects MSSQL code execution and in	Matched Data: \x22!n found within ARGS:pro	Warning. Pattern match "(	product=ZAP+%251
addProductToCart	/checkout/cart/add/uenc/aH	POST	942190	Detects MSSQL code execution and in	Matched Data: \x22!n found within ARGS:pro	Warning. Pattern match "(	product=ZAP+%251
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:produ	Warning. Matched phrase	product=%22%3E%
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:produ	Warning. Matched phrase	product=%22%3E%
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=755&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	932150	Remote Command Execution: Direct U	Matched Data: =\x22ls found within ARGS:u	Warning. Pattern match "(	product=755&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	932150	Remote Command Execution: Direct U	Matched Data: =\x22ls found within ARGS:u	Warning. Pattern match "(	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	932150	Remote Command Execution: Direct U	Matched Data: =\x22ls found within ARGS:u	Warning. Pattern match "(	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=755&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	932150	Remote Command Execution: Direct U	Matched Data: =\x22ls found within ARGS:u	Warning. Pattern match "(	product=755&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=755&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:produ	Warning. Matched phrase	product=%22%3E%
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:produ	Warning. Matched phrase	product=%22%3E%
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=755&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	932150	Remote Command Execution: Direct U	Matched Data: =\x22ls found within ARGS:u	Warning. Pattern match "(	product=755&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	932150	Remote Command Execution: Direct U	Matched Data: =\x22ls found within ARGS:u	Warning. Pattern match "(	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	932150	Remote Command Execution: Direct U	Matched Data: =\x22ls found within ARGS:u	Warning. Pattern match "(	product=742&uenc=
addProductToCart	/checkout/cart/add/uenc/aH	POST	941180	Node-Validator Blacklist Keywords	Matched Data: <!-- found within ARGS:uenc	Warning. Matched phrase	product=755&uenc=

Figura A.3: Archivo que muestra las URLs que están asociadas a la activación de reglas. De cada una se muestra la URL transformada, la original, el método, e información de la regla

A.2. Diagram de clases

Paquete Filter

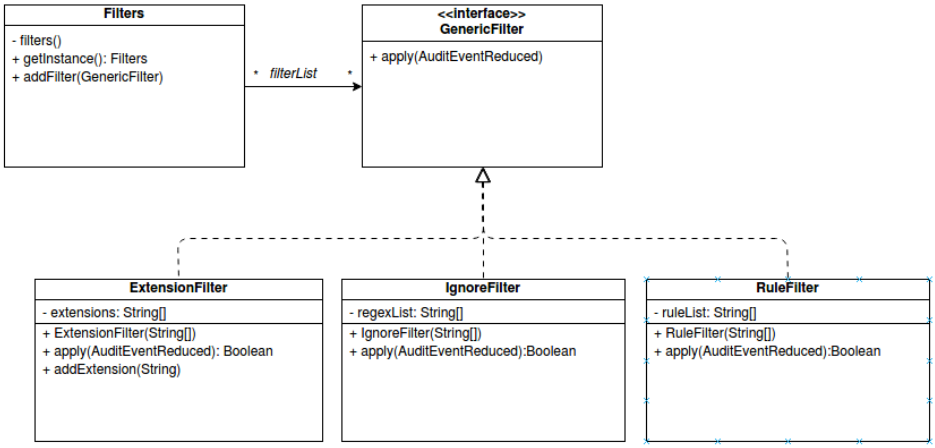

Figura A.4: Diagrama de clases del paquete Filter

Paquete Model

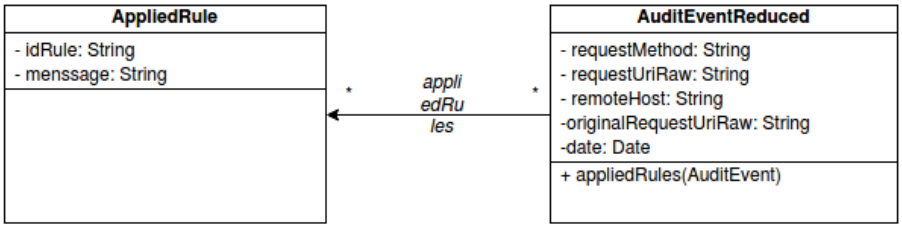

Figura A.5: Diagrama de clases del paquete Model

Paquete Parser

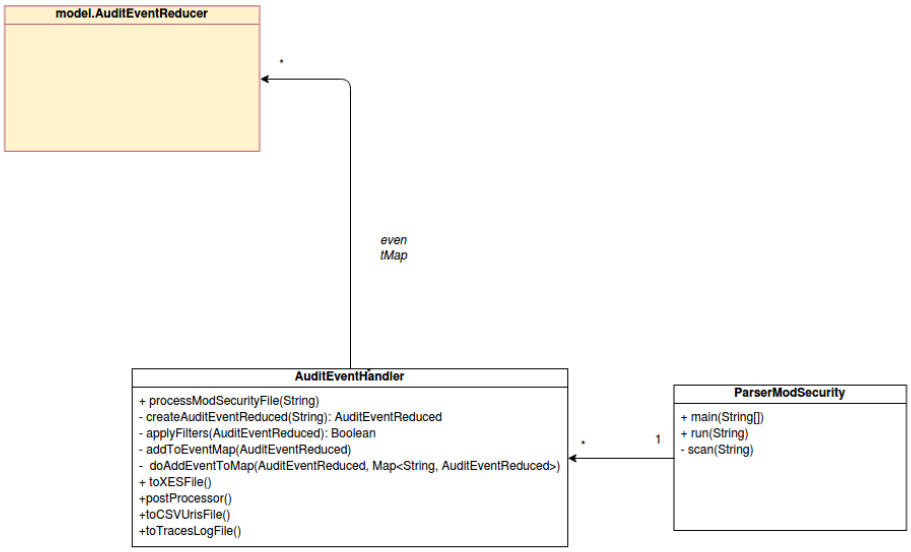

Figura A.6: Diagrama de clases del paquete Parser

Paquete Transformation

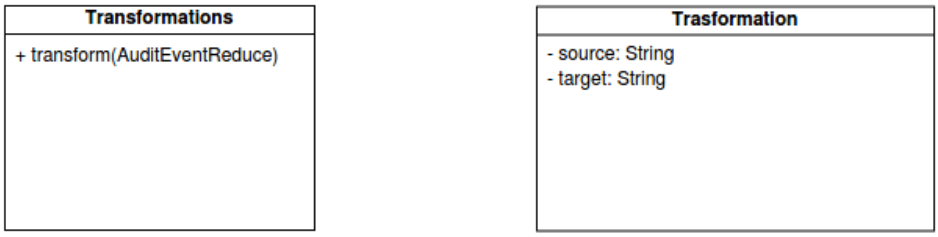

Figura A.7: Diagrama de clases del paquete Transformation

Paquete Util

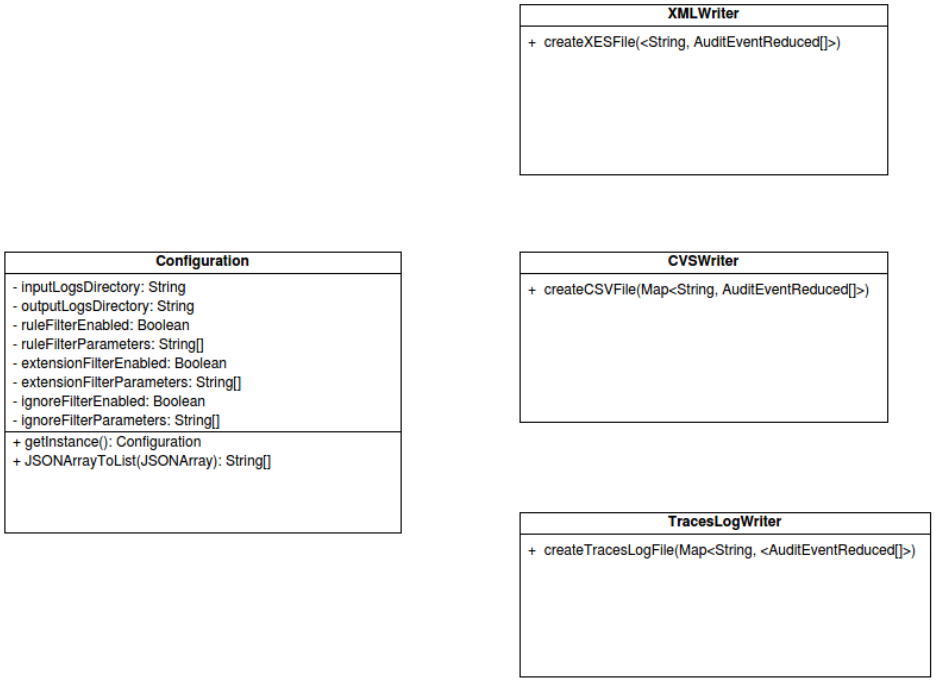

Figura A.8: Diagrama de clases del paquete Util

B

Agrupamientos y reglas

B.1. URLs activadas utilizando OWASP ZAP

URL	Reglas activadas
addProductToCart	920270, 921160, 921120, 930100, 930110, 930120, 932100, 932160, 933160, 932130, 941110, 942100, 941100, 941160, 941180, 932150, 942190
catalogSearch	920270, 921160, 921120, 930100, 930110, 930120, 932160, 933160, 932130, 941110, 941100, 941160, 942100, 941180, 932150, 942190, 980140, 952100, 959100

Tabla B.1: Raglas de ModSecurity activadas en los procesos críticos

Rule id	**Description**
920270	Invalid character in request (null character)
921120	HTTP Response Splitting Attack
921160	HTTP Header Injection Attack via payload (CR/LF and header-name detected)
930100	Path Traversal Attack (/../)
930120	OS File Access Attempt
932100	Remote Command Execution: Unix Command Injection
932130	Remote Command Execution = Unix Shell Expression Found
932150	Remote Command Execution: Direct Unix Command Execution
932160	Remote Command Execution = Unix Shell Code Found
933160	PHP Injection Attack = High-Risk PHP Function Call Found
941100	XSS Attack Detected via libinjection
941110	XSS Filter - Category 1 = Script Tag Vector
941160	NoScript XSS InjectionChecker: HTML Injection
941180	Node-Validator Blacklist Keywords
942100	SQL Injection Attack Detected via libinjection
942190	Detects MSSQL code execution and information gathering attempts
952100	PL1 error Java Source Code Leakage
959100	PL1 none Check of outbound anomaly score
980140	PL1 none Anomaly score correlation rule

Tabla B.2: Significado de las reglas activadas

B.2. Agrupamientos realizados en sitio web Magento

Actividad abstracta	**Expresión regular**
menuCategory	^\/what-is-new\\.html$
menuCategory	^\/women\\.html$
menuCategory	^\/men\\.html$
menuCategory	^\/gear\\.html$
menuCategory	^\/training\\.html$
menuCategory	^\/sale\\.html$
menuSubcategory	^\/women\/[a-z-]+\/?[a-z-]+\.html$
menuSubcategory	^\/men\/[a-z-]+\/?[a-z-]+\.html$
menuSubcategory	^\/gear\/[a-z-]+\/?[a-z-]+\.html$
menuSubcategory	^\/training\/[a-z-]+\/?[a-z-]+\.html$
menuSubcategoryFiltered	^\/women\/[a-z-]+\/?[a-z-]+\.html\?.*
menuSubcategoryFiltered	^\/men\/[a-z-]+\/?[a-z-]+\.html\?.*
menuSubcategoryFiltered	^\/gear\/[a-z-]+\/?[a-z-]+\.html\?.*
menuSubcategoryFiltered	^\/training\/[a-z-]+\/?[a-z-]+\.html\?.*
productDetail	/^\/[^\/]*\.html$
addProductToCart	^\/checkout\/cart\/add\/uenc\/[\w|%]+\/product\/[\d]{1,4}\/
catalogSearchResult	^\/catalogsearch\/result\/\?q=[\w\+]{1,150}$
catalogSearchAdvanced	^\/catalogsearch\/advanced\/result\/\?.+
makeReview	^\/review\/product\/post\/id\/\d+\/$
videoDownload	^\/downloadable\/download\/sample\/sample_id\/\d+\/$
promotions	^\/promotions\/.+\.html$
collections	^\/collections\/.+\.html$
collectionsFiltered	^\/collections\/.+\.html\?.+
promotionsFiltered	^\/promotions\/.+\.html\?.+
compareProducts	^\/catalog\/product_compare\/index\/items\/.+
addProductToCompare	^\/catalog\/product_compare\/add\/.+
removeProductToCompare	^\/catalog\/product_compare\/remove\/.+
clearCompare	^\/catalog\/product_compare\/clear\/.+
showProductRated	^\/catalog\/product\/view\/id\/\d+\/$
editItemFromCart	^\/checkout\/cart\/configure\/id\/\d+\/product_id\/\d+\/$
deleteAddress	^\/customer\/address\/delete\/id\/\d+\/form_key\/\w+$
editAddressRequest	^\/customer\/address\/edit\/id\/\d+\/$
editAddressSucceeded	^\/customer\/address\/formPost\/id\/\d+\/$
showReviewProductRated	^\/review\/customer\/view\/id\/\d+\/$
viewReview	^\/review\/product\/listAjax\/id\/\d+\/$
makeReview	^\/review\/product\/post\/id\/\d+\/$
salesReorder	^\/sales\/order\/reorder\/order_id\/\d+\/$
salesViewOrder	^\/sales\/order\/view\/order_id\/\d+\/$
salesOrderPrint	\/sales\/order\/print\/order_id\/\d+\/$
productSearch	^\/search\/ajax\/suggest\/\?q=[a-z\d]+&_\d+$
selectSizeOrColorProduct	^\/swatches\/ajax\/media\/\?product_id=\d+isAjax=(truelfalse).?
addAllWishlistToCart	^\/wishlist\/index\/allcart\/\?qty\[\d+\]=.+$
showWishList	^\/sales\/order\/view\/order_id\/\d+\/$
shareWishlist	^\/wishlist\/index\/share\/wishlist_id\/\d+\/$
updateWishlist	^\/wishlist\/index\/update\/wishlist_id\/\d+\/$
wishlistIndex	^\/wishlist\/index\/index\/wishlist_id\/\d+
catalogSearchAdvanced	^\/catalogsearch\/advanced\/result\/\?.+
estimateShippingMethods	^\/rest\/default\/V1\/guest-carts\/[a-z\d]+\/estimate-shipping-methods$
sendFriendProduct	^\/sendfriend\/product\/send\/id\/\d+\/$
productEspecification	^\/swatches\/ajax\/media\/\?isAjax=(truelfalse)_=\d+$
catalogSearchAdvance	^\/catalogsearch\/advanced\/\?.*$
checkoutCartUpdateItemOptions	^\/checkout\/cart\/updateItemOptions\/id\/\d+\/$
cartsPaymentInformation	^\/rest\/default\/V1\/carts\/mine\/payment-information\?_=\d+$
guestCartsPaymentInformation	^\/rest\/default\/V1\/guest-carts\/.+\/payment-information$
sendFriendProductMail	^\/sendfriend\/product\/sendmail\/id\/\d+\/$
guestCartShippingInformation	^\/rest\/default\/V1\/guest-carts\/.+\/shipping-information$
fbclid	\/?fbclid=.*$

Tabla B.3: Agrupamiento en entidades abstractas