Informe

Proyecto de grado

Definición y Enforcement de Políticas de Seguridad sobre Aplicaciones (DEPSA)

Tutores

Felipe Zipitría – Rodrigo Martínez

Rodrigo de la Fuente

Luis González

Juan Pírez

[Página dejada en blanco intencionalmente.]

Resumen

En la actualidad los sistemas de información forman parte de la vida cotidiana de las personas, brindando de forma sencilla la capacidad de manejar una gran cantidad de datos que permiten la realización de tareas que no serían posibles de otra forma.

Pese a la gran cantidad de información disponible sobre cómo evitar las vulnerabilidades de seguridad, los tiempos y costos del mercado llevan a los equipos de desarrollo a descuidar los aspectos referentes a la seguridad generando aplicaciones con vulnerabilidades, que permiten a usuarios acceder a información restringida o provocar la no disponibilidad del servicio. Los ataques provocan pérdida de confianza de los usuarios al sistema y económicas para las organizaciones, por lo que interesa encontrar una solución a las vulnerabilidades lo antes posible.

La solución recomendada por los especialistas en seguridad es la corrección de la vulnerabilidad, pero existen casos en los que esto no es viable, por ejemplo cuando se trata de aplicaciones legadas. Para estos casos puede utilizarse la técnica de *virtual patching*, que consiste en agregar una capa externa encargada de filtrar los datos que llegan y parten de la aplicación. Esta capa es capaz de detectar los ataques en tránsito y bloquearlos para impedir que sean exitosos.

Se busca investigar y diseñar un *framework* que asista en la tarea de configuración de herramientas de *virtual patching*, que requiere de conocimientos específicos en cada herramienta particular, para que un especialista en seguridad junto con un especialista en la realidad de la aplicación definan la política de seguridad sobre el modelo de la aplicación y que las herramientas realicen el *enforcement*. Este *framework* deberá aplicar la política de seguridad sin necesidad de modificar la aplicación para contemplar los casos en los que no es posible corregirla.

Al definir la política de seguridad sobre el modelo de la aplicación se permite incorporar los requerimientos de seguridad a los procesos de desarrollo existentes y que dicha política de seguridad sea empleada por varias herramientas de *virtual patching* en simultáneo.

En el proyecto se presenta una solución que permite a partir de la política de seguridad definida sobre el modelo de la aplicación, generar configuración para herramientas de *virtual patching* que realizan el *enforcement*, mitigando vulnerabilidades detectadas que necesitan una rápida solución.

Palabras clave: seguridad de aplicaciones, virtual patching, políticas de seguridad, modelo de la aplicación.

[Página dejada en blanco intencionalmente.]

Índice

1 Introducción....1
1.1 Caso de estudio....3
2 Objetivos....7
2.1 Objetivo general....7
2.2 Objetivos específicos....7
3 Estado del arte....9
3.1 Vulnerabilidades de software....9
3.2 Virtual patching....10
3.3 Trabajos relacionados con los objetivos del proyecto....14
3.4 Representación de políticas de seguridad....15
3.5 Funciones de mapeo de dominios....23
4 Análisis....27
4.1 Actores....27
4.2 Proceso de definición y enforcement de la política de seguridad....27
4.3 Políticas de seguridad....28
4.4 Especificación de requerimientos....31
4.5 Casos de uso....33
5 Diseño de la solución....37
5.1 Diseño de la arquitectura....37
5.2 Interfaces definidas....39
5.3 Decisiones de diseño....44
5.4 Lenguajes definidos....49
6 Implementación....57
6.1 Tecnologías utilizadas....57
6.2 Implementación de los lenguajes definidos....57
6.3 CRS de OWASP....59
6.4 Generación de reglas para ModSecurity....60
6.5 Creación de nuevos tipos de reglas....64
7 Conclusiones....67
7.1 Trabajos a futuro....68
Referencias....71
Glosario....75
Notas....79

[Página dejada en blanco intencionalmente.]

Índice de figuras

Figura 1 - Cantidad de personas que usan internet en el mundo....1
Figura 2 - Número de vulnerabilidades web detectadas....2
Figura 3 - Diagrama de casos de uso de LeagueManager....3
Figura 4 - Diagrama de clases de LeagueManager....4
Figura 5 - Filtrado de mensajes de virtual patching....11
Figura 6 - Firewall, IPS y web application firewall....12
Figura 7 - Tipos de diagramas de UML....16
Figura 8 - Fragmento del diagrama....17
Figura 9 - Role Based Access Control (RBAC)....18
Figura 10 - Metamodelo SecureUML, extraído de [28]....19
Figura 11 - FPSIVA para la clase Persona de LeagueManager....21
Figura 12 - Función de mapeo....23
Figura 13 - Catálogo de CDs expresado en XML....24
Figura 14 - Catálogo de CDs expresado en JSON....24
Figura 15 - Diagrama con el flujo de uso del framework....28
Figura 16 - Tipos de reglas....29
Figura 17 - Relacionamiento de los tipos de reglas....30
Figura 18 - Diagrama de casos de uso....33
Figura 19 - Diagrama de arquitectura....37
Figura 20 - Estructura de datos Mapping....39
Figura 21 - Estructura de datos SecurityPolicies....40
Figura 22 - Estructura de datos PreGeneratedPolicies....41
Figura 23 - Estructura de datos PreGeneratedSecurityPolicy....42
Figura 24 - Estructura de datos GeneratedPolicies....43
Figura 25 - Estructura de datos Generator....43
Figura 26 - Responsabilidades de los módulos....44
Figura 27 - Extensión del framework....44
Figura 28 - Patrón Strategy aplicado en DEPSA....45
Figura 29 - Clase abstracta que define las reglas de PreGenerator....45
Figura 30 - Clase abstracta que define las reglas de ModSecurityGenerator....46
Figura 31 - Estructura para calcular compatibilidades....46
Figura 32 - Patrón Template Method aplicado en DEPSA....47
Figura 33 - Adaptación de patrón Observer utilizada en DEPSA....48
Figura 34 - Ejemplo de lexer y parser de ANTLR....57

[Página dejada en blanco intencionalmente.]

Índice de códigos

Código 1 - Regla de ModSecurity....13
Código 2 - Función definida en la interfaz de Mappings....39
Código 3 - Función definida en la interfaz de SecurityPolicies....40
Código 4 - Funciones definidas en la interfaz de PreGenerator....41
Código 5 - Función definida en la interfaz de SpecificGenerator....42
Código 6 - Funciones definidas en la interfaz de Generator....43
Código 7 - Pseudocódigo de la función generatePolicies....49
Código 8 - Regla de EBNF....49
Código 9 - Definición de recurso en SPLang....50
Código 10 - Ejemplo de recursos de LeagueManager, expresados en SPLang....50
Código 11 - Regla ruleDefinition de SPlang....51
Código 12 - Regla dataTypeValidation de SPlang....51
Código 13 - Regla constraint de SPlang....51
Código 14 - Ejemplos de reglas Default definidas para LeagueManager....52
Código 15 - Blacklist para el atributo password de la clase Usuario....52
Código 16 - Sanitization para el atributo contrasena de la clase Usuario....52
Código 17 - Whitelist para el atributo edad de la clase Persona....53
Código 18 - Control de acceso para el atributo email de la clase Persona....53
Código 19 - Regla domain de MapLang....53
Código 20 - Ejemplos de recursos "sistema" para LeagueManager....54
Código 21 - Regla resourceMapping de MapLang....54
Código 22 - Regla relatedResource de MapLang....54
Código 23 - Definición de mapeos para LeagueManager....55
Código 24 - Ejemplo de regla de SPLang en ANTLR....58
Código 25 - Método visit del nodo Whitelist....59
Código 26 - Método syntaxError....59
Código 27 - Configuración de ModSecurity....61
Código 28 - Regla SecRule de ModSecurity....62
Código 29 - Ejemplo de regla SecDefaultAction de ModSecurity....62
Código 30 - Regla para el atributo Titulo....62
Código 31 - Regla Blacklist generada para ModSecurity....63
Código 32 - Regla Whitelist generada para ModSecurity....63
Código 33 - Regla Sanitization generada para ModSecurity....63
Código 34 - Regla que desactiva XSS para toda la aplicación....64
Código 35 - Regla que desactiva XSS sólo para el recurso password....64
Código 36 - Directorio donde agregar nuevas reglas a nivel PreGenerator....64
Código 37 - Directorio donde agregar la compatibilidad con las nuevas reglas....65
Código 38 - Directorio donde agregar nuevas reglas a nivel ModSecurityGenerator....65

[Página dejada en blanco intencionalmente.]

1 Introducción

Los sistemas de información son el soporte de las actividades económicas, sociales y culturales de la sociedad actual, donde el acceso a la información, su generación y manipulación tienen un papel fundamental. La sociedad de la información se ve impulsada por la implantación de las Tecnologías de la Información y Comunicaciones (TIC) que soportan la gestión y control de los servicios de emergencia, suministros de agua, redes eléctricas, apoyan la atención de la salud, los mercados financieros, la comunicación entre personas, entre otros.

ITU, la agencia de Naciones Unidas especializada en TICs, estima que a finales de 2015 más de tres mil millones de personas en el mundo usarán internet, lo que representa el 43,4% del total de la población mundial [1]. Como se puede apreciar en la Figura 1, el crecimiento de usuarios en internet es constante y en 2014 este aumento fue de un 6,6%. Cada vez más personas participan de la creación, el intercambio y la carga de contenido mediante aplicaciones basadas en internet, que brindan disponibilidad en cualquier momento y lugar.

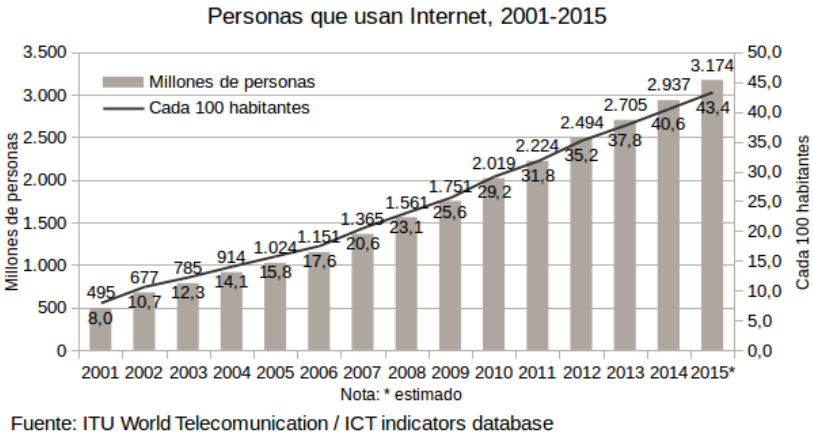


Figura 1 - Cantidad de personas que usan internet en el mundo

Los avances de las tecnologías web y el incremento en las velocidades de transferencia de datos por internet, permitieron mejorar la calidad de las interfaces que se le presentan a los usuarios así como también entregar más información, mejorando como consecuencia sus experiencias con los sistemas y aumentando así su popularidad. Tal es así que cada vez más tareas cotidianas, como realizar trámites con distintas organizaciones, se pueden efectuar vía internet sin necesidad de presentarse en el lugar.

Toda construcción de *software* implica la realización de tareas esenciales [2] como la especificación de las funcionalidades, el diseño de la solución y su implementación para que cumpla las especificaciones, la validación de que el *software* realiza lo que los *stakeholders* desean y la evolución mediante modificaciones para cumplir con nuevas necesidades. Estas actividades forman parte de los procesos de *software* e incluyen otras actividades como la validación de requisitos, diseño de la arquitectura del *software*, entre otras.

Para apoyar las tareas de análisis de los requerimientos y el diseño de aplicaciones [3], los desarrolladores emplean modelos en alto nivel desde etapas tempranas del proceso de desarrollo para detectar y evitar defectos en el *software*, debido a que el costo de detectar los defectos y repararlos en etapas posteriores tiende a aumentar. Los modelos también son útiles para el mantenimiento del *software* así como para tareas de reingeniería de sistemas. Incluso en los casos donde no se cuenta con el modelo, existen herramientas para generarlo automáticamente a partir de la implementación del sistema.

En la actualidad, al desarrollar una aplicación existe la urgencia de que sea puesta en producción en el menor tiempo posible. Por ejemplo, debido a que los interesados requieren de sus funcionalidades

para mejorar sus servicios y procesos, salir al mercado con el producto para aprovechar los nichos existentes o para posicionarse mejor que otras empresas. Al reducir los tiempos se descuidan tareas del proceso de desarrollo lo que puede causar una pérdida de la calidad del producto. Uno de los factores que determinan su calidad es la seguridad [4] y sus requisitos no suelen ser tomados en cuenta por la industria del software ya que su valor no es tangible para los inversores salvo cuando es comprometida. Incluso en los casos en que han sido considerados, los requerimientos de seguridad cambian a medida que pasa el tiempo por el descubrimiento de nuevas técnicas o ataques, así como el avance de la tecnología que brinda un mayor poder de cómputo para burlar los mecanismos existentes para brindar seguridad.

El acceso a la información respecto a las vulnerabilidades [5] y de cómo los desarrolladores pueden evitarlas es cada vez mayor, pero a pesar del esfuerzo, un gran número de aplicaciones sufren de serios problemas de seguridad. Según el último reporte de amenazas de seguridad en internet de Symantec [6], el número de vulnerabilidades detectadas luego de ser explotadas por un atacante en aplicaciones web continúa creciendo, como se aprecia en la Figura 2. Las consecuencias de los ataques incluyen cambios de los precios de los bienes de sitios comerciales, obtención de datos relevantes de usuarios como nombres y contraseñas, extorsión digital a través del bloqueo de computadoras, entre otros.

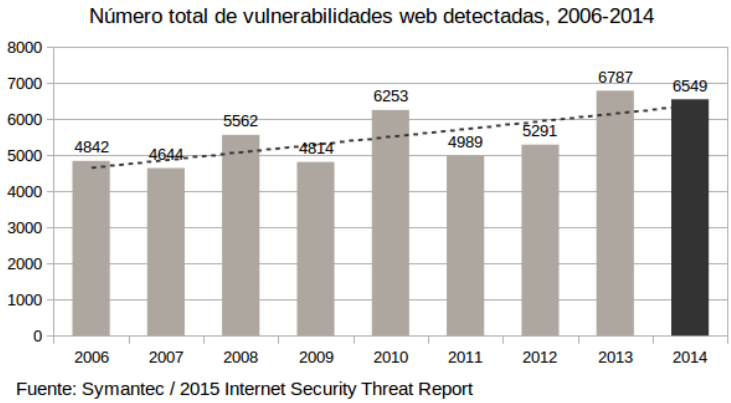


Figura 2 - Número de vulnerabilidades web detectadas

Cuando se detecta una vulnerabilidad en una aplicación, la solución recomendada por los especialistas en seguridad consiste en corregir el código, porque por más que se agregue una medida paliativa, el código vulnerable podría ser accedido por otro camino al detectado y ser explotado. Sin embargo existen diversas situaciones en la realidad de las organizaciones [7] que hacen necesaria una solución alternativa. Algunos ejemplos son:

- El *software* es realizado por terceros y sólo se dispone de los ejecutables por lo que es necesario esperar por un parche oficial.
- El equipo que desarrolló la aplicación no está disponible.
- La aplicación ya se encuentran en producción y no puede detenerse por su criticidad en el proceso de la organización.
- La aplicación es legada o puede estar desarrollada en una tecnología en la que ya no se cuenta con técnicos por lo que el tiempo y costo de detectar el error que provoca la vulnerabilidad y corregirla puede ser muy elevado.

En muchos casos reparar una vulnerabilidad requiere generar un nuevo proyecto de desarrollo porque implica detectar la vulnerabilidad, realizar revisiones del código fuente, realizar tests de penetración para identificar los problemas que puede haber generado dicha vulnerabilidad, para luego tomar las medidas necesarias para proteger la aplicación.

Por los tiempos que lleva reparar las vulnerabilidades detectadas, surgen técnicas alternativas a rediseñar o corregir los errores. Una de estas técnicas es *virtual patching* [8] que consiste en agregar

una capa externa, encargada de filtrar los datos que llegan y parten de la aplicación para detectar los ataques en tránsito y no permitir que estos sean exitosos. Si bien no se modifica el código de la aplicación para generar la protección necesaria, requiere de un equipo capacitado en configurar la herramienta y con conocimiento de la aplicación y sus vulnerabilidades.

En este contexto, donde la información es tan importante para las organizaciones y para los usuarios, se propone generar una herramienta que colabore en facilitar la tarea de proteger adecuadamente los datos, considerando las herramientas utilizadas para el desarrollo de software.

A continuación se presenta como caso de estudio, la aplicación LeagueManager, que se utilizará como ejemplo para mostrar los problemas de una aplicación genérica que colabore con la comprensión de los distintos temas abordados en este proyecto. En el capítulo 2 se describen los objetivos planteados para la realización de un *framework* que permita la definición y el *enforcement* de una política de seguridad. En el capítulo 3 se presentan las áreas de estudios relacionadas con los objetivos definidos. En el capítulo 4 se especifican los requerimientos que debe alcanzar la solución que se plantee para el proyecto. El capítulo 5 muestra las decisiones de diseño que definen los componentes de la solución para cumplir con los requerimientos definidos. En el capítulo 6 se describen las tecnologías particulares empleadas así como características del desarrollo del *framework*. Por último en el capítulo 7 se plantean los trabajos a futuro que se desprenden del proyecto y las conclusiones del mismo.

1.1 Caso de estudio

Para ser utilizada a lo largo del proyecto fue creada la aplicación LeagueManager para ejemplificar los conceptos expuestos, de manera de ayudar al lector en la compresión de la temática, y para comprobar que los resultados obtenidos con el proyecto cumplen con los objetivos planteados, expresados en la sección 2 Objetivos.

LeagueManager es una aplicación web que ofrece información sobre una liga deportiva. Entre la información que se brinda se encuentran los equipos, jugadores, partidos que se disputan y noticias de la liga, para permitir a los usuarios mantenerse actualizados sobre los aconteceres de la liga desde la comodidad de su hogar.

Además de permitir la consulta de información la aplicación cuenta con funcionalidades que permiten la gestión de los datos, para simplificar la tarea de mantener la información actualizada y por ejemplo disminuir los tiempos necesarios para la publicación de nuevas noticias.

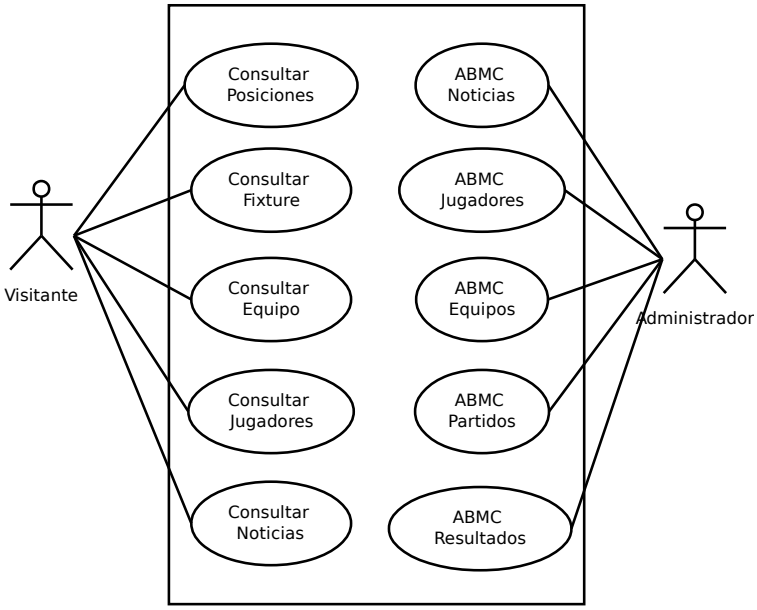


Figura 3 - Diagrama de casos de uso de LeagueManager

En la Figura 3 se presenta un diagrama con las funcionalidades que provee LeagueManager a los usuarios que visitan el sitio (nombrados como visitantes) y a los usuarios encargados de gestionar los datos contenidos.

Se puede observar en la Figura 4 el diagrama de clases de LeagueManager, con los atributos de las entidades así como las relaciones entre ellas. En este diagrama se ve la estructura con la que se realiza una representación de la realidad en el sistema.

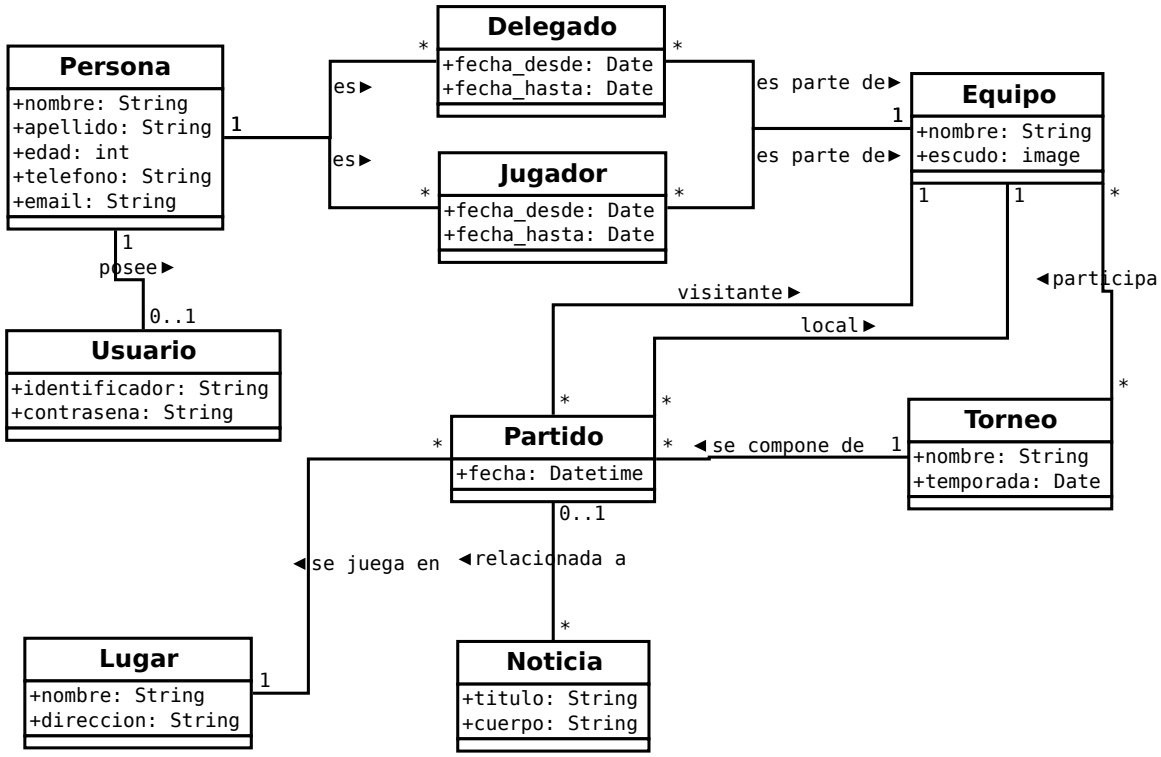


Figura 4 - Diagrama de clases de LeagueManager

Al realizar el desarrollo de *LeagueManager* no se tomaron en cuenta aspectos de seguridad debido a que se deseaba que contara con vulnerabilidades que fuera posible explotar para luego realizar la protección utilizando la técnica de *virtual patching*.

Luego de terminado el desarrollo fue posible comprobar que en LeagueManager un usuario no autorizado puede ingresar a las funcionalidades administrativas. Además como no se realiza un control sobre los datos ingresados, es posible inyectar código dentro de la aplicación para que sea ejecutado por los clientes.

Para realizar la protección se plantea la posibilidad de agregar dos *firewalls*, uno de aplicación web y otro de base de datos. Un *firewall* de aplicación web (WAF por sus siglas en inglés, *Web Application Firewall*) analiza los mensajes HTTP que llegan y parten de la aplicación, mientras que un *firewall* de base de datos analiza el flujo de datos entre la aplicación y la base de datos.

Para generar la configuración de cada *firewall* es necesario conocer el lenguaje en el que cada uno expresa las restricciones, que no solo implica un cambio en la sintaxis sino que además cambian los recursos sobre los que se aplican las restricciones. Por ejemplo en el caso del WAF, que se encarga de filtrar mensajes HTTP, los recursos son los parámetros que vienen en un pedido GET o POST para una cierta URL, pero para el caso del *firewall* de base de datos los recursos son las columnas de una tabla dentro de un esquema. Estas diferencias en las configuraciones implican que se deben generar todas las protecciones de forma duplicada pese a que lo que se está generando es la misma protección, sobre el mismo elemento del modelo de la aplicación. Por ejemplo si queremos que el

atributo identificador de la entidad Usuario solo se permitan letras, se debe validar mediante las herramientas de *virtual patching* correspondiente, el parametro identificador de la URL `/LeagueManager/Home.php` y la colmuna identificador de la tabla Usuarios en el esquema LeagueManager.

Se busca una herramienta que simplifique las tareas de configuración, permitiendo definir las restricciones sobre un lugar común y que genere las configuraciones adecuadas para los diversos *firewalls*. Un lugar común donde definir las restricciones es el modelo de la aplicación, pero como los elementos que se encuentran en este modelo no tienen una correspondencia trivial con los recursos sobre los que se aplican las restricciones de los *firewalls*, es necesario entonces que la herramienta provea de un mecanismo con el que se pueda explicitar la correspondencia de cada elemento del modelo con los recursos a los que se aplican las restricciones para un cierto *firewall*.

Esta herramienta buscada permitiría definir sobre el modelo de LeagueManager las políticas de seguridad necesarias para filtrar los posibles intentos de explotar las vulnerabilidades detectadas y luego generar las configuraciones necesarias para el WAF y el *firewall* de base de datos. Dicha definición se haría sin la necesidad de conocer las peculiaridades de los lenguajes de configuración particulares de cada *firewall*.

A continuación se presentan los objetivos que deberían cumplirse en la realización de un *framework* con las características planteadas en los puntos anteriores, describiendo también los objetivos específicos a ser alcanzados en el proyecto.

[Página dejada en blanco intencionalmente.]

2 Objetivos

Se presentan en esta sección las metas a cumplir en la realización de un *framework* que permita definir una política de seguridad sobre el modelo de la aplicación con vulnerabilidades, que esta política sea aplicada por herramientas de *virtual patching* y asegure la aplicación sin la necesidad de modificar su código fuente.

La sección comienza con una descripción general de las características con las que debe contar el *framework*, para posteriormente plantear los objetivos específicos que se buscan alcanzar con el proyecto actual.

2.1 Objetivo general

En el proyecto se busca diseñar un *framework* que permita a un especialista en seguridad junto con un especialista en la realidad de la aplicación, definir una política de seguridad sobre el modelo de la aplicación.

El *framework* debe aplicar la política de seguridad sin necesidad de modificar la aplicación y debe crearse un procedimiento iterativo e incremental que permita agregar o modificar restricciones en el futuro.

Una vez que la política de seguridad se encuentra expresada sobre el modelo, el *framework* podrá generar la configuración necesaria para que herramientas de *virtual patching* apliquen la política de seguridad definida sobre la aplicación.

Con el objetivo general definido se enumeran a continuación objetivos específicos para el proyecto.

2.2 Objetivos específicos

Los objetivos específicos a ser alcanzados en la realización del proyecto son:

- Utilizar un lenguaje que permita expresar políticas de seguridad usando como base el modelo de la aplicación.
- Expresar vulnerabilidades de software en el lenguaje utilizado para definir las políticas de seguridad.
- Utilizar un método que permita traducir los elementos del modelo de la aplicación donde se definirá la política de seguridad, y los dominios en el que trabajan las herramientas de *virtual patching* que son potencialmente distintos.
- Diseñar un *framework* que considere la posibilidad de agregar o modificar los tipos de restricciones a la definición de la política de seguridad, además de permitir extender o remplazar sus funcionalidades.
- Brindar la posibilidad de incorporar nuevas herramientas de *virtual patching* para generar su configuración e incluso para varias en simultáneo.
- Crear un prototipo, al que se denominará DEPSA, que permita validar la solución.

A continuación se presenta el estado del arte, en donde se presentarán tecnologías estudiadas que pueden ayudar a cumplir los objetivos.

[Página dejada en blanco intencionalmente.]

3 Estado del arte

En este capítulo se presentan las áreas de investigación estudiadas con el fin de conocer la actualidad en los temas relacionados con los objetivos.

Se comienza presentando el estudio realizado sobre las vulnerabilidades más comunes conocidas para el software y la forma de expresarlas y así utilizarlas en la definición de la política de seguridad.

El estudio continúa con una introducción a la técnica de *virtual patching* dedicando una atención especial a los *web application firewalls* [9], que serán los usados en la realización del prototipo y se presentan trabajos relacionados sobre herramientas que permiten asegurar aplicaciones sin modificar al código fuente.

Posteriormente se analizan las herramientas más utilizadas por los desarrolladores para expresar modelos, como es UML y sus extensiones, además de realizarse un estudio sobre la representación de políticas de seguridad dada la necesidad de que estas sean expresadas sobre el modelo de la aplicación. Se hará un especial hincapié en la representación de la validación de entrada debido al gran número de ataques que aprovechan este tipo de vulnerabilidades.

Se finaliza el estudio del estado del arte contemplando alternativas para la traducción entre los elementos del modelo de la aplicación y los dominios utilizados por las herramientas de *virtual patching*.

3.1 Vulnerabilidades de software

Se analizan el *top* 10 elaborado por OWASP [10] y el *top* 25 elaborado por CWE [11] para determinar cuales son actualmente las vulnerabilidades más relevantes en la industria del software y observar cómo expresarlas, para ser representadas al definir las políticas de seguridad.

Los listados de vulnerabilidades generados por OWASP y CWE pretenden educar sobre las consecuencias que pueden acarrear estos problemas, además de presentar formas de evitarlos.

Las vulnerabilidades presentadas en el top 10 de OWASP son:

1. ***Injection***: datos no deseados son ingresados al sistema en una instrucción válida y esta es ejecutada por un intérprete generando resultados no deseados.
2. ***Broken authentication and session management***: un atacante aprovecha fallas en el sistema de manejo de permisos logrando tener acceso a zonas que no le son permitidas, o robando la identidad de otro usuario.
3. ***Cross Site Scripting (XSS)***: la aplicación recibe código malicioso por parte de un cliente y lo transmite a otros clientes, permitiendo robar la sesión de las víctimas, redirigir a sitios maliciosos o realizar modificaciones en como el sitio es mostrado.
4. ***Insecure direct object references***: objetos de la aplicación que pueden ser accedidos por referencia directa, ya sea por medio de una URL o algún otro medio, a la vez que no cuentan con las validaciones de autenticación y autorización correspondientes.
5. ***Security misconfiguration***: problema con la configuración de la aplicación o servidor que pueden generar problemas de seguridad. Por ejemplo existencia de cuentas de usuarios por defecto, utilización de software sin actualizar o páginas no necesarias publicadas que pueden darle la posibilidad a un atacante de obtener acceso a recursos restringidos.

6. ***Sensitive data exposure***: datos que son especialmente sensibles no cuentan con un mecanismo para que no sean fáciles de interpretar en el caso de que un atacante logre acceso al contenedor de los datos.
7. ***Missing function level access control***: funciones que cuentan con validaciones de acceso a través de la interfaz o del lado del cliente y no repiten dichos controles del lado del servidor. De este modo si se consiguen burlar las validaciones del lado del cliente el servidor ejecutará las funciones aunque el usuario no tenga permisos.
8. ***Cross site request forgery (CSRF)***: fuerza al navegador de la víctima a realizar pedidos a un sitio enviando toda la información de autenticación de dicho usuario, para que el sitio considere que se trata de pedidos legítimos.
9. ***Using components with known vulnerabilities***: librerías utilizadas en los proyectos suelen ejecutar con todos los privilegios. Si una aplicación utiliza una biblioteca de la cual se conocen sus vulnerabilidades de seguridad, puede permitir a un atacante tener acceso con todos los privilegios ignorando las defensas implementadas por la aplicación.
10. ***Unvalidated redirects and forwards***: si un sitio web realiza redirecciones no verificadas, puede permitir a un atacante modificar estas redirecciones de manera de enviar a la víctima a un sitio falso o malicioso.

En el top 25 elaborado por la CWE las vulnerabilidades son categorizadas en:

- Interacción insegura entre componentes: vulnerabilidades detectadas que dependen de cómo la información es enviada y recibida entre componentes.
- Manejo riesgoso de recursos: vulnerabilidades relacionadas con la forma en que es manejada la creación, uso, transferencia o destrucción de recursos.
- Defensas permeables: relacionadas con las técnicas defensivas que son mal usadas, abusadas o inexistentes.

Se observa en ambos listados que para expresar las vulnerabilidades se les asigna:

- nombre, que la identifica.
- descripción, que permite conocer las causas de dicha vulnerabilidad.
- vector de ataque, que expresa la dificultad para explotar la vulnerabilidad.
- prevalencia de la debilidad, es una medida sobre qué tan común es encontrarla.
- detectabilidad, qué tan fácil es de detectar.
- impacto técnico, una medida sobre el daño que se puede realizar si se explota.
- evaluación, que expresa un resumen sobre las propiedades y el riesgo que implica.

A continuación se presenta *virtual patching*, que es una técnica para prevenir que las vulnerabilidades sean explotadas, intentando bloquear los ataques sin modificar el código de la aplicación.

3.2 Virtual patching

La técnica de *virtual patching* surge de la necesidad de encontrar una solución a las vulnerabilidades de seguridad de una aplicación, incluso en los casos en los que no se la puede corregir. Como se observa en la Figura 5, se agrega una capa exterior a la aplicación que analiza el

tráfico de mensajes desde y hacia la misma para prevenir que los ataques lleguen a su destino.

Entre sus características se destaca que un especialista en seguridad puede dar lineamientos y pautas a seguir al aplicar una herramienta de *virtual patching* sin ser experto en la tecnología empleada para desarrollar la aplicación. Además se puede utilizar *virtual patching* como parche temporal para frenar los ataques mientras se implementa la solución definitiva, reduciendo el tiempo de respuesta.

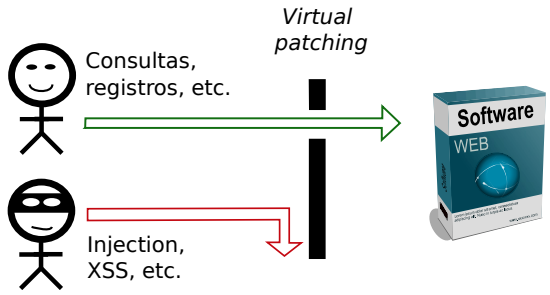


Figura 5 - Filtrado de mensajes de virtual patching

OWASP define un proceso de desarrollo de *virtual patching* [12], que sigue los lineamientos de la industria para responder a incidentes de tecnologías de información y que es útil para no tomar medidas apresuradas.

El proceso sugiere pasar por las siguientes fases:

- **Preparación**: Instalar y configurar la herramienta adecuada para implementar *virtual patching*.
- **Identificación**: Identificar las vulnerabilidades existentes en la organización de manera proactiva o reactiva. La primera se puede realizar mediante revisiones de código o contratando a un experto, mientras que la segunda se realiza luego de que ya se conoce una vulnerabilidad.
- **Análisis**: Obtener el nombre por el cual se conoce la vulnerabilidad comúnmente y las posibles consecuencias e impacto que surgen de explotarla, así como la versión y configuración del software que van a servir luego para solucionarla.
- **Creación**: Determinar lo que se desea hacer para la herramienta definida en la fase de preparación. Esto requiere especial cuidado para no bloquear el tráfico válido hacia la aplicación, es decir no generar falsos positivos. Se puede abordar este problema inicialmente sólo generando *logs* con información de los mensajes para luego pasar a bloquear el tráfico no deseado.
- **Implementación/testing**: Ejecutar la herramienta y probar que efectivamente se solucione el error. Es recomendado usar otras herramientas aparte de navegadores web para modificar el contenido de los mensajes de manera más sencilla, como clientes de línea de comando.
- **Recuperación/seguimiento**: Documentar la solución e incorporarla al proceso de reparación de vulnerabilidades cuando esto sea posible, y realizar revisiones periódicas para analizar cuando una solución antigua pueda ser removida, aunque en muchos casos resulten útiles los datos que genera y se desee conservarla.

Existe un gran número de herramientas que sirven para implementar *virtual patching*, que van desde hardware especializado en filtrar mensajes en la red del sistema, hasta software específico que se agrega a la infraestructura existente para realizar dicha tarea. Algunas de ellas son los sistemas de seguridad para redes de computadoras, conocidos como *firewalls* o cortafuegos, y sistemas de prevención de intrusos (IPS por su sigla en inglés, *Intrusion Prevention System*) [13].

Los IPS analizan el tráfico de los paquetes que circulan por la red, comparando los paquetes con patrones de ataques predefinidos en una base de firmas o detectando anomalías de lo definido como

tráfico normal. Es capaz de registrar estos ataques, notificarlos e incluso prevenirlos, descartando los paquetes de posibles ataques. Luego de inspeccionar un paquete en muchas ocasiones lo descarta para mejorar la *performance*, pero pierde el contexto de la comunicación con un cliente en particular, lo que implica que ciertos ataques no puedan ser defendidos.

Se pueden configurar estos sistemas a nivel de la red o en un *host* en particular, pero no son capaces de conocer los protocolos de comunicación de la aplicación que se desea proteger como si lo hace un *firewall* de aplicación.

Los *firewalls* se implementan tanto en hardware, software o ambos y son capaces de revisar el tráfico de la red y comprender los datos que viajan en los mensajes. Basándose en los datos toman decisiones sobre los mensajes que pueden llegar a ser potencialmente peligrosos para un sistema y bloquearlos. De acuerdo a su capacidad para comprender cada mensaje se los puede clasificar en primera, segunda y tercera generación. Los de primera generación son capaces de reconocer las direcciones de red y sus puertos para realizar el filtrado a partir de ellos. Los de segunda generación agregan control acerca del estado de la conexión entre los destinatarios finales de los mensajes y pueden recolectar varios mensajes antes de tomar una decisión. Los de tercera generación conocidos como *firewalls* de aplicación, además son capaces de reconocer los parámetros que se definen en los protocolos de red empleados por las aplicaciones.

Si bien la técnica de *virtual patching* posee las ventajas presentadas, requiere conocimiento y experiencia en el lenguaje particular de la herramienta a emplear para efectuar el *enforcement*.

3.2.1 Web Application Firewall

Los *Web Application Firewall*, son *firewalls* pensados específicamente para asegurar aplicaciones web. Al igual que un *firewall* tradicional, este dispositivo es configurado para permitir, limitar, cifrar y descifrar el tráfico entre dos subredes, aumentando la seguridad en las mismas. En particular WAF funciona a nivel de capa de aplicación, por lo que no solo es capaz de monitorear las direcciones de origen y destino, sino que también puede generar y aplicar reglas que validen el contenido de los mensajes, lo que aumenta su poder de control (ver Figura 6). Un WAF es capaz de detectar vulnerabilidades conocidas de aplicaciones web al identificar patrones que podrían derivar en un ataque de seguridad, como en el caso de Cross Site Scripting. También puede ser empleado para especificar los permisos que tiene un usuario para acceder a los recursos de una aplicación.

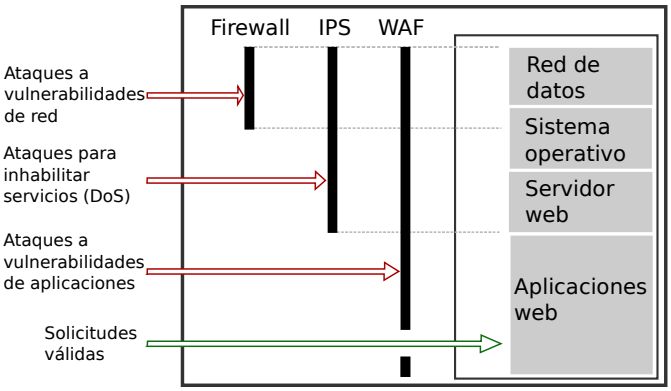


Figura 6 - Firewall, IPS y web application firewall

Las políticas de seguridad más utilizadas para la configuración de *firewalls* son la restrictiva (o germánica) en la cual se definen reglas indicando el tráfico conocido como adecuado y se prohíbe o registra todo lo demás, y la permisiva en la que se definen las reglas bloqueando específicamente lo que se considera incorrecto y se permite el resto. La política restrictiva suele ser considerada más segura, ya que es más difícil permitir por error tráfico potencialmente peligroso, mientras que en la política permisiva es posible no contemplar algún caso.

Uno de los WAFs que se ha transformado en un estándar de facto en la actualidad es ModSecurity [14], una herramienta de código abierto que puede ser modificada o extendida para cubrir

necesidades particulares. Sus principales características son su pasividad, flexibilidad, ser una herramienta predecible y priorizar entrega de soluciones de calidad antes que cantidad. La pasividad de ModSecurity se debe a que no toma decisiones, genera información y es el usuario quien decide qué se debe hacer y su flexibilidad es útil para los expertos de seguridad que pueden definir exactamente qué debe hacer en lugar de incorporar funcionalidades obligatorias. Puede ser empleado para monitorizar y controlar el acceso a las aplicaciones web en tiempo real, así como mitigar vulnerabilidades.

ModSecurity puede trabajar en dos modos: detección y prevención. En el primero se generan registros de la actividad del tráfico y es de utilidad para el administrador de la aplicación que podrá identificar los posibles ataques y sus objetivos, mientras que en el segundo, además de detectar los ataques, el WAF le agrega protección extra a la aplicación descartando mensajes que podrían ser una amenaza para la misma y retornar un mensaje de error adecuado al usuario.

ModSecurity es configurado mediante directivas [15] donde se especifican variables, operaciones y acciones, que serán agregadas a los archivos de configuración que determinan cómo actuará al procesar los mensajes. Las variables indican al argumento del mensaje al que aplica la regla, mientras que las operaciones son funciones que se aplican a esa variable y cuando su resultado evalúa a verdadero se deben aplicar las acciones definidas que expresan las medidas que debe aplicar el *firewall* con el mensaje. Por ejemplo en el Código 1, se expresa una regla que aplica al cuerpo de un pedido HTTP y en caso que este posea ciertos caracteres especiales, se debe descartar el mensaje.

```
SecRule     REQUEST_BODY      "@contains (abc|efg)"   "phase:2,log,deny,id:1"
____^____   ______^______     ________^________       ___________^___________

Directiva     Variables          Operaciones                  Acciones
```

Código 1 - Regla de ModSecurity

Se emplea la variable `REQUEST_BODY` para indicar que la regla aplica al cuerpo del mensaje, la operación `CONTAINS`, una expresión regular que determine los caracteres a filtrar y la acción `DENY` que detiene el procesamiento de la regla e intercepta la transacción. Si para cierta configuración no existe una acción específica, se toma la por defecto (`SecDefaultAction`).

Para detectar o prevenir una serie de vulnerabilidades comunes y conocidas, ModSecurity incluye un conjunto de reglas por defecto, pero su mayor utilidad reside en la posibilidad de generar reglas específicas que derivan de la propia aplicación. Además existe un proyecto en OWASP llamado Core Rule Set (CRS)[16] donde la comunidad genera y prueba un conjunto de reglas o configuraciones genéricas, para un conjunto de vulnerabilidades comunes y de interés general, por ejemplo, expresiones regulares que sirven para detectar cuando en una cadena de caracteres existe la posibilidad de un ataque de inyección de código SQL. Se puede ejecutar dicho conjunto de reglas en dos modos [17]: el tradicional donde se considera que cada regla es auto-contenida y no comparte lógica de detección con otras reglas, y por otro lado el modo de detección de anomalías con puntuación en el cual cada regla no aplica una acción disruptiva, sino que contribuye a un puntaje que se define por transacción. En cada transacción si la regla aplica, en lugar de bloquear, incrementa el puntaje de una categoría definida y si el puntaje acumulado supera un umbral que se puede personalizar, se bloquea. El primer método es el que se instala por defecto debido a que es más sencillo de comprender para el encargado de definir la seguridad y su *performance* es mejor ya que no necesita almacenar información de la transacción. Sin embargo cuando una regla del CRS genera falsos positivos, modificarlas es muy complejo y al hacerlo se pierden actualizaciones que la comunidad haga a las mismas. La ventaja de emplear el modo con puntajes es que reglas de

severidad baja pueden no significar una amenaza por sí solas para la aplicación, pero varias a la vez sí pueden alertar de un posible ataque.

La siguiente sección aborda los trabajos estudiados que presentan similares características a las definidas con los objetivos del proyecto.

3.3 Trabajos relacionados con los objetivos del proyecto

Se presentan aquí las características de SPECTRE [18] de David Scott y Richard Sharp, y MDSE@R [19] de Mohamed Almorsy, que son alternativas que abordan los objetivos propuestos para el proyecto.

3.3.1 SPECTRE

SPECTRE (*Security Policy EnforCement Through Run-timE Checks*) se trata de una herramienta diseñada para incorporar seguridad en las etapas de desarrollo de aplicaciones web, aunque también puede ser usada para solucionar vulnerabilidades de seguridad en aplicaciones existentes. El sistema de SPECTRE está compuesto por un compilador de políticas, un *gateway* de seguridad y un motor de inferencia.

- El compilador de políticas (*policy compiler*) traduce las restricciones de validación y transformaciones expresadas en SPDL, un lenguaje especializado para expresar políticas de seguridad, al código que será ejecutado por el *security gateway*. Las restricciones de validación definen patrones que deben cumplir los datos ingresados por el cliente para ser considerados como válidos y las transformaciones expresan funciones a aplicar sobre los datos ingresados.
- El *gateway* de seguridad (*security gateway*) se encarga de filtrar el tráfico de mensajes HTTP desde y hacia la aplicación forzando que se cumplan las restricciones impuestas por el compilador de políticas. Al realizar el filtrado sobre los mensajes HTTP, SPECTRE opera de forma independiente al código de la aplicación, por lo que no es necesario tener acceso al código fuente de la aplicación a asegurar.
- El motor de inferencia (*security-policy inference*) de SPECTRE infiere restricciones a ser aplicadas sobre los datos de la aplicación. El objetivo es aliviar la generación de código SPDL para una aplicación de gran tamaño, dadas las dificultades que esto puede generar.

SPECTRE presenta alternativas sobre cómo expresar políticas de seguridad pero al tener embebido el firewall encargado de filtrar los mensajes HTTP, genera una dependencia no extensible con la herramienta de virtual patching incluida. Adémas, para la configuración es necesario el conocimiento de SPDL el cual solo permite restricciones sobre URLs y parametros HTTP.

3.3.2 MDSE@R

MDSE@R (*Model Driven Security Engineering at Runtime*) promueve la definición y el estudio de la seguridad de una aplicación desde la etapa de diseño hasta cuando esta se encuentra en un ambiente de producción.

Se basa en dos conceptos claves, uno es separar el manejo y la ejecución de la seguridad de la aplicación, y el otro es el uso de modelos para describir el sistema y las propiedades de seguridad.

El objetivo de separar el manejo de la ejecución es no sobrecargar el sistema con la forma en que se define, se ejecuta o modifica la seguridad.

Para el uso de modelos se definen el *System Description Model* (SDM) y el *Security Specification Model* (SSM).

- El SDM es expresado utilizando un perfil de UML, en donde se capturan funcionalidades del sistema, componentes, clases, comportamiento y detalles de implantación.
- El SSM es desarrollado por especialistas en seguridad, capturando los objetivos, controles, requerimientos y arquitectura de seguridad del software a proteger. Es mantenido y actualizado mientras el sistema se encuentra en uso, para reflejar los cambios en el ambiente y los nuevos ataques existentes.

Usando el SDM y el SSM MDSE@R genera un modelo de seguridad, que es usado para inyectar extensiones de seguridad en la aplicación.

MDSE@R presenta una alternativa para expresar políticas de seguridad y el uso de modelos a ser utilizadas en la resolución de los objetivos del proyecto, pero al igual que SPECTRE tiene embebido el firewall encargado de filtrar los mensajes HTTP. Además es necesario definir todo el modelo de la aplicación y la política de seguridad utlizando su interfaz gráfica no permitiendo su integración con otras herramientas.

La siguiente sección introduce el concepto de políticas de seguridad y estudia las formas de representarlas a partir del modelo de la aplicación.

3.4 Representación de políticas de seguridad

Las políticas de seguridad son lineamientos en alto nivel para definir las reglas de comportamiento aceptables con respecto al acceso, el uso de los recursos y servicios de un sistema de información. En el glosario del TCSEC (*Trusted Computer System Evaluation Criteria*, del departamento de defensa de Estados Unidos de América) [20] se define política de seguridad como "El conjunto de leyes, reglas y prácticas que regulan cómo una organización administra, protege y distribuye información sensible".

Para definir una política de seguridad se debe evaluar la aplicación para determinar las posibles vulnerabilidades y el valor de lo que estamos asegurando, para que se considere el impacto de un incidente de seguridad en el sistema, que puede llegar a afectar la credibilidad, la reputación y las relaciones con los principales interesados. En la política de seguridad se debe definir qué se desea proteger y el por qué hacerlo, ya que es importante que indiquen cómo van a ser aplicadas y las consecuencias de que estas fallen.

Debido a los objetivos del proyecto las políticas de seguridad se definirán a partir del modelo del sistema, por esto se analizan las herramientas más utilizadas por desarrolladores para modelar sistemas, como UML (de su sigla en inglés *Unified Modeling Language*) [21] y sus extensiones que puedan servir para agregar los requerimientos de seguridad. Además como un gran número de ataques son causa de la falta de validación de las entradas al sistema [22], se profundizará cómo modelar la validación de entradas y por último se expone el lenguaje de políticas de seguridad empleado por SPECTRE.

3.4.1 Modelado de aplicaciones

Todo sistema de *software* debe ser estructurado de manera específica para cumplir las necesidades de la realidad contemplada. Por ejemplo grandes aplicaciones empresariales o aplicaciones móviles cuyo objetivo es brindar un servicio al mayor número de usuarios posibles, deben ser diseñadas contemplando su robustez en situaciones de estrés así como permitir escalabilidad, para que al

crecer el número de usuarios estos requerimientos no se vean comprometidos. Este diseño, conocido como arquitectura del sistema, debe ser lo suficientemente claro para que el sistema pueda ser desarrollado y corregido con mayor facilidad, incluso luego que las personas que lo desarrollaron ya no se encuentren en el proyecto. El modelado de los sistemas es útil para validar requerimientos de los clientes, así como para su posterior mantenimiento, tanto para aplicaciones grandes, medianas o pequeñas.

UML es el lenguaje estándar de la industria para especificar y modelar sistemas de software, mediante el cual ingenieros de sistemas, arquitectos de sistemas y desarrolladores pueden describir la estructura, el comportamiento, la arquitectura, los procesos de negocio y la estructura de datos del sistema. Esto se debe a que es un lenguaje definido de forma muy precisa por el *Object Management Group* (OMG) [23], un consorcio de estándares tecnológicos sin fines de lucro con miembros de todo el mundo, incluidos instituciones académicas, vendedores de aplicaciones y usuarios finales. OMG desarrolla estándares para una amplia gama de tecnologías e industrias.

Los diagramas de UML se pueden clasificar en diagramas de comportamiento y de estructura [24] como se observa en la Figura 7. Los primeros se emplean para visualizar, especificar, construir los aspectos dinámicos de un sistema, incluye los diagramas de actividad, de máquinas de estado, de casos de uso y los diagramas de interacción. Estos últimos son un subconjunto de diagramas de comportamiento que se centran en la interacción entre los objetos, como en el caso de los diagramas de comunicación, secuencia, tiempo e interacción. Los diagramas de estructura describen los elementos de la especificación que no están relacionados con el tiempo. Incluye los diagramas de clases, composición estructural, componentes, deployment, objetos y paquetes.

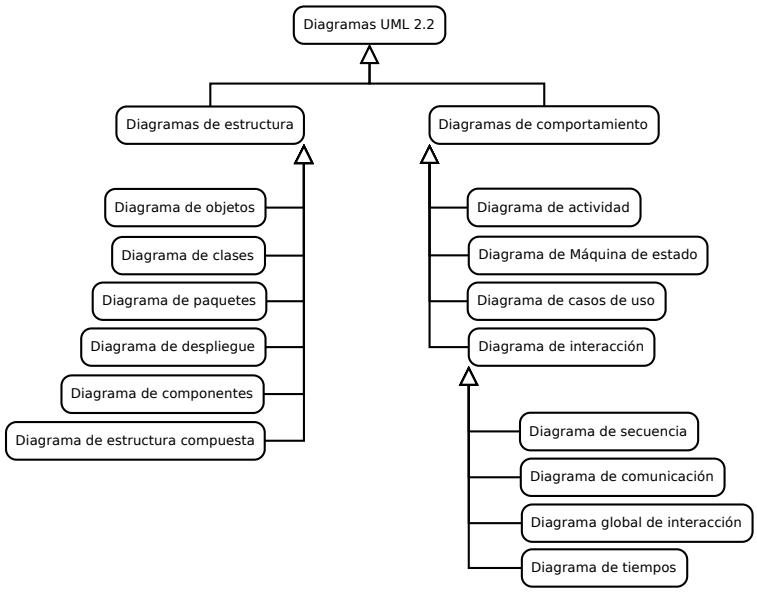


Figura 7 - Tipos de diagramas de UML

Existe un gran número de herramientas [25] basadas en la especificación de UML para realizar estos diagramas de forma gráfica, otras más complejas que analizan el código fuente existente o generan código a partir de los diagramas para algún lenguaje en particular, así como generan tests a partir del modelo. La Figura 8 presenta un fragmento del diagrama de clases de LeagueManager, con los conceptos que se definen como clases del lenguaje a emplear para implementar esta aplicación. En él se definen los atributos de cada clase y las relaciones que existen entre ellas. Es posible entender la realidad que se modela mediante el diagrama de UML, sin necesidad de conocer el lenguaje de programación que se emplee para desarrollar la aplicación. Por ejemplo se aprecia la entidad Persona de la que se conoce el nombre, apellido, la edad, el teléfono y un correo electrónico. A su vez una Persona posee o no un Usuario del que se registra un identificador y su contraseña (representado con 0..1 en la relación Persona-Usuario).

De la arquitectura del lenguaje UML [26] se destaca su extensibilidad, que permite nuevos dialectos llamados perfiles, con el propósito de particularizar el lenguaje para determinadas plataformas (*J2EE, .NET*) y/o dominios (por ejemplo finanzas, telecomunicaciones). También se permiten nuevos lenguajes o familias de lenguajes relacionados con UML, reutilizando algunos paquetes del mismo, definiendo meta-clases y meta-relaciones apropiadas.

Se distingue el paquete núcleo (*core package*) que contiene el metamodelo completo, diseñado particularmente para ser altamente reutilizable, y los perfiles que dependen del paquete núcleo, pero con particularidades de un lenguaje o dominio.

A continuación se detallarán extensiones de UML que sirven para modelar aspectos de seguridad como UMLSec [27], SecureUML [28] y UMLPac [29].

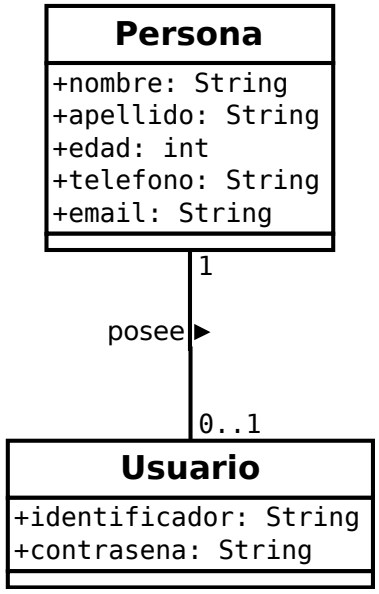


Figura 8 - Fragmento del diagrama de clases de League Manager

3.4.1.1 UMLsec

UMLsec es una extensión de UML que permite expresar información relevante a la seguridad del sistema mediante estereotipos, etiquetas y restricciones incluidas en UML y pueden ser empleados en diversos diagramas. Su principal utilidad es permitirle a los desarrolladores en una notación ampliamente usada expresar conceptos estándares de seguridad y su especificación puede servir para realizar chequeos formales que evalúan si los diagramas presentan posibles vulnerabilidades.

En los diagramas de clases se emplea UMLsec para asegurar que el intercambio de datos obedece al nivel de seguridad definido. Para especificar el nivel de seguridad alto se emplea la etiqueta (*tag*) "*{high}"* sobre los elementos del modelo, es decir a los atributos, parámetros de las operaciones o sus retornos que se desea definir el nivel de seguridad alto. Si no se define la etiqueta se considera que el nivel de seguridad es bajo.

Incorporando UMLsec a los diagramas de máquinas de estados se previene el flujo indirecto de datos, sobre todo en sistemas de objetos distribuidos. Se define que un objeto preserva seguridad si en el diagrama de máquinas de estados no existe un valor de salida con nivel de seguridad bajo que dependa de un valor de entrada con nivel alto de seguridad.

En los diagramas de interacción se define el intercambio de mensajes entre los objetos y con UMLsec se busca garantizar la seguridad en la interacción entre los objetos. En particular se modelan los protocolos de cifrado de datos y otros aspectos de seguridad para sistemas de objetos distribuidos, donde los mensajes son enviados por redes de computadoras y pueden ser interceptados, modificados o eliminados.

UMLsec destaca que mediante las extensiones de UML se puede representar conceptos estándar de métodos formales de seguridad así como definir posibles evaluaciones sobre los modelos, para evitar vulnerabilidades en los sistemas.

Otra extensión con similares características es SecureUML, diseñada para integrar la información relevante al control de acceso junto al modelo de la aplicación, que se presenta en la siguiente sección.

3.4.1.2 SecureUML

El objetivo de SecureUML es generar una nueva herramienta que dé soporte al desarrollo de seguridad en aplicaciones, y plantear una metodología para modelar las políticas del control de acceso y su integración al proceso de desarrollo, que sirva como guía para aquellos desarrolladores que no tengan la suficiente experiencia.

Consiste en extender UML para especificar información sobre el control de acceso dentro del diseño del sistema y como esta puede ser usada para automáticamente generar los accesos, ya que gracias a su notación visual da la posibilidad de realizar diseños con un alto nivel de abstracción. Al estar basado en UML permite que incluso los desarrolladores sin un gran conocimiento de seguridad puedan comprender de manera sencilla aspectos de seguridad, por estar adecuados al lenguaje y de esta manera construir sistemas seguros.

SecureUML se basa en el modelo de acceso de control RBAC (Role Based Access Control, Figura 9) [30], donde se definen permisos que consisten en las acciones u operaciones que puede ejecutar un usuario sobre un objeto del sistema. Para desacoplar los usuarios de los permisos, se introduce el concepto de rol, que refiere a la tarea o función que cumple un usuario en una organización, permitiendo definir una jerarquía de roles, donde un rol r_1 hereda de r_2 todos sus permisos. A los usuarios (u otro software) se le asigna un conjunto de roles, y a estos roles se les definen los permisos de acceso, que son un conjunto de operaciones sobre un objeto en particular. De esta manera el control de acceso se efectúa cuando un usuario intenta realizar una operación sobre un objeto y se controla que dicho usuario contenga alguno de los roles que poseen permisos de acceso para ejecutar la operación deseada sobre el objeto.

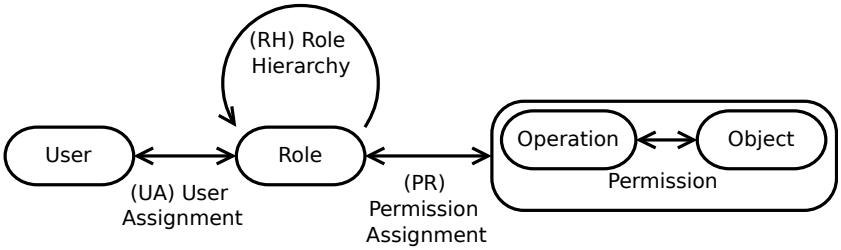


Figura 9 - Role Based Access Control (RBAC)

SecureUML define un perfil de UML para modelar los distintos aspectos que forman el modelo RBAC y se incluye además el estado del sistema, es decir el estado del objeto protegido, sus valores o una fecha y tiempo específicos. Como muestra la Figura 10, el meta-modelo definido expresa los objetos protegidos mediante el elemento estándar de UML, como el concepto *ModelElement*, permitiendo que cualquier elemento de un modelo UML sea un objeto protegido. También se define el concepto *ResourceSet* que representa las restricciones de permisos y autorizaciones que el usuario indique, como un conjunto de elementos del modelo. La semántica de los Permisos se define mediante el concepto *ActionType*. Cada *ActionType* representa una clase de operaciones relevantes a la seguridad sobre un tipo particular de objeto, como leer, escribir, modificar, borrar, etc. El concepto *ResourceType* define los tipos de acciones posibles para un tipo de meta-modelo en particular. *AutorizationConstraint* deriva del tipo *Constraint* del núcleo de UML, expresa las precondiciones que aplican a cada llamada de un método particular de un objeto, por ejemplo para modelar cuando un método puede ser invocado en un rango horario, y se lo invoca fuera de este horario, el sistema deberá lanzar una excepción de acceso denegado.

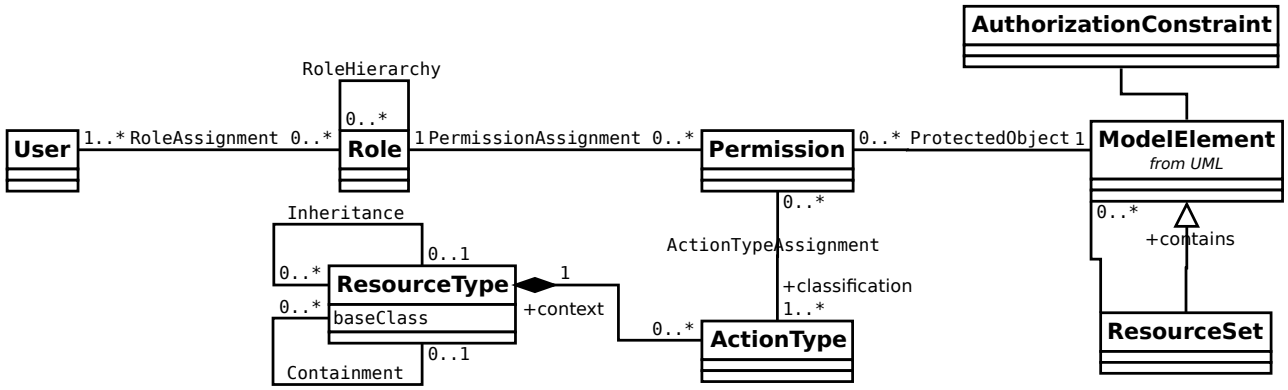


Figura 10 - Metamodelo SecureUML, extraído de [28]

Se destaca nuevamente las ventajas de este enfoque donde los requerimientos de seguridad pueden ser expresados con un alto nivel de abstracción.

Por último se presenta UMLpac, una extensión de UML que sirve para representar aspectos de seguridad en un sistema en forma de paquetes UML.

3.4.1.3 UMLpac

UMLpac puede ser empleado en diagramas de clases del sistema, manteniendo el nivel de abstracción. En los paquetes de UML se definen las amenazas de seguridad mediante entradas de catálogos y sus correspondientes técnicas de prevención y se puede complementar con SecureUML y UMLsec que se enfocan más en el control de acceso de una aplicación, permitiendo generar un paquete que contenga un descriptor de los principios de seguridad definidos en estas extensiones antes desarrolladas.

Se emplea el estereotipo de paquetes para representar cada aspecto de la seguridad, con tres atributos: *Risk Factor*, *Security Tile*, *Security Descriptor*. El factor de riesgo (*Risk Factor*) indica que tan probable es que se produzca un ataque para ese paquete de seguridad. *Security Tile* es el lugar donde los analistas en seguridad especifican como un determinado paquete protege una determinada parte del sistema. *Security Descriptor* determina una categoría específica que protege una parte en particular del sistema. El tipo del *Security Descriptor* identifica qué información se encuentra en el *Security Tile*.

UMLpac permite generar combinaciones y conjuntos de paquetes, para una mejor visualización del diagrama. De esta manera, se puede definir de forma abstracta un paquete que agrupe un conjunto o la combinación de paquetes lo que simplifica la apariencia del diagrama.

Para indicar que parte del sistema es asegurado por un determinado paquete, se emplea la asociación con el estereotipo `<<protects>>`. Este estereotipo también es empleado para indicar las entradas y salidas de un paquete, mostrando también quién es el responsable de recibir o enviar la salida/entrada.

Si bien UMLpac introduce características destacables como el factor de riesgo y la definición de paquetes con los requerimientos de seguridad, no fue posible hallar el perfil de UML que lo defina completamente.

3.4.2 Validación de entradas

Consiste en verificar y filtrar los datos externos al sistema antes que sean usados. Es parte crítica de

la seguridad de los sistemas dado que al ingresar datos que no son validados, un atacante puede llevar al sistema a condiciones impredecibles y explotar esto para su propio beneficio. Estos datos también pueden producir resultados inesperados y ser analizados por un atacante para lograr su cometido. En general la validación de entradas no es un tema cubierto en la etapa de diseño del software por lo que genera muchas vulnerabilidades que suelen ser críticas para las aplicaciones, especialmente en las aplicaciones web.

En el trabajo planteado por Pedram Hayati et al [22], proponen incorporar la validación de entradas a los diagramas de casos de uso, de clases, de secuencia y actividades, mediante la extensión de UML con un nuevo framework para ingeniería de seguridad basada en modelos, como guía hacia un camino ideal para diseñar software seguro.

Se propone que antes de usar los datos en la aplicación, estos deberán estar completa y adecuadamente validados por los mecanismos de validaciones. Si estos mecanismos de validación no son correctos, se rechazarán datos de entrada correctos, lo que es muy molesto para los usuarios, así como datos incorrectos pueden ser usados por la aplicación, generando vulnerabilidades.

La integración de ingeniería de seguridad al desarrollo de software basado en modelos presenta ventajas en cuanto a que los requerimientos de seguridad pueden ser integrados y formulados a un alto nivel de abstracción y que la aplicación sea pensada para evitar que ciertas políticas de seguridad sean violadas. El modelo puede ser empleado para validar la correctitud de los requerimientos de seguridad y su realización en diseño, ahorrando además presupuesto.

Se plantean tres modelos de validaciones para la validación de entradas:

1. Lista de valores no aceptables (*blacklist*): rechaza los datos incorrectos, en base a una lista de valores a ser rechazados y acepta los restantes.
2. Lista de valores aceptables (*whitelist*): acepta sólo los valores correctos basado en cinco atributos primarios de seguridad de validación de entradas (FPSIVA, *five primary security input validation attributes*) que son el tipo de dato, el largo, un conjunto de caracteres, formato y usos razonables.
3. Sanitizando la entrada (*sanitization*): al sanitizar los datos de entrada, las posibles amenazas dejan de serlo para el sistema.

El primer enfoque requiere que el desarrollador sepa de antemano cuales son todas las posibles entradas a invalidar, y esto se ve afectado cuando el atacante varía la forma de hacerlo, quedando el software vulnerable. El tercero posee la misma desventaja porque para poder sanitizar la entrada, se requiere conocer los posibles ataques. Cuando el desarrollador no conoce todos los tipos de ataques no podrá definir una lista de datos a sanitizar. Sin embargo en el segundo enfoque, el desarrollador se centra en los datos buenos y seguros para la aplicación. Se determina los cinco atributos para cada campo de entrada y en caso de no cumplir alguno de ellos, los datos son rechazados.

Los cinco aspectos empleados en FPSIVA son muy importantes. En cuanto al tipo, se refiere al tipo de dato esperado, que puede ser un valor entero, una cadena de caracteres, un valor booleano, etc. El formato indica la sintaxis de como se representa el dato ingresado mientras que el largo sirve para limitar la cantidad de caracteres ingresados. El conjunto de caracteres son los números, alfabetos, símbolos y sus combinaciones, que limita los caracteres dentro del tipo seleccionado. El último atributo sirve para determinar qué dato es razonable y cuál no, a partir de su semántica y es muy útil para prevenir futuros ataques que aún no son conocidos. Todos estos atributos los puede definir un desarrollador sin ser un experto en seguridad, lo cual es una ventaja.

El uso de validación de entrada puede ser empleado en los diagramas de caso de uso, donde se

especifican los requerimientos de seguridad así como en los diagramas de secuencia o de actividad. En particular se destaca el uso de la validación de entradas en los diagramas de clases mediante los mecanismos de estereotipo y restricciones. Para cada clase que recibe una entrada del entorno, se define una clase de restricciones de validación de entrada (IVC de su sigla en inglés *Input Validation Constraints*). En esta clase se definen los 5 atributos de FPSIVA, para cada parámetro de la clase que recibe la entrada y estas restricciones deberán ser empleadas a la hora de validar el ingreso de datos. Como IVC deriva del tipo restricción del núcleo de UML, se puede emplear la asociación estándar de UML para relacionar ambos elementos.

Como ejemplo en la Figura 11 se definen los cinco atributos primarios de seguridad de validación de entradas para la clase Personas de LeagueManager. Para el atributo "edad" se define que es del tipo Integer (número entero), el largo máximo del atributo es 2 dígitos. Se detalla el conjunto de posibles caracteres [0-9] que son los dígitos y mediante una expresión regular o patrón se define el formato aceptable de ese atributo que en este caso consiste en los dos dígitos. Además se listan posibles valores razonables de uso, por ejemplo, su valor es mayor que 0 y menor que 99. Se destaca que la definición de buenas expresiones regulares para los datos correctos o legítimos, es de mucha utilidad para disminuir la posibilidad de fallas que provoquen vulnerabilidades. Además el enfoque presentado posee la ventaja de un alto nivel de abstracción para definir los requerimientos de seguridad y resuelve el problema de falta de background de seguridad de los desarrolladores.

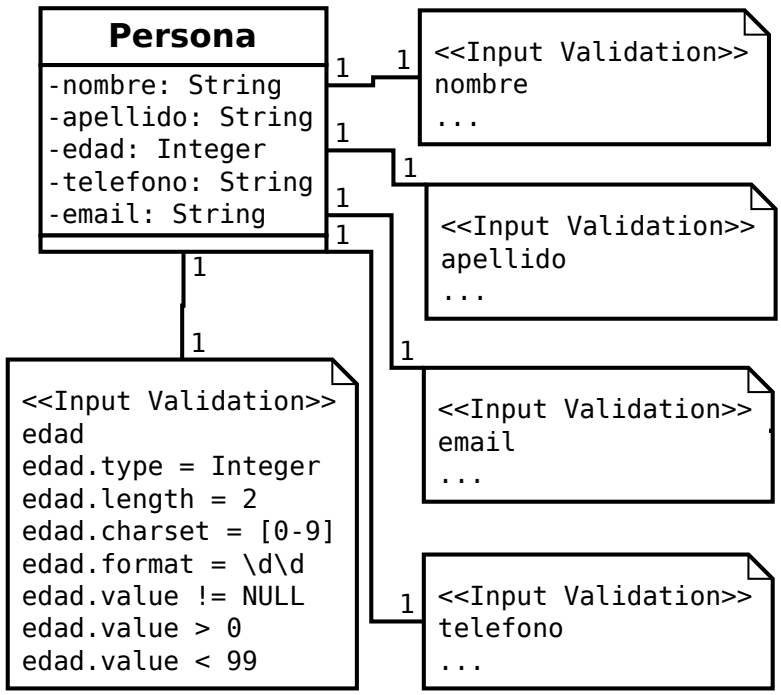


Figura 11 - FPSIVA para la clase Persona de LeagueManager

Para finalizar el estudio de cómo representar políticas de seguridad, en el capítulo siguiente se describe SPDL, el lenguaje empleado para describir políticas de seguridad de SPECTRE.

3.4.3 SPDL

Debido a que las vulnerabilidades cambian, los lenguajes para definir políticas de seguridad deben estar bien definidos y ser flexibles, para poder incorporar nuevas políticas o nueva información a las ya existentes. David Scott y Richard Sharp presentan un sistema que está formado por un lenguaje de descripción de políticas de seguridad y una aplicación (SPECTRE) que transforma las políticas de seguridad definidas en reglas para un firewall de nivel de aplicación. Con este enfoque se busca quitarle la mayor cantidad de responsabilidades posibles a los programadores e incluir lo relativo a la seguridad de la aplicación a un nivel de abstracción mayor. En esta sección nos centraremos en el lenguaje SPDL (Security Policies Description Language) [31], que sirve para generar políticas de seguridad independientes del lenguaje en que esté implementada una aplicación a la cual se desea

proteger.

La especificación de SPDL es mediante un documento XML y cada política de seguridad contiene una serie de URLs y *cookies*. Para cada URL se declara un conjunto de parámetros, que son los campos que se envían al usuario de la aplicación. Para cada uno de estos parámetros, se define un conjunto de atributos que son los que generan las restricciones sobre los datos pasados mediante el parámetro. Estos atributos son: el largo máximo y mínimo, si el parámetro es obligatorio, el tipo del mismo, es decir si es un entero, un valor real, una cadena de caracteres y si son campos que se deben garantizar que no sean modificados, para ello se le asocia un código de autenticación del mensaje (MAC). Además se define un atributo que especifica el método al cual aplica la política, que puede ser `GET` cuando el parámetro es enviado al usuario de la aplicación, `POST` cuando es enviado desde un formulario desde el cliente a la aplicación, o `GETandPOST` cuando la política aplica para ambos casos.

Si bien se espera que con los atributos antes mencionados, se puedan expresar la gran mayoría de las restricciones y validaciones a emplear, para los casos que se requiera más detalle, se especifica un lenguaje de validación. En él cada elemento es una expresión de validación que es un subconjunto de la especificación de Standard ML [32]. Una expresión de validación bien formada debe evaluar al valor booleano *true* si los datos del parámetro son correctos, y *false* cuando se trate de una falla de validación. Se permite referenciar otro parámetro definido antes, para los casos donde en una regla, el valor de un parámetro depende del otro. Cuando son declarados para un método `POST`, se indica el parámetro mediante `postparam.nombre`, y si es un método `GET` mediante `getparam.nombre`.

Para las expresiones de validación se define un conjunto de funciones primitivas y operadores binarios de los cuales se destacan:

- Operadores aritméticos (+, *, /, -) que funcionan para enteros o reales.
- Operadores sobre cadenas de caracteres, como concatenar dos cadenas de caracteres (`s1 + + s2`), `format(s, expreg)` que retorna verdadero si la cadena de caracteres `s`, cumple con la expresión regular `expreg`. Funciones que retornan un substring de una cadena de caracteres, y de conversión entre tipos, como por ejemplo, `String.fromInt(i)`, que retorna una cadena de caracteres que representa el valor entero `i`.
- Funciones sobre los parámetros, como por ejemplo para chequear si un parámetro `p` está definido: `isdefined(p)`.

La etiqueta `<transformation>` se emplea para identificar una secuencia de transformaciones. Si están anidadas dentro de un parámetro, indican las transformaciones a aplicar, una a continuación de la otra, al dato recibido en dicho parámetro. Se definen previamente en una librería y alguna de ellas son para reemplazar caracteres especiales por su código HTML, brindando facilidades para que los usuarios puedan definir sus propias transformaciones y agregarlas a la librería. Este tipo de transformaciones son de particular utilidad, sobretodo en casos como los de *Cross Site Scripting*, por lo que se considera por defecto que todos los parámetros requieren de la misma, salvo que se indique lo contrario explícitamente.

Con SPDL se pone de manifiesto las ventajas de definir un lenguaje para la definición de las políticas de seguridad, en lugar de generar directamente las reglas para la herramienta de *virtual patching*. Al emplear el lenguaje específico para definir las políticas de seguridad, se gana en mantenimiento, claridad e incluso la posibilidad de reutilizar el código, reduciendo el tamaño del mismo. Si bien la flexibilidad de un lenguaje de propósito general es mayor, le deja al encargado de la seguridad mayores responsabilidades, tanto para la definición de las políticas como para su

enforcement.

SPDL tiene un enfoque orientado a la capa web y *firewalls* de aplicación web. Si bien permite expresar las políticas de seguridad en alto nivel, se definen las reglas para cada recurso (URL o *cookie*) y sus atributos, y no sobre los recursos del modelo en sí, por lo que se pierde la abstracción de definir la regla una vez en el modelo y que se aplique en todos los lugares que corresponda. Además si se requiere definir las reglas para otro dominio como puede ser base de datos, se debería modificar el lenguaje para que permita la definición de recursos de este nuevo dominio y diferenciarlos de los anteriores.

La siguiente sección introduce el estudio de como relacionar los elementos del modelo de la aplicación con los de los dominios donde se aplican las políticas de seguridad en las diferentes herramientas para aplicar *virtual patching*.

3.5 Funciones de mapeo de dominios

Como las políticas de seguridad se definen en el modelo de la aplicación y las restricciones en los dominios visibles por las herramientas de *virtual patching*, se necesita una función que permita relacionar los elementos entre dichos dominios como puede verse en la Figura 12.

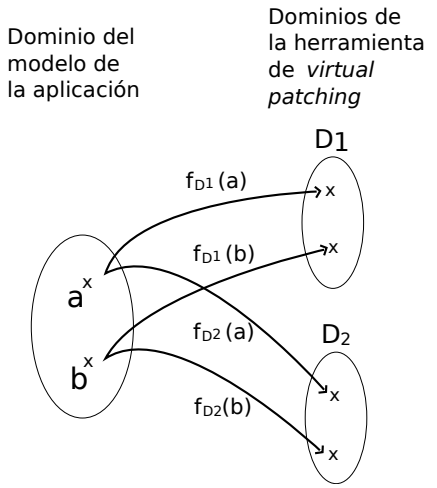


Figura 12 - Función de mapeo

Es de interés investigar los lenguajes y métodos que permiten relacionar las políticas de seguridad indicadas por los especialistas de seguridad que utilicen el *framework*, con los elementos que se encuentran en los dominios que manejan las distintas herramientas de *virtual patching*.

Además se investigan aquellos que se utilizan para describir las aplicaciones, sus interfaces y las bases de datos, ya que los mismos son potenciales dominios a considerar como codominio de la función de mapeo.

Se comienza describiendo lenguajes altamente flexibles que se pueden utilizar para relacionar elementos de distintos dominios como XML [33] y JSON [34]. Se continúa con WADL [35] y WSDL [36] que describen aplicaciones y finalmente se presentan trabajos relacionados con la problemática de mapear elementos de distintos dominios.

3.5.1 Lenguajes para el intercambio de información

XML (eX*tensible Markup Language*) es un lenguaje flexible en formato de texto que permite relacionar elementos de distintos dominios, es derivado de SGML (*Standard Generalized Markup Language*). Su principal función es la de almacenar datos en forma legible y permite definir la gramática de ciertos lenguajes. Es estandarizado y ampliamente aceptado debido a que es desarrollado y mantenido por la W3C (*World Wide Web Consortium*) [37].

Es un estándar para el intercambio de información entre diferentes plataformas, ya que se puede usar en casi cualquier tecnología existente. Su habilidad para compartir información entre distintos sistemas es particularmente útil para establecer relaciones entre elementos de distintos dominios y de esta forma utilizarse para mapear el sistema.

Es un lenguaje basado en etiquetas, lo cual permite que el programador modifique las capacidades del lenguaje en función de sus necesidades. En la Figura 13 puede apreciarse un ejemplo de un catálogo de CDs donde se emplean etiquetas XML para describir los distintos atributos de un disco compacto.

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOGO>
   <CD>
      <TITULO>Empire Burlesque</TITULO>
      <ARTISTA>Bob Dylan</ARTISTA>
      <PAIS>USA</PAIS>
      <COMPANIA>Columbia</COMPANIA>
      <PRECIO>10.90</PRECIO>
      <ANIO>1985</ANIO>
   </CD>
   <CD>
      <TITULO>Raro</TITULO>
      <ARTISTA>El cuarteto de nos</ARTISTA>
      <PAIS>UY</PAIS>
      <COMPANIA>Bizarro - EMI</COMPANIA>
      <PRECIO>8.95</PRECIO>
      <ANIO>2006</ANIO>
   </CD>
</CATALOGO>
```

Figura 13 - Catálogo de CDs expresado en XML

Junto con XML se definen lenguajes que se utilizan para manipular y utilizar su estructura [38]. Se distinguen DTD y XSD [39] que permiten definir la semántica de un documento XML, así como los que derivan de XSL (e*Xtensible Stylesheet Language*) [40] [41]: XSLT [42], XPath [43] y XQuery [44] que permiten manipular y recorrer sus estructuras.

DTD (*Document Type Definition*) es un esquema definido originalmente dentro del estándar XML que permite definir la semántica y sintaxis de dichos documentos, mientras que XSD (*XML Schema Definition Language*) es otro esquema que tiene la particularidad de ser el primero definido por la W3C por fuera del estándar XML. Este último presenta varias ventajas entre las que se destacan utilizar la sintaxis de XML, permitir la definición de tipos de datos y ser extensible.

XSLT (*eXtensible Stylesheet Language Transformations*) se utiliza para transformar documentos XML, aunque no ha sido diseñado como una herramienta de transformación multi propósito, presenta las bases necesarias para lograrlo. XPath (*the XML Path language*) es utilizado para navegar a través de los documentos XML explotando la semántica de los mismos. XQuery (*XML Query language*) se utiliza para realizar consultas sobre la información que se almacena en los documentos XML.

Se han realizado numerosos trabajos útiles basados en XML por ser poderoso y estándar, de los cuales destaca la API de JAVA para el procesamiento de XML (JAXP por sus siglas en inglés) [45] que permite validar y recorrer documentos XML a través de varias interfaces.

```
{"CATALOGO":
   {"CD":[
      {"TITULO": "Empire Burlesque" ,
       "ARTISTA":"Bob Dylan",
       "PAIS":"USA",
       "COMPANIA":"Columbia",
       "PRECIO":"10.90",
       "ANIO":"1985"
      },
      {"TITULO":"Raro",
       "ARTISTA":"El cuarteto de nos",
       "PAIS":"UY",
       "COMPANIA":"Bizarro - EMI",
       "PRECIO":"8.95",
       "ANIO":"2006"
      }]
   }
}
```

Figura 14 - Catálogo de CDs expresado en JSON

Otro lenguaje flexible que permite relacionar elementos de distintos dominios es JSON (*JavaScript Object Notation*), el cual es idempotente a XML [46]. Generalmente se usan como alternativas aunque también pueden verse casos de aplicaciones que utilizan ambos intentando obtener el máximo beneficio de ellos. Propone un lenguaje que no utiliza etiquetas y es por definición extensible. Una de las principales ventajas que se le reconocen es que es más simple generar analizadores sintácticos para este lenguaje, aunque en la práctica XML es tan popular que hoy día pueden encontrarse varios procesadores nativos, por ejemplo en los navegadores. Al no tener etiquetas JSON sigue siendo una mejor alternativa cuando la cantidad de datos que se enviarán entre los

distintos programas o dispositivos es un factor crítico. Por otra parte una desventaja que presenta JSON es que en ambientes JavaScript se ejecuta utilizando la función `eval()`, práctica no recomendable desde el punto de vista de seguridad.

3.5.2 Lenguajes de definición de aplicaciones

Web Application Description Language (WADL) es una descripción de una aplicación web realizada en lenguaje XML. Se utiliza para modelar los recursos que proveen los servicios de una aplicación y sus relaciones. Está pensado para los servicios REST (*REpresentational State Transfer*) [47] y fue propuesto por Sun Microsystems.

Su principal objetivo es el de simplificar la reutilización de servicios web basados en la arquitectura HTTP de la web y es independiente de las plataformas y los lenguajes en que son desarrolladas las mismas.

Presenta la desventaja que pese a haber sido admitida por la W3C desde el 2009 [48] no está estandarizado y actualmente no es ampliamente aceptado.

Web Services Description Language (WSDL) es comúnmente usado para describir en detalle los servicios SOAP, puede verse como el equivalente WADL de REST. WSDL es muy flexible en cuanto a las opciones para establecer la conexión, sin embargo en su primera versión [49] no soportaba operaciones HTTP más allá de `GET` y `POST`. Por esto WSDL no podía usarse para describir servicios REST, ya que los mismos suelen usar otras operaciones de HTTP como ser `PUT` y `DELETE`.

A partir de la versión 2.0 [50] WSDL pasó a soportar todas las operaciones de HTTP y por esto hoy en día se considera un lenguaje aceptable para documentar servicios REST. Es una alternativa más flexible que WADL, aunque esta última es una versión más liviana y generalmente más fácil de comprender.

A continuación se presentan trabajos que plantean métodos para mapear entre lenguajes predefinidos.

3.5.3 Trabajos relacionados sobre mapeos

En un trabajo planteado por Yongzhen Ou [51], se estudia cómo mapear en ambas direcciones recursos entre un modelo UML y un Modelo Entidad Relación (MER) . Estos mapeos se basan en la sintaxis de los lenguajes particulares por lo que extenderlos se vuelve un problema cuya dificultad se incrementa en orden cuadrático con la cantidad de lenguajes a considerar. Si se emplea un lenguaje intermedio al que todos se traduzcan este incremento en la dificultad pasa a orden lineal, ya que el problema de traducir todos los lenguajes entre ellos cambia por traducirlos a uno en particular.

Otra alternativa consiste en elevar el nivel de abstracción de este problema generando un exhaustivo y riguroso esquema conceptual, que permite asignarle un significado a cada elemento de estos dominios con el fin de facilitar el intercambio de información entre los mismos. Con esto es posible resumir todos los elementos de los dominios a su significado y posteriormente aprovechando el esquema conceptual, mapear los que tienen el mismo significado entre los distintos dominios. En computación esto es llamado ontología [52].

Con estos trabajos se pueden asentar las bases para automatizar el mapeo de los recursos y evitar que sea definido de forma manual.

Seguidamente se detallarán los requerimientos del *framework* para alcanzar los objetivos.

[Página dejada en blanco intencionalmente.]

4 Análisis

En este capítulo se describe el proceso de definición y *enforcement* de la política de seguridad y los actores involucrados en él.

Se definen los tipos de reglas que serán considerados y las posibles incompatibilidades que existan entre los mismos.

Se continúa con el detalle de los requerimientos funcionales que debe cumplir DEPSA y los lenguajes para expresar políticas de seguridad y las funciones de mapeos entre el modelo de la aplicación y los dominios de las herramientas de *virtual patching*. Además se especificarán los requerimientos no funcionales de DEPSA y los casos de uso que se desarrollarán.

Se presentan a continuación los roles que tienen las distintas personas involucradas en el uso del *framework* para realizar el *enforcement* de una aplicación.

4.1 Actores

En la definición y *enforcement* de la política de seguridad hay diversos actores con distintas responsabilidades:

- **Especialista en seguridad**: responsable de la seguridad que se encarga de detectar vulnerabilidades de la aplicación y determinar las políticas de seguridad necesarias para mitigarlas.
- **Especialista en el modelo de la aplicación**: experto en el modelo de la aplicación y la forma en que se relaciona con los distintos dominios del sistema. Puede existir un especialista por cada dominio del sistema.
- **Usuario de DEPSA**: persona que ejecuta DEPSA para generar las reglas de seguridad a partir de las políticas de seguridad y mapeos definidos.
- **Responsable de la herramienta de *virtual patching***: encargado de configurar la herramienta de *virtual patching* para proteger el sistema.
- **Usuario de la aplicación**: persona que utiliza la aplicación sobre la que se desea aplicar la técnica de *virtual patching*.

Se procede a realizar una descripción del proceso a efectuar para aplicar el *enforcement* a una aplicación.

4.2 Proceso de definición y enforcement de la política de seguridad

Para realizar la definición y *enforcement* de la política de seguridad se sigue una secuencia de pasos en los que cada actor tiene una responsabilidad que cumplir. En la Figura 15 se observa un diagrama de flujo que expresa las tareas en el proceso y los actores involucrados.

El proceso comienza cuando el especialista en el modelo de la aplicación relaciona los elementos del modelo con los recursos en los dominios de las herramientas de *virtual patching* para los que se ejecutará el *Framework*.

Mientras se realizan los mapeos el especialista en el modelo de la aplicación junto con el especialista en seguridad puede ir definiendo las políticas de seguridad que se desea sean aplicadas en la aplicación objetivo.

Una vez definidos los mapeos y la política de seguridad a ser aplicada, el usuario de DEPSA inicia

el *framework*, para que este genere las configuraciones necesarias para las herramientas de *virtual patching* deseadas.

Finalmente el responsable de configurar la herramienta de *virtual patching* toma la configuración generada y asegura la aplicación.

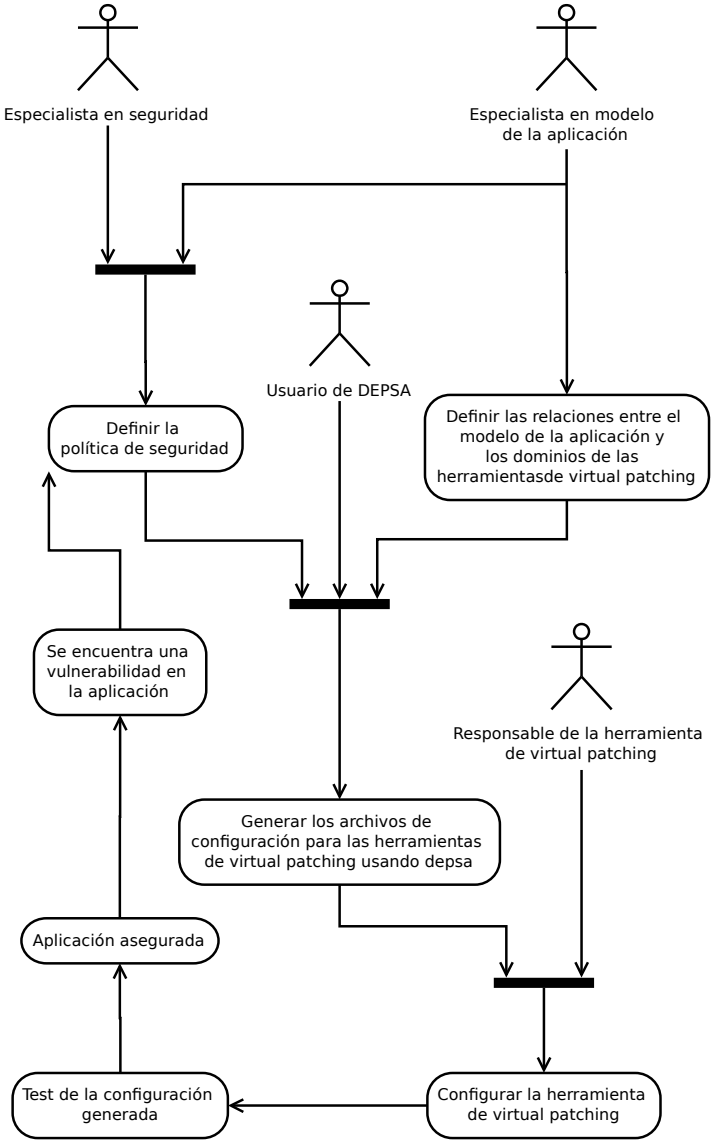


Figura 15 - Diagrama con el flujo de uso del framework

Cuando una nueva vulnerabilidad es detectada en la aplicación, se agregan los nuevos elementos o restricciones a los existentes en la política de seguridad previamente definida y luego se repite el proceso presentado. La tarea de definición de los mapeos no es necesaria en estos casos, ya que las relaciones entre los dominios no se ve afectada por la modificación de la política a aplicar. Sí sería necesario en caso de agregar una nueva herramienta de *virtual patching* que tuviera un dominio que no se encuentre definido o si se define una restricción en un elemento que no se encuentre mapeado.

4.3 Políticas de seguridad

Las políticas de seguridad son un conjunto de reglas que se aplican sobre los recursos de la

aplicación. Las reglas son las restricciones que deben cumplir los valores de los recursos o el acceso a ellos. Los recursos se definen como los elementos del modelo de la aplicación o el sistema. Por ejemplo aplicar una regla al sistema implica aplicar la restricción a todos los recursos contenidos en el modelo de la aplicación.

Las particularidades de los tipos de reglas se definen en la siguiente sección.

4.3.1 Tipos de reglas

Se definen a continuación los tipos de reglas que formarán las políticas de seguridad que serán consideradas para crear el lenguaje y para ser manipuladas por el *framework*. Como se aprecia en Figura 16, los tipos de reglas se dividien en tres grupos: P*ermission, Default* y *Specific*.

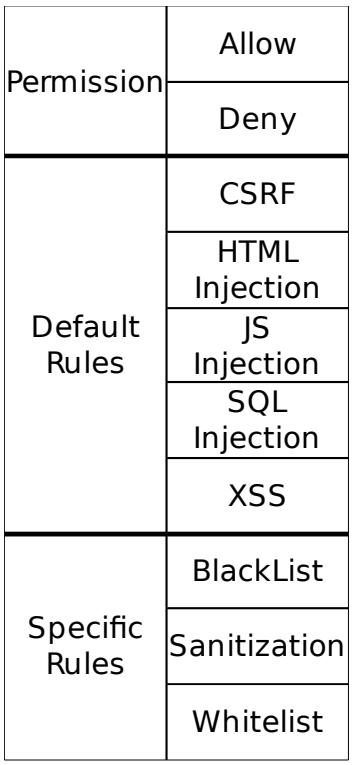


Figura 16 - Tipos de reglas

Las reglas de tipo *Permission*, se utilizan para representar el control de acceso sobre un recurso particular. Se subdividen en *Allow* y *Deny* e indican si un usuario, grupo de usuarios o una sesión particular pueden acceder en determinado momento a un recurso específico. Las reglas *Allow* representan que un usuario tiene permisos sobre cierto recurso mientras que las reglas *Deny* representan lo contrario.

Las de tipo *Default* se definen en función de vulnerabilidades genéricas de seguridad como las nombradas en el OWASP *Top Ten*. En esta primera versión se considerarán XSS, CSRF, *JavaScript Injection*, *SQL Injection* y *HTML Injection*.

Finalmente las reglas de tipo *Specific* son aquellas sobre las que el especialista en seguridad tiene mayor control, permiten especificar directamente que formatos son aceptados o rechazados por los recursos de la aplicación, así como definir comportamientos particulares ante determinadas entradas o acciones. Se clasifican como *Blacklist*, *Whitelist* y *Sanitization*. Las reglas de tipo *Blacklist* indican formatos que no deben ser admitidos por la herramienta de *virtual patching*, las *Whitelist* determinan formatos que sí deben ser admitidos, y las *Sanitization* determinan entradas que deben ser sanitizadas.

Al aplicar reglas a un recurso se pueden generar contradicciones, no solamente sobre las reglas definidas en el mismo recurso sino también con las reglas que se definieron para el sistema, lo que puede generar resultados no deseados. Por esto se presenta a continuación un estudio sobre el comportamiento de las reglas al interactuar entre ellas.

4.3.2 Incompatibilidades entre tipos de reglas

El lenguaje a utilizar para describir las políticas de seguridad debe permitir definir a cada recurso todos los tipos de reglas. A pesar que esto es sintácticamente correcto, la compatibilidad de las reglas entre sí se ve definida en función de su semántica. Por ejemplo es posible definir para LeagueManager una política donde se defina para el recurso nombre de Persona una regla de tipo *Whitelist* que acepte solo letras y espacios, y una regla de tipo *Blacklist* que no acepte espacios. En este caso la política es sintácticamente correcta pero la regla *Whitelist* permite nombres con espacio que luego la regla *Blacklist* impide. El *framework* debe advertir de esto al usuario para indicarle que puede obtener resultados distintos de los que espera.

Definir cuando dos reglas presentan problemas con precisión es una tarea muy compleja y no está planteada como uno de los objetivos del proyecto, por lo que se realiza una validación únicamente en función del tipo de regla. Entonces, se considera que dos reglas son incompatibles cuando presentan potenciales problemas que deben ser evaluados por otra herramienta o un usuario calificado, por lo que se genera una advertencia al momento de detectarlas.

La Figura 17 muestra un resumen de como se relacionan los tipos de reglas entre sí.

		Permission		Default Rules					Specific Rules		
		Allow	Deny	CSRF	HTML Injection	JS Injection	SQL Injection	XSS	BlackList	Sanitization	Whitelist
Permission	Allow	✔	⚠	✔	✔	✔	✔	✔	✔	✔	✔
	Deny	⚠	✔	✔	✔	✔	✔	✔	✔	✔	✔
Default Rules	CSRF	✔	✔	✔	✔	✔	✔	✔	⚠	⚠	⚠
	HTML Injection	✔	✔	✔	✔	✔	✔	✔	⚠	⚠	⚠
	JS Injection	✔	✔	✔	✔	✔	✔	✔	⚠	⚠	⚠
	SQL Injection	✔	✔	✔	✔	✔	✔	✔	⚠	⚠	⚠
	XSS	✔	✔	✔	✔	✔	✔	✔	⚠	⚠	⚠
Specific Rules	BlackList	✔	✔	⚠	⚠	⚠	⚠	⚠	✔	⚠	⚠
	Sanitization	✔	✔	⚠	⚠	⚠	⚠	⚠	⚠	✔	⚠
	Whitelist	✔	✔	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠

✔ compatibles ⚠ incompatibles

Figura 17 - Relacionamiento de los tipos de reglas

Las reglas de tipo *Permission* se definen en función de quien realiza determinada acción, más que en las particularidades del pedido en sí. Se definen entonces sobre una relación acceso-recurso, por lo tanto no tienen problemas de compatibilidad sobre políticas que se definen exclusivamente sobre los recursos. En consecuencia las reglas *permission* son compatibles con los demás tipos de reglas. Resta definir la compatibilidad de estas reglas entre sí. Las reglas *Allow* representan que un usuario tiene permisos sobre cierto recurso, las reglas *Deny* representan lo contrario. Es posible asignar varios permisos del mismo tipo a un recurso, ya sea bloqueando el acceso al mismo o autorizándolo, sin embargo es necesario prestar atención ante casos de recursos que tengan asignados permisos de tipo *allow* y *deny* a la vez, por esto las reglas *Allow* y *Deny* se determinan incompatibles entre sí.

Para las reglas de tipo *Default* se espera que sean incluidas por las herramientas a utilizar para aplicar la técnica de *virtual patching*. Se espera de las mismas que dicha herramienta genere estas reglas de modo que sean compatibles o controle en caso que no lo sean. Por esto se considera que las reglas de esta categoría son compatibles entre sí. En caso de no ser así para alguna herramienta se deberá evaluar por separado para dicho caso particular.

Las reglas de tipo *Specific* permite al especialista en seguridad controlar las reglas a un nivel que es posible afectar el funcionamiento de las reglas de tipo *Default*, por lo tanto las mismas se declaran incompatibles. Restan evaluar las reglas que se encuentran dentro de esta categoría, es decir

Blacklist, *Whitelist* y *Sanitization*. Deben ser evaluadas con precaución, por lo que se definen como incompatibles entre ellas. Es posible bloquear distintos tipos de entradas para un recurso dado, por lo que las reglas de tipo *Blacklist* son compatibles con ellas mismas, al igual que las reglas S*anitization*. Las reglas de tipo *Whitelist* implican que se acepten determinadas entradas, y potencialmente que cualquier otra sea rechazada, por lo cual en caso de tener varias deben evaluarse con cuidado, definiendo así las mismas como incompatibles.

Se describen a continuación los requerimientos detectados que debe cumplir el *framework*.

4.4 Especificación de requerimientos

En esta sección se detallan los requerimientos generales que deben ser cumplidos para alcanzar los objetivos del proyecto.

Basado en los objetivos surge el requisito de definir una arquitectura para DEPSA que permita intercambiar los distintos módulos sin necesidad de rehacer todo el sistema. Esta arquitectura deberá además contemplar la posibilidad de agregar nuevos módulos para la generación de la configuración necesaria para nuevas herramientas de *virtual patching* capaces de efectuar el *enforcement* de la política de seguridad definida.

Para generar independencia entre la definición de las políticas de seguridad y la herramienta de *virtual patching* es necesario definir dos lenguajes. Uno de los lenguajes debe permitir definir las políticas de seguridad sobre el modelo de la aplicación, y el otro lenguaje definir la relación entre los elementos del modelo y los recursos de los dominios de las distintas herramientas de *virtual patching*.

4.4.1 Requerimientos funcionales

Para la especificación de los requerimientos funcionales se realiza una división en tres subsecciones: los requerimientos de DEPSA, los del lenguaje para expresar las políticas de seguridad denominado SPLang y los del lenguaje para expresar los mapeos denominado MapLang.

4.4.1.1 Requerimientos de DEPSA

A partir de la definición de una politica de seguridad expresada en SPLang, y de la definición del mapeo expresado en MapLang, DEPSA deberá procesarlos y generar la configuración necesaria para que ModSecurity, una herramienta de *virtual patching*, realice el *enforcement* de la política definida.

Uno de los problemas de DEPSA es la resolución de conflictos entre las restricciones definidas en la política de seguridad que aplican a un mismo atributo o que fueron definidas para el sistema. En estos casos se deberá considerar que las reglas definidas sobre un atributo específico prevalecen sobre las reglas definidas sobre el sistema.

Se deberán proveer funcionalidades que permitan validar la sintaxis de los archivos de SPLang y MapLang para simplificar la creación de estos archivos.

Se resumen entonces los requerimientos necesarios para DEPSA en:

- Procesar un archivo conteniendo una política de seguridad expresada en SPLang.
- Procesar un archivo conteniendo un mapeo entre dominios expresado utilizando MapLang.
- Generar un archivo de configuración para la herramienta de *virtual patching* ModSecurity.
- Chequear la sintaxis de un archivo expresando una política de seguridad utilizando SPLang.

- Chequear la sintaxis de un archivo expresando mapeos entre dominios utilizando MapLang.
- Implementar una política de seguridad que permita validar el trabajo realizado.

4.4.1.2 Requerimientos de SPLang

SPLang será el lenguaje mnemotécnico utilizado para expresar la política de seguridad en el modelo de la aplicación. Este lenguaje deberá permitir expresar el control de acceso y las restricciones que deben cumplir los valores de los atributos en las entidades, así como identificar las entidades del modelo de la aplicación para evitar colisiones. Además de expresar restricciones a nivel de las entidades, se podrán definir restricciones a nivel del sistema, lo que se interpretará de que la restricción se aplica en todos los atributos.

Con el fin de simplificar la tarea de definición de reglas para los recursos, el lenguaje debe proveer una forma de definir conjuntos de reglas, que luego podrán ser aplicadas a diversos recursos que cumplan con las mismas restricciones.

Como el conjunto de las vulnerabilidades existentes no es estático, el lenguaje debe permitir la extensión para incluir las nuevas vulnerabilidades que sean descubiertas.

El lenguaje establecerá un mecanismo que permita definir cual es la regla que se aplica antes al atributo en caso de existir inconsistencias entre ellas.

Se resumen los requerimientos que deberá cumplir SPLang:

- Definir un lenguaje mnemotécnico que permita expresar políticas de seguridad sobre un modelo.
- Identificar las entidades en el modelo de la aplicación de manera de evitar colisiones entre atributos de igual nombre.
- Expresar restricciones que deben cumplir los valores de los atributos de las entidades para proteger la aplicación de las vulnerabilidades de seguridad conocidas. (SQLInjection, XSS, RBAC, etc).
- Expresar restricciones que permitan realizar validación de entrada para un atributo en particular dado que son útiles para evitar falsos positivos. (*Whitelist, Blacklist*)
- Expresar restricciones a nivel del sistema que apliquen a todos los recursos.
- Definir conjuntos de restricciones para luego ser aplicados a varios recursos.
- Permitir la extensión con nuevos tipos de reglas.
- Definir un método que sirva para determinar qué restricción se debe aplicar antes.

4.4.1.3 Requerimientos de MapLang

MapLang será el lenguaje utilizado para realizar el mapeo entre el modelo de la aplicación y los distintos dominios donde aplican las restricciones las herramientas de *virtual patching*, esto implica que los elementos del modelo de la aplicación son mapeados con múltiples elementos de los dominios de las herramientas, creando una relación 1 a N.

Se resumen los requerimientos que deberá cumplir MapLang:

- Definir un lenguaje mnemotécnico que permita relacionar los elementos del modelo de la aplicación con los recursos del dominio de una o varias herramientas de *virtual patching*.

- Identificar las entidades en el modelo de la aplicación de manera de evitar colisiones entre atributos de igual nombre.
- Soportar relaciones con cardinalidad 1 a N, donde todos los elementos de los dominios deben ser mapeados con un único elemento del modelo de la aplicación.

4.4.2 Requerimientos no funcionales

El sistema debe ser extensible permitiendo la inclusión de nuevas reglas de seguridad o la modificación de las existentes. Además debe permitir la sustitución de los lenguajes de entrada (SPLang y MapLang) y la posibilidad de agregar nuevas herramientas de *virtual patching*, generando el menor impacto posible.

Se detallan a continuación los casos de uso implementados.

4.5 Casos de uso

En la Figura 18 se pueden apreciar los casos de uso relevados para DEPSA.

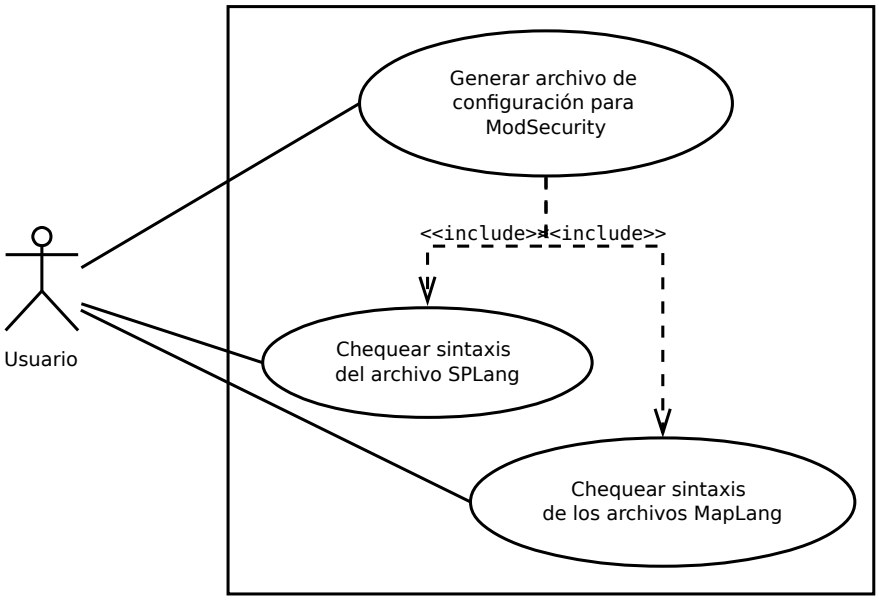


Figura 18 - Diagrama de casos de uso

A continuación se realiza una descripción detallada de los mismos.

4.5.1 Generación del archivo de configuración de ModSecurity

Sinopsis	Genera un archivo de configuración para ModSecurity a partir de un archivo con la política de seguridad escrito en SPLang y un archivo que contenga los mapeos entre el modelo de la aplicación y la interfaz web escrito en MapLang.
Actores	Usuario de DEPSA
Pre-Condiciones	1. Archivo con la política de seguridad expresado en lenguaje SPLang. 2. Archivo con el mapeo entre los elementos del modelo de la aplicación y los correspondientes elementos visibles en ejecución expresado en MapLang.

Post-Condiciones	1. Genera un archivo de configuración para ModSecurity con el fin de realizar el enforcement de la política de seguridad definida en la ruta especificada.	
Flujo Principal		
Paso	*Actor*	*Descripción*
1	Usuario	Especifica las rutas a los archivos que contienen la política de seguridad en lenguaje SPLang y el mapeo entre los dominios en lenguaje MapLang. Indica que desea generar los archivos de configuración para una lista de herramientas disponibles para realizar el enforcement de la política de seguridad definida.
2	Sistema	Incluye el caso de uso [Chequear sintaxis de los archivos de SPLang y MapLang] en el paso 2 del flujo principal.
3	Sistema	Valida que no existen incompatibilidades en las restricciones a aplicar en cada atributo y entre los atributos y las restricciones a aplicar a todo el sistema.
4	Sistema	No existen incompatibilidades.
5	Sistema	Genera el archivo con la política de seguridad definida en la ruta especificada.
6		Fin del CU.
Flujo alternativo 4A - Existen incompatibilidades: dos restricciones con igual prioridad para un mismo atributo o ModSecurity no puede resolverlas.		
Paso	*Actor*	*Descripción*
1	Sistema	Muestra al usuario un mensaje de error indicando cuáles reglas no son compatibles.
2		Fin del CU.
Flujo alternativo 4B - Existen incompatibilidades: dos restricciones que pueden generar un resultado inesperado pero que no puede ser detectado por su definición.		
Paso	*Actor*	*Descripción*
1	Sistema	Muestra al usuario un mensaje de advertencia indicando cuáles reglas no son compatibles y pueden generar un resultado no esperado.
2		Fin del CU.

4.5.2 Chequear sintaxis de los archivos de SPLang

Sinopsis	Se realiza un chequeo de la sintaxis del archivo de políticas de seguridad escrito en SPLang.
Actores	Usuario de DEPSA
Pre-Condiciones	1. Archivo con la política de seguridad expresado en lenguaje SPLang.

Post-Condiciones	---	
Flujo Principal		
Paso	*Actor*	*Descripción*
1	Usuario	Especifica la ruta al archivo que contiene la política de seguridad en lenguaje SPLang. Indica que desea revisar la sintaxis.
2	Sistema	El sistema recorre el archivo con las políticas de seguridad analizando si la sintaxis del archivo es correcta.
3	Sistema	La sintaxis del archivo con las políticas de seguridad era correcta.
4		Fin de caso de uso.
Flujo alternativo 3A - La sintaxis del archivo con las políticas de seguridad no es correcta.		
Paso	*Actor*	*Descripción*
1	Sistema	Muestra al usuario un mensaje de error en el que informa el lugar en donde se encuentra el error de sintaxis.
2		Fin de caso de uso.

4.5.3 Chequear sintaxis de los archivos de MapLang

Este caso de uso es análogo al de 4.5.2 Chequear sintaxis de los archivos de SPLang.

[Página dejada en blanco intencionalmente.]

5 Diseño de la solución

Se presenta en este capítulo el diseño del *framework*, comenzando por la arquitectura del sistema, la descripción de los módulos que la componen, sus responsabilidades y las interfaces definidas para permitir la extensibilidad.

Las decisiones de diseño describen las soluciones a los problemas encontrados para cumplir con requerimientos especificados en la etapa de análisis. Se destacan las consideraciones para que los módulos sean intercambiables, se permitan múltiples generadores para las diferentes herramientas de *virtual patching*, la posibilidad de incorporar nuevos tipos de reglas, la detección de posibles incompatibilidades y particularidades del generador de ModSecurity.

Finalmente se presenta el lenguaje definido para representar políticas de seguridad sobre el modelo de la aplicación y el lenguaje que permite relacionar los elementos que se encuentran en el modelo de la aplicación con los elementos que se encuentran en los dominios utilizados por las diferentes herramientas de *virtual patching*.

5.1 Diseño de la arquitectura

Para alcanzar los objetivos, manteniendo la independencia de cada funcionalidad se descompone la solución en 6 módulos: `SecurityPolicies`, `Mappings`, `PreGenerator`, `Generator`, `ModSecurityGenerator`, `DEPSA`. En la Figura 19 se aprecia el diagrama con la arquitectura definida.

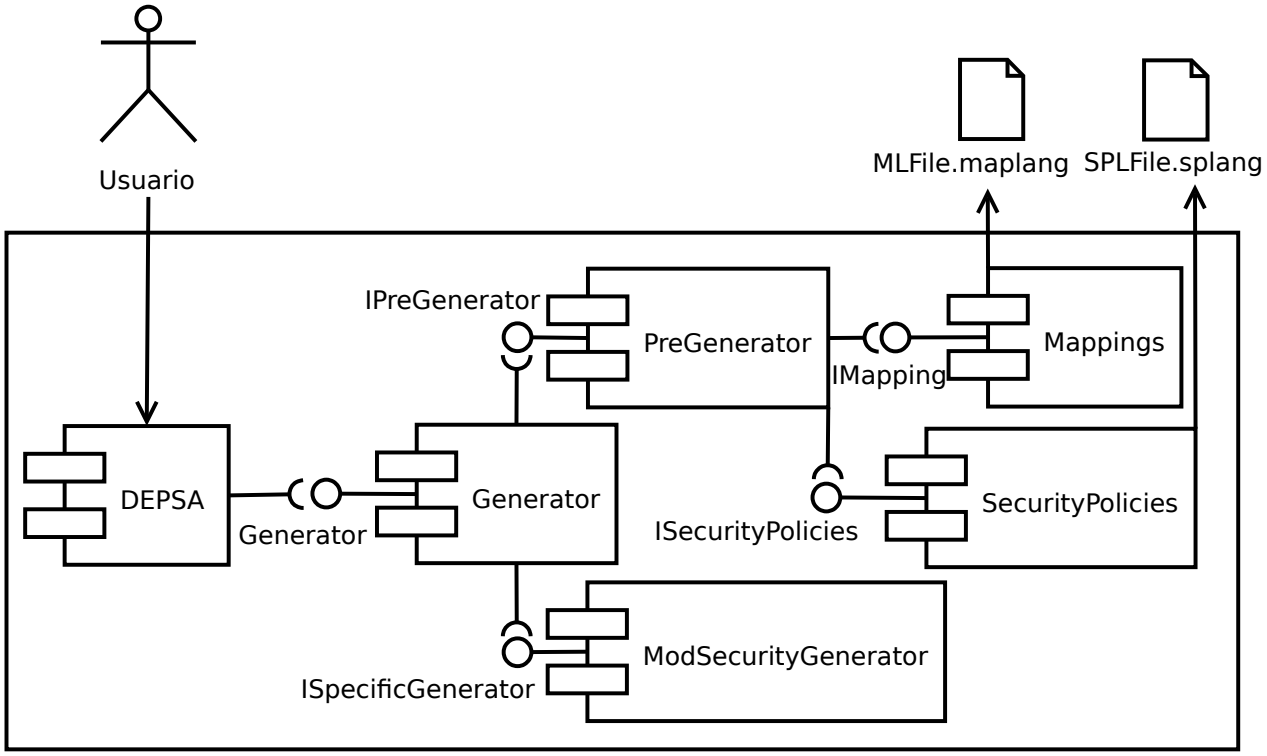


Figura 19 - Diagrama de arquitectura

Se describen a continuación las responsabilidades que se le asigna a cada módulo que colaboran a alcanzar los objetivos de DEPSA.

5.1.1 Módulo SecurityPolicies

Responsable de procesar las políticas de seguridad escritas en lenguaje SPLang y generar una estructura de clases adecuada para ser procesada por el módulo `PreGenerator`.

El procesamiento incluye verificar que la sintaxis en la que se encuentra escrita la política de seguridad sea correcta, en caso de detectar errores se generarán mensajes dando información sobre su ubicación.

En caso que se desee utilizar otro lenguaje para definir políticas de seguridad, se deberá definir un nuevo módulo equivalente a `SecurityPolicies`, que sea capaz procesar el nuevo lenguaje y que respete la interfaz de datos del módulo `PreGenerator`, para de esta forma encapsular el lenguaje dentro del módulo.

5.1.2 Módulo Mappings

Encargado de procesar el archivo que mantiene las relaciones entre el modelo de la aplicación y los dominios utilizados por las herramientas de *virtual patching*, escrito en lenguaje MapLang.

El procesamiento incluye verificar que la sintaxis en la que se encuentra escrito el mapeo sea correcto, en caso de detectar errores se generarán mensajes dando información sobre su ubicación.

Sustituyendo el módulo `Mappings` por otro que cumpla con la interfaz `IMapping` es posible utilizar otros lenguajes distintos a MapLang para realizar los mapeos entre dominios.

5.1.3 Módulo PreGenerator

Realiza tareas que se desprenden de la vinculación entre los módulos de `SecurityPolicies` y `Mappings` que no pueden ser realizadas de forma independiente y que son comunes a todos los módulos de los generadores específicos.

Se genera una estructura que vincule los recursos de cada dominio mapeado con las reglas que se le deben aplicar expresadas en la política de seguridad.

Además de la tarea de vincular los recursos, verifica la compatibilidad entre los tipos de reglas que se aplican a un recurso, como se detalla en la sección 4.3.2 Incompatibilidades entre tipos de reglas.

5.1.4 Módulo Generator

Su función es invocar las tareas que debe realizar `PreGenerator` y los módulos generadores específicos, por ejemplo `ModSecurityGenerator`. Además se hará cargo de la escritura a disco de los archivos de configuración generados.

5.1.5 Módulo ModSecurityGenerator

Implementación de un generador específico. Recibirá como entrada los datos generados por el módulo `PreGenerator` y a partir de ellos realizará una representación de un archivo de configuración para ModSecurity que expresa la política de seguridad definida en SPLang.

En caso de que se desee generar archivos de configuración para otras herramientas de *virtual patching*, es necesario agregar nuevos módulos que cumplan con la interfaz `ISpecificGenerator` y notificar al generador para que los agregue a su orden de ejecución, incluso es posible generar archivos de configuración para múltiples *firewalls* en una sola ejecución.

5.1.6 Módulo DEPSA

Brinda la interfaz de usuario a ser utilizada para realizar las invocaciones al módulo `Generator`.

5.2 Interfaces definidas

Se describen a continuación las interfaces definidas para la comunicación entre los módulos. Para extender las funcionalidades de DEPSA solo es necesario agregar una nueva implementación de las interfaces.

IMpapping

Define la comunicación con el módulo que procesa los mapeos entre los dominios.

Define la operación *`map`*, la firma de esta puede verse en el Código 2. La operación recibe la ruta del archivo donde se encuentran definidos los mapeos y retorna una estructura de tipo `Mapping` que puede apreciarse en la Figura 20.

```
public Mapping map(String path)
```

Código 2 - Función definida en la interfaz de Mappings

La estructura está compuesta por una lista que representa todos los dominios mapeados del sistema, donde cada uno cuenta con un identificador, la representación del sistema dentro de dicho dominio y un conjunto de relaciones. Las relaciones indican la correspondencia entre cada recurso del modelo de la aplicación (`Resource`) con los recursos dentro de ese dominio (`RelatedResources`).

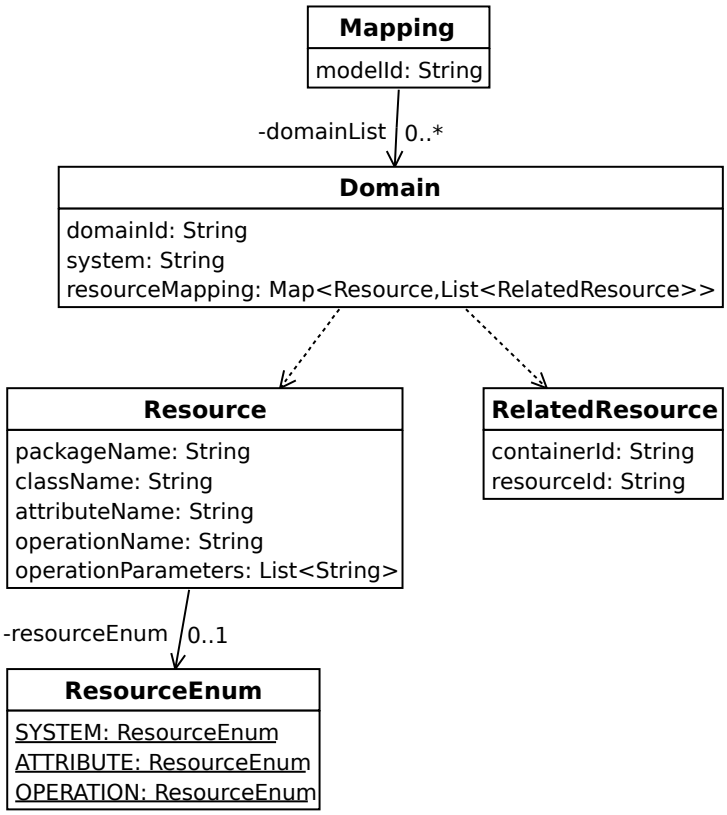


Figura 20 - Estructura de datos Mapping

ISecurityPolicies

Define la interacción del módulo encargado de leer la política de seguridad de un archivo.

```
public SecurityPolicies securityPolicies(String path)
```

Código 3 - Función definida en la interfaz de SecurityPolicies

Cuenta con una única operación, cuya firma se puede ver en el Código 3, la cual recibe la ruta del archivo donde se encuentra la política de seguridad y la retorna procesada en una estructura de tipo `SecurityPolicies` que se aprecia en la Figura 21.

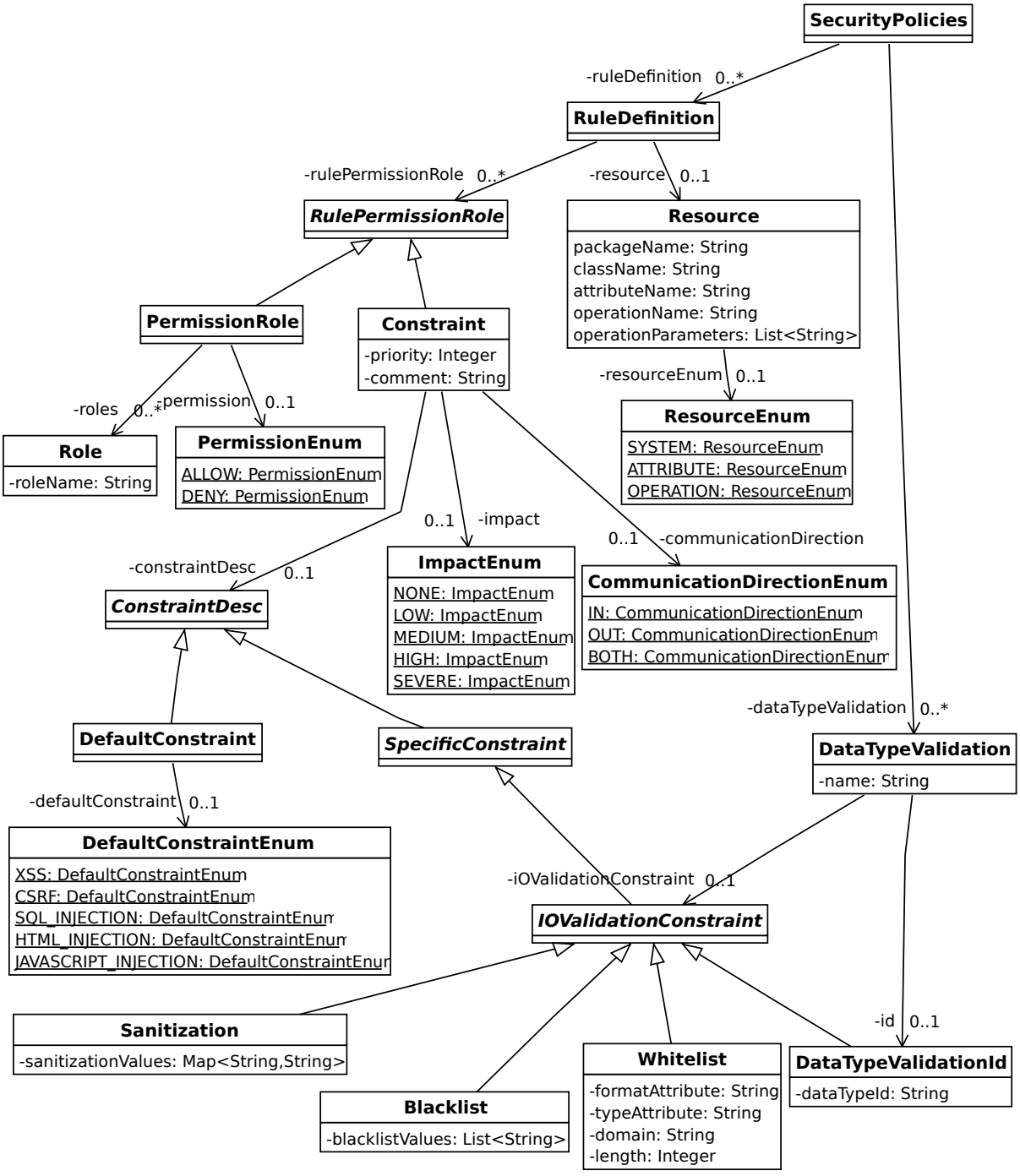


Figura 21 - Estructura de datos SecurityPolicies

`SecurityPolicies` se compone de dos colecciones. Una colección de `DataTypeValidation` que representa el conjunto de reglas para ser referenciado posteriormente en los distintos recursos a partir de su identificador único. La otra colección hace referencia a la definición de las reglas que aplican a los recursos. Los tipos de reglas se agrupan en `RulePermissionRole`.

Como el lenguaje debe ser extensible se emplea el mecanismo de herencia de clases y enumerados. Para las reglas de tipo `PermissionRole` y `DefaultConstraint` se define un enumerado porque con el valor del mismo es suficiente para determinar el significado de la regla, y si se desea agregar una nueva, alcanza con agregar un valor al enumerado. En los restantes puntos de extensión se emplea la herencia de clases debido a que puede ser necesario contar con más información, como es el caso de `IOValidationConstraint` y las clases que la extienden.

IPreGenerator

El módulo responsable de procesar el mapeo y las políticas de seguridad debe implementar esta interfaz. Se observa en el Código 4 las tres operaciones con las que cuenta.

```
public PreGeneratedPolicies execute (String policiesPath, String mappingsPath)
boolean checkPolicies(String policiesPath)
boolean checkMappings(String mappingsPath)
```

Código 4 - Funciones definidas en la interfaz de PreGenerator

La operación `execute` retorna una estructura de tipo `PreGeneratedPolicies` la cual vincula las reglas de cada recurso con los dominios en los que aparece. Las operaciones `checkPolicies` y `checkMappings` retornan si la sintaxis de los archivos es correcta. En la Figura 22 y la Figura 23 se muestra la estructura correspondiente a `PreGeneratedPolicies`.

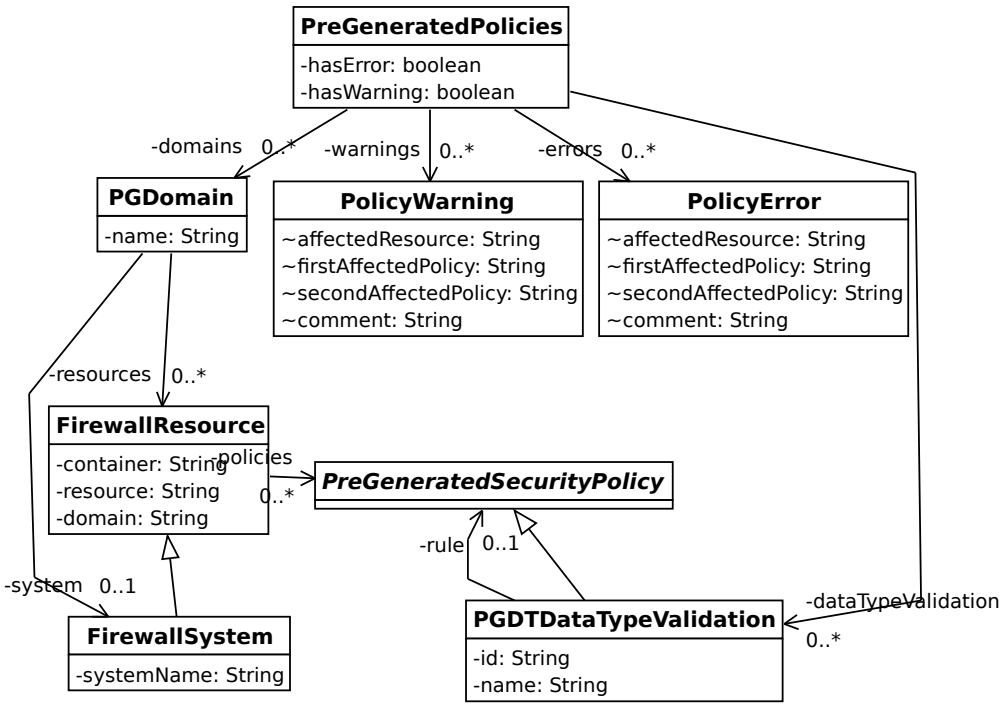


Figura 22 - Estructura de datos PreGeneratedPolicies

`PreGeneratedPolicies` cuenta con una colección de `PolicyWarning` y `PolicyError` que se utilizan para notificar a quien invoque al módulo de los errores o advertencias que se han encontrado durante la ejecución. Además tiene una colección de `PGDTDataTypeValidation` que son todos los `DataTypeValidation` definidos en `SecurityPolicies`, y una colección de `PGDomain` que son los dominios con algún recurso (`FirewallResource`) que posee reglas asignadas. A su vez cada uno de los recursos contiene una lista de `PreGeneratedSecurityPolicy` que representa todas las reglas que se le aplican.

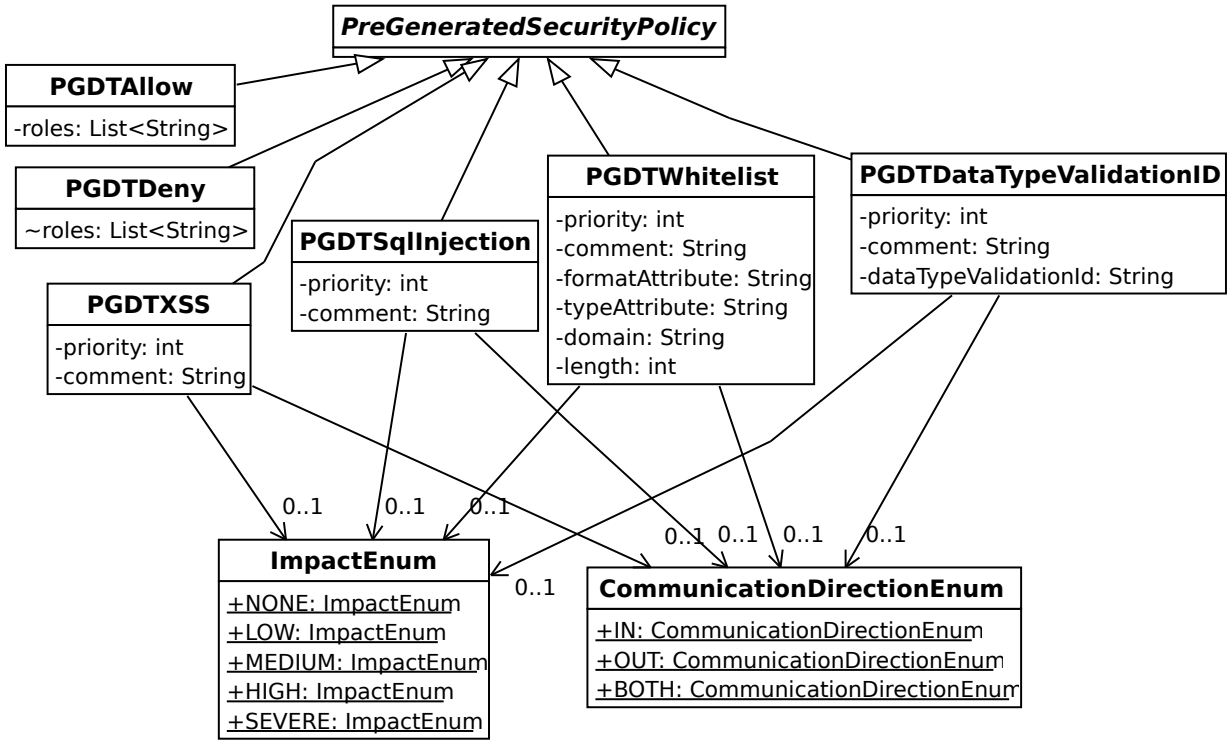


Figura 23 - Estructura de datos PreGeneratedSecurityPolicy

Por extensibildad de los tipos de reglas todas deben extender `PreGeneratedSecurityPolicy` como se puede ver en la Figura 23, la estructura de las reglas que son soportadas por el *framework* en esta versión.

ISpecificGenerator

Define la operación que deben implementar los generadores de archivos de configuración, cuya firma se ve en el Código 5.

```
GeneratedPolicies generate(PreGeneratedPolicies preGeneratedPolicies,
                                        HashMap<String, Object> parameters)
```

Código 5 - Función definida en la interfaz de SpecificGenerator

La operación recibe una estructura de `PreGeneratedPolicies`, y un hash con parámetros de configuración que se utilizan en caso que un generador específico requiera configuración particular para operar y su retorno es de tipo `GeneratedPolicies`.

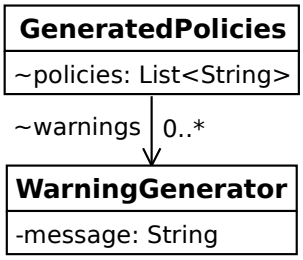


Figura 24 - Estructura de datos GeneratedPolicies

En la Figura 24 se observa la estructura de `GeneratedPolicies`, la cual es una lista de *strings*, donde cada *string* representa una línea completa dentro del archivo con la configuración de la política de seguridad para esa herramienta.

Generator

Define la interfaz con todas las operaciones que se pueden invocar sobre el *framework*. Siendo estás tres operaciones las que pueden observarse en el Código 6.

```
public List<SpecificGeneratorWarning> generate(String policiesPath, String
                            mappingsPath, List<SpecificGeneratorDT> generators)

boolean checkPolicies(String policiesPath)

boolean checkMappings(String mappingsPath)
```

Código 6 - Funciones definidas en la interfaz de Generator

La operación `generate` realiza la generación completa de las reglas para los generadores que se le indique en el parámetro `generators`. Además se le debe pasar por parámetro la ruta donde se encuentra el archivo de la política de seguridad y la ruta donde se encuentra el archivo que contiene los mapeos del sistema.

`CheckPolicies` y `checkWarning` son dos operaciones que permiten validar la correcta sintaxis de los lenguajes definidos en dichos archivos de forma individual.

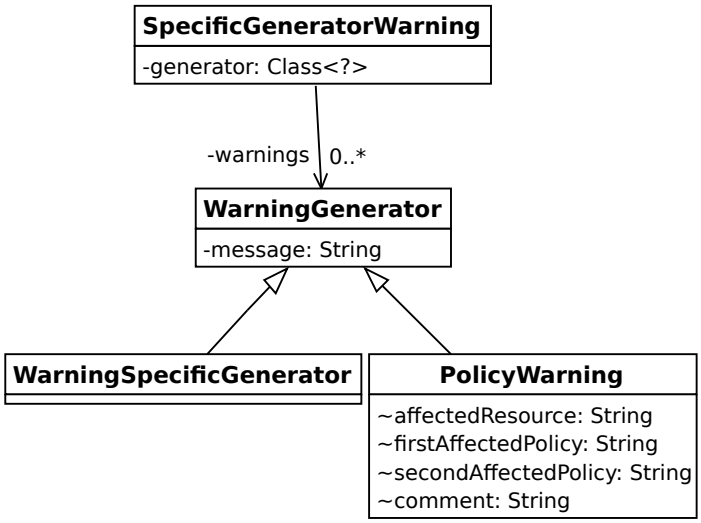


Figura 25 - Estructura de datos Generator

La estructura de `SpecificGeneratorWarning` se puede ver en la Figura 25. Contiene la clase del generador para identificarlo y una lista de `WarningGenerator` que son las advertencias que se generaron tanto en los generadores específicos (`WarningSpecificGenerator`) como en `PreGenerator` (`PolicyWarning`).

5.3 Decisiones de diseño

Para cumplir con los requisitos definidos en la etapa de análisis es necesario tomar decisiones sobre como resolver los problemas que surgen. En otros casos los problemas surgen por las tecnologías que se utilizan. Se plantea en esta sección las decisiones más importantes que moldearon tanto la arquitectura como el diseño final de DEPSA.

5.3.1 Extensibilidad de módulos

Durante el diseño del f*ramework* se hizo un fuerte hincapié en la extensibilidad del mismo, ya que es un requisito no funcional que el sistema permita la inclusión de nuevos lenguajes y tipos de reglas. Fueron pensados todos los módulos de manera independiente con responsabilidades fijas y con interfaces bien definidas. Esto hace que sea posible intercambiar los módulos sin afectar la funcionalidad del sistema, para esto cada módulo debe cumplir con las responsabilidades correspondientes a determinadas etapas de los compiladores modernos como puede verse en la Figura 26.

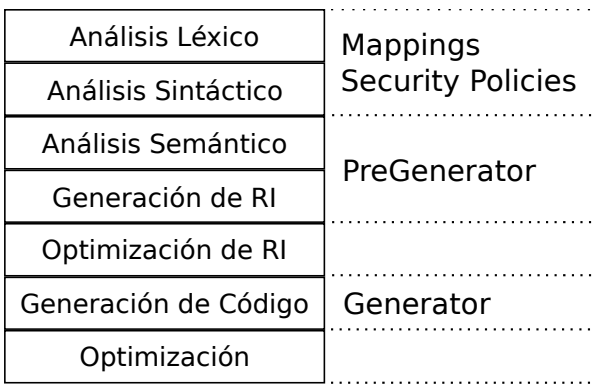


Figura 26 - Responsabilidades de los módulos

Cada uno de estos módulos debe implementar una de las interfaces definidas del *framework* para poder ser incluidos en el mismo, de este modo los nuevos módulos conocen el formato de las salidas que deben generar o las entradas que deben recibir. Por este medio es posible cambiar los lenguajes, generando nuevos módulos equivalentes a `Mappings` o `SecurityPolicies` según corresponda, que sepan cómo realizar el análisis léxico y sintáctico de los nuevos lenguajes. Por otro lado esto permite crear nuevos módulos y agregarlos al *framework* con impacto mínimo en la arquitectura. Por ejemplo en el caso que se desee agregar en un futuro un módulo que se responsabilice de manejar roles y usuarios, el mismo puede agregarse creando una única interfaz para él e implementando la misma al tiempo que se debe modificar el módulo `PreGenerator` para que consuma los datos del mismo sin afectar el resto del sistema. Este cambio puede apreciarse en la Figura 27.

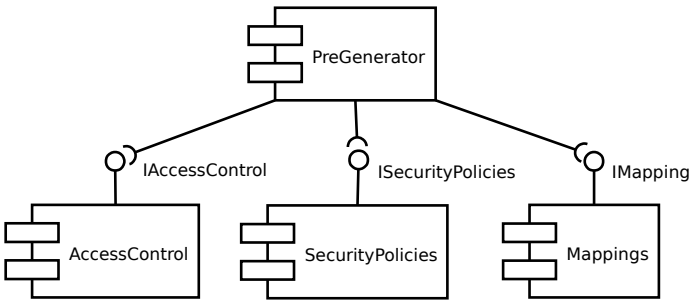


Figura 27 - Extensión del framework

Para que no exista dependencia entre los módulos se utiliza el patrón *Template Factory* para invocar las clases que implementan las interfaces `IPreGenerator`, `ISecurityPolicies` e `IMappings`.

5.3.2 Múltiples generadores

DEPSA debe soportar la generación de reglas para múltiples herramientas de *virtual patching*, por lo que se utiliza un diseño que permita que mediante una única interfaz sea posible invocar los distintos generadores. Se diseña `Generator` en base al patrón *Strategy*, el cual permite determinar qué generador específico (estrategia) utilizar en tiempo de ejecución. La forma en que se instancia este patrón se indica en la Figura 28.

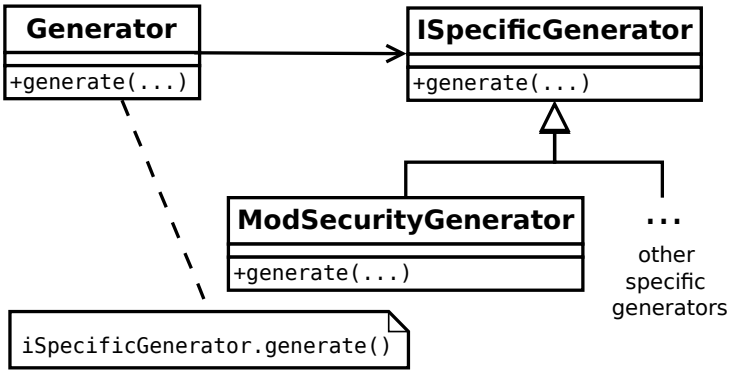


Figura 28 - Patrón Strategy aplicado en DEPSA

Se utiliza como identificador del generador específico el nombre de la clase que implementa la interfaz `ISpecificGenerator` que puede no estar disponible en tiempo de desarrollo.

5.3.3 Extensibilidad de reglas

La arquitectura de la solución debe permitir que se agreguen nuevos tipos de reglas o modificar el comportamiento de las ya existentes. Por esto es necesario que la representación de los tipos de reglas sean autocontenidos en el módulo `PreGenerator`.

El módulo PreGenerator interactuará con los tipos de reglas por medio de una clase abstracta que se puede ver en la Figura 29. Los tipos de reglas, que extienden `PreGeneratedSecurityPolicy`, sabrán como crearse mediante la función `makeMe` a partir de las clases de `RulePermissionRole` contenidas en la estructura `SecurityPolicies`. `PreGenerator` recorre todos los tipos de reglas existentes buscando aquel que pueda instanciarse correctamente a partir de la instancia de `RulePermissionRole`.

Figura 29 - Clase abstracta que define las reglas de PreGenerator

Los generadores específicos que deseen mantener la propiedad de extensibilidad para sus tipos de reglas deberán implementar un mecanismo que se lo permita. En el caso de `ModSecurityGenerator` se utiliza un mecanismo similar. La Figura 30 muestra la clase abstracta para representar los tipos de reglas.

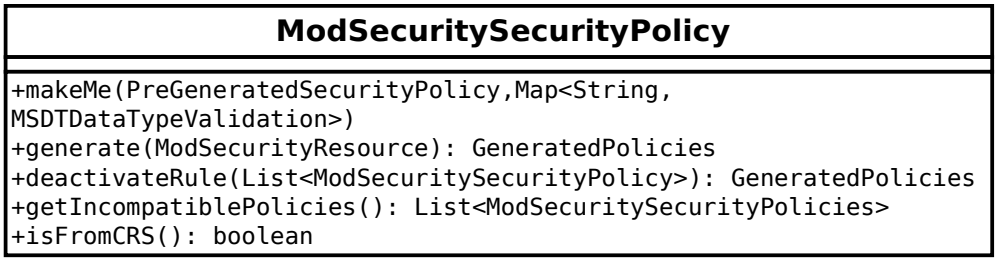


Figura 30 - Clase abstracta que define las reglas de ModSecurityGenerator

`ModSecuritySecurtiyPolicy` además de `makeMe` agrega métodos para generar la configuración de ModSecurity y conoce con que otras reglas no es compatible según las características propias de la herramienta.

5.3.4 Detección de reglas incompatibles

En el módulo `PreGenerator` es necesario que las reglas conozcan con que otras son compatibles, pero esta información no debe ser incluida en la estructura `PreGeneratedPolicies`, para que los generadores puedan hacer sus propias validaciones. Por esto `PreGenerator` cuenta con una estructura interna que se dedica exclusivamente a almacenar la relación entre los tipos de las reglas aplicadas a los recursos y el sistema. Dicha estructura se puede ver en la Figura 31 y es la que se utiliza para calcular la compatibilidad entre los tipos. La misma almacena para cada recurso (`ResourceRulesResume`), incluyendo el sistema, un representante (`RuleResume`) de los tipos de reglas que se le aplican que conoce la compatibilidad entre ellas.

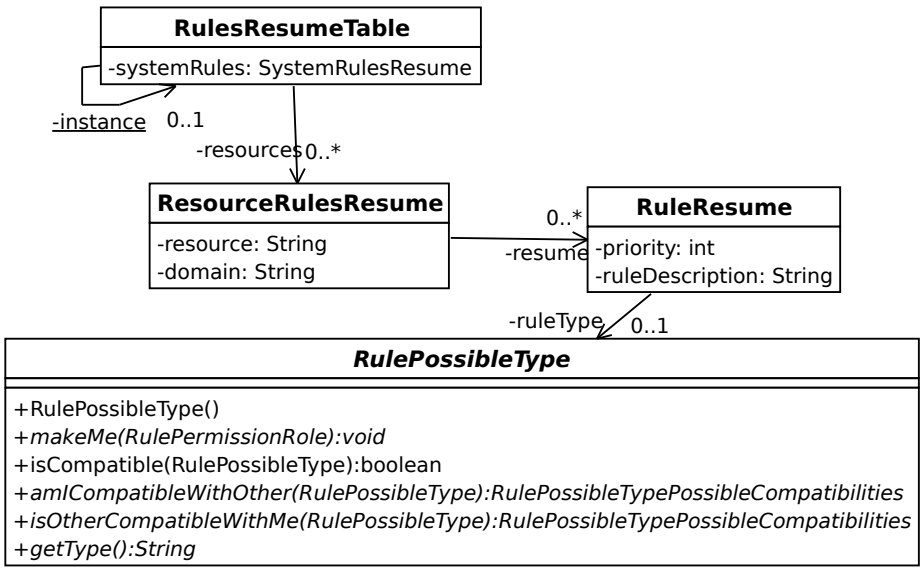


Figura 31 - Estructura para calcular compatibilidades

Para calcular la compatibilidad entre los tipos de reglas, los representantes utilizan *Template Method*, en la Figura 32 se ve su aplicación. Cada tipo conoce con que tipo de regla de los existentes al momento de su creación es compatible y con cuales no. En otro caso no conoce su compatibilidad retornando que no sabe resolver esa compatibilidad. `RulePossibleType`

responderá sobre la compatibilidad de las mismas por medio de la función `isCompatible`, la cual retorna un booleano que indica si son compatibles. Para resolver esto, se consulta a la primer clase sobre su compatibilidad por medio de la función `amICompatibleWithOther` que retorna un enumerado con los valores (`TRUE/FALSE/DONTKNOW`). Si esta conoce la respuesta retorna `TRUE/FALSE`, y en caso que retorne `DONTKNOW` se consulta a la segunda invocando la función `isOtherCompatibleWithMe`. En caso que esta también retorne `DONTKNOW`, se consideran las mismas incompatibles.

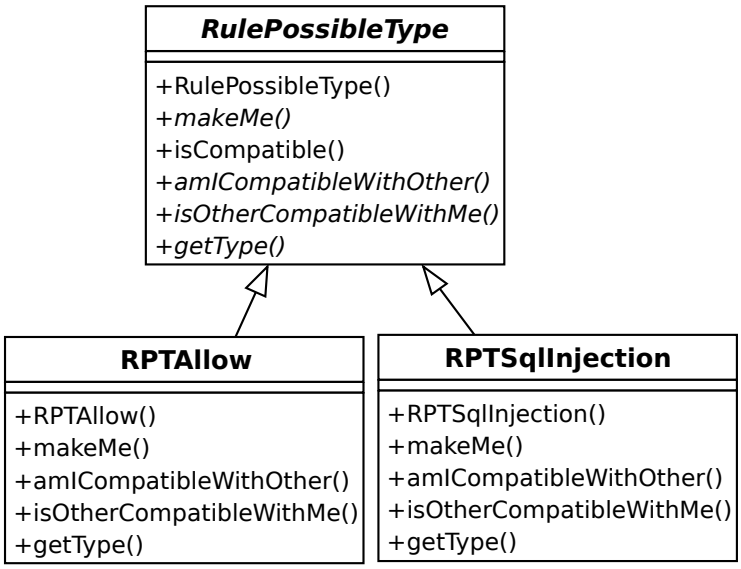


Figura 32 - Patrón Template Method aplicado en DEPSA

Ante la detección de incompatibilidades, ya sean errores o advertencias, se sigue la ejecución normal del *framework* y se devuelve al finalizar un resumen con todas las advertencias encontradas para darle la mayor cantidad de información posible al usuario.

Si hay dos reglas incompatibles aplicadas a un mismo recurso y poseen la misma prioridad, se considera un error debido a que el generador no sabrá cual debe aplicar. Si dichas reglas poseen prioridades distintas se considera una advertencia ya que el generador tendrá información suficiente para determinar que acción tomar con ellas. En el caso en que las reglas incompatibles aplicadas a un recurso se encuentren definidas una en el sistema y otra en el propio recurso se considera una advertencia y siempre predominará la regla aplicada al recurso.

5.3.5 Particularidades del generador ModSecurity

En ModSecurity al instalar el CRS de OWASP las reglas definidas se activan para todo el sistema. Cuando no se desea que una regla se active para un atributo en particular, la estrategia de ModSecurity consiste en desactivarla mediante excepciones definidas en la configuración. Esto contrasta con el enfoque del *framework*, donde por cada recurso se definen las reglas que se le debe aplicar.

Ambos enfoques entran en conflicto cuando a un recurso se define una regla del CRS mientras que a nivel de sistema no. Por ejemplo en LeagueManager, al atributo nombre de la entidad Jugador se define el tipo de regla XSS, para que no se explote dicha vulnerabilidad. Para el resto de los recursos de LeagueManager no se desea proteger de esa vulnerabilidad, por lo que no es definida a nivel de sistema. Cuando esta política de seguridad debe ser generada para ModSecurity, todos los

recursos excepto nombre de jugador deberían desactivar la regla de XSS. Estos recursos no conocen la existencia de la regla que se debe desactivar, debido a que en el enfoque del framework, se definen solo las reglas que se deben aplicar.

Es necesario que todos los recursos informen a un objeto central qué reglas del CRS se le debe aplicar y cuales no, para que luego tome la decisión. Se define como objeto central al recurso sistema debido a que conoce las reglas que se le debe aplicar a todos los recursos.

Este funcionamiento se basa en el patrón *Observer*, como se ve en la *Figura 33*, donde los recursos (`ModSecurityResource`) le informan al sistema (`ModSecuritySystemResource`) si se les debe aplicar alguna regla del CRS o si tienen alguna regla que es incompatible con ellos y no se le debe aplicar.

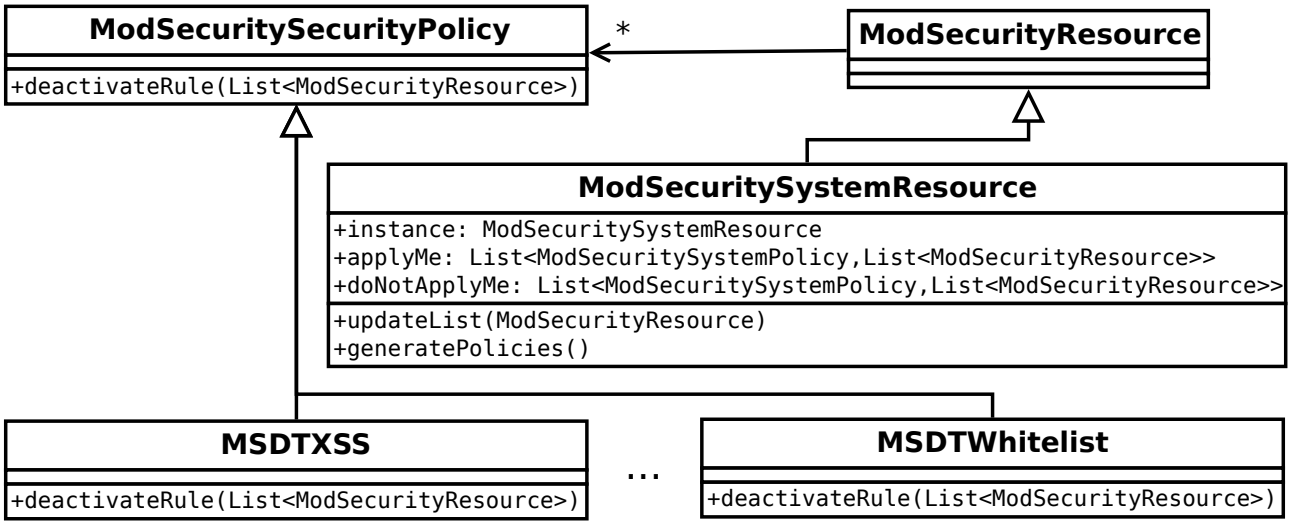


Figura 33 - Adaptación de patrón Observer utilizada en DEPSA

Luego que el sistema ha sido informado por todos los recursos, se invoca la función `generatePolicies`, para desactivar las reglas de acuerdo al pseudocódigo del Código 7.

```
FOREACH(CRSRuleType){
      //las reglas del CRS vienen por defecto prendidas
      IF(Regla aplica a Sistema){
            //Se desactiva unicamente para todos los recursos que han indicado
            //que tienen problemas con la política al tiempo que
            //no han especificado explicitamente que desean que se le aplique
            FOREACH(Resource in DoNotApplyMeList and NOT in ApplyMeList){
                  Resource.deactivateRule(Regla)
            }
      }ELSE{//Regla no aplicada al sistema
            IF(Existe Recurso al que se le aplica Regla){
                  //Quita la regla de todos los recursos que no deseen la regla
                  FOREACH(Resource not in ApplyMeList){
                        Resource.deactivateRule(Regla)
                  }
            }ELSE{//Nadie desea la regla
                  DesactivarRegla
            }
      }
}
```

Código 7 - Pseudocódigo de la función generatePolicies

5.4 Lenguajes definidos

Para definir los lenguajes se empleó EBNF (*Extended Backus–Naur Form*) [53], un metalenguaje que sirve para definir gramáticas libres de contexto, es decir, lenguajes formales. Si bien XML presenta ventajas con respecto a los otros lenguajes estudiados en la sección de 3.5.1 Lenguajes para el intercambio de información, este también es definido por una gramática en EBNF [54]. Se comenzó la definición de los lenguajes en XML pero al investigar herramientas para reconocerlos, se detectó la existencia de un gran número de herramientas que asisten en el reconocimiento de los lenguajes definidos en EBNF. Empleando EBNF no es necesario definir todas las etiquetas definidas en XML, que pueden hacer menos comprensible el lenguaje al ser interpretado por una persona en lugar de una computadora.

Un lenguaje expresado en EBNF consiste en una especificación de un conjunto de reglas de derivaciones de la forma expresada en Código 8, donde “símbolo” es un nodo no terminal, y “__expresión__” es una posible sustitución para el símbolo a la izquierda que consiste en secuencias de símbolos o secuencias separadas por la barra vertical '|' que indica una opción. Los símbolos que nunca aparecen en un lado izquierdo son terminales o marcas del lenguaje que deben aparecer, no son opcionales y se deben escribir tal cual se define.

```
símbolo : __expresión__
```

Código 8 - Regla de EBNF

A continuación se ampliará en las definiciones de SPLang y MapLang, los lenguajes de definición de políticas de seguridad sobre el modelo de la aplicación y de mapeos entre los atributos del

modelo de la aplicación y los diferentes dominios donde se aplica el *virtual patching* respectivamente. En ambos lenguajes se emplean expresiones regulares que cumplen con PCRE (*Perl Compatible Regular Expressions*) [55], debido a la conveniencia de que coincidiera con el utilizado por ModSecurity.

5.4.1 SPLang

Se define el lenguaje SPLang para expresar las políticas de seguridad sobre el modelo de la aplicación y no sobre los dominios particulares de las herramientas de *virtual patching*, como sucedería por ejemplo si se deseara reutilizar SPDL. Los módulos de mapeos y de las políticas de seguridad se mantienen independientes y además de esta manera sería más sencillo agregar este lenguaje a UML porque se definiría sobre los mismos elementos del diagrama. Su definición completa se encuentra en Anexo I - SPLang, y en esta sección se detallarán las definiciones más relevantes para la comprensión del mismo.

La política de seguridad consiste en un conjunto de reglas definidas sobre el modelo de la aplicación que impactan en los distintos atributos u operaciones de una clase. Estos recursos deben poder ser identificados entre los restantes del modelo, por lo que para su definición se requiere la especificación del paquete donde se encuentra la clase, el nombre de la misma y el nombre del atributo o la operación. Como muchos lenguajes de programación permiten la sobrecarga de métodos, el nombre de la operación no es suficiente y se debe especificar los parámetros que esa operación recibe. Por sobrecarga de métodos se entiende cuando una clase puede presentar dos operaciones o métodos con igual nombre pero diferentes parámetros (cantidad o tipos de datos diferentes). Del análisis de las vulnerabilidades y de las herramientas de *virtual patching* se desprende que puede ser útil la definición de reglas que apliquen a todo el sistema que se está protegiendo.

El recurso en SPLang se define como muestra el Código 9.

```
resource : 'SYSTEM'
         | 'PACKAGE_NAME' packageName 'CLASS_NAME' className 'ATTRIBUTE'
           attributeName
         | 'PACKAGE_NAME' packageName 'CLASS_NAME' className 'OPERATION'
           operationName '(' (parameters)? ')'
         ;
```

Código 9 - Definición de recurso en SPLang

Las reglas `packageName`, `className`, `attributeName` y `operationName` se definen como una expresión que admite números, letras y algunos caracteres especiales como [. / - _]. Mientras que `parameters` es una lista de parámetros separados por coma y cada uno de ellos admite también números, letras y caracteres especiales. En Código 10 por ejemplo, para la aplicación LeagueManager se definen los recursos que representan el sistema y el atributo apellido de la clase Persona respectivamente.

```
RESOURCE SYSTEM
RESOURCE PACKAGE_NAME "uy.com.leaguemanager.entity" CLASS_NAME Persona
                                                    ATTRIBUTE apellido
```

Código 10 - Ejemplo de recursos de LeagueManager, expresados en SPLang

Una regla de la política de seguridad se define como una lista con uno o más restricciones o

permisos a aplicar sobre un recurso. El recurso se representa con la derivación `resource` antes definida y la regla `rulePermissionRole` que agrupa las definiciones de las restricciones y los permisos, como se aprecia en el Código 11.

```
ruleDefinition : 'RULE' 'RESOURCE' resource (rulePermissionRole)+ ;
```

Código 11 - Regla ruleDefinition de SPlang

En la sección 4.3 Políticas de seguridad se define que los tipos de reglas en el alcance de este proyecto son restricciones del tipo *Default*, *Specific* y *Permission*. Las primeras dos categorías se agrupan como restricciones (*constraint* en inglés) y la última en permisos para que se puedan definir más categorías en cada una de ellas y extender el lenguaje de manera sencilla.

Para que una regla del tipo restricción pueda ser aplicada a varios atributos y/u operaciones a la vez sin necesidad de escribir la regla para cada uno, se permite definir `DataTypeValidation`, con la regla definida en Código 12 que contiene un identificador único, un nombre y la propia regla. De esta manera a las reglas se agrega una referencia al `DataTypeValidation` para aplicar a ese recurso.

```
dataTypeValidation : 'DATA_TYPE_VALIDATION' 'ID' dataTypeValidationId 'NAME'
                     name 'IO_VALIDATION_CONSTRAINT' ioValidationConstraint ;
```

Código 12 - Regla dataTypeValidation de SPlang

Para cada restricción se define un campo llamado prioridad que es empleado para determinar el orden o prioridad de cada regla dentro de la política de seguridad. Se consideró tomar el orden en que se encuentran escritas en el archivo y otorgarle mayor prioridad a las que vienen primero, pero esto le quita poder al especialista en seguridad que define las políticas dado que si se incorpora el lenguaje a una interfaz de usuario, es esta quien define el orden. Por eso se emplea el campo prioridad, y en caso que se reutilicen estereotipos ya definidos de UML no se requeriría modificar los programas existentes para que reparen en el orden de las reglas.

Como se observa en Código 13, además se define la dirección del mensaje a inspeccionar que puede ser entrada, salida o ambos; el impacto de que esa regla falle de proteger al sistema y permite ingresar un comentario que sea de utilidad para el especialista en seguridad.

```
constraint : 'PRIORITY' priority 'COMMUNICATION_DIRECTION'
             communicationDirection constraintDesc 'IMPACT' impact 'COMMENT'
             comment ;
```

Código 13 - Regla constraint de SPlang

La descripción de la restricción (`constraintDesc`) es la categoría que agrupa las restricciones que realizan validaciones en los datos transferidos desde y hacia la aplicación, donde se definen dos subcategorías, la que implica aplicar reglas conocidas para vulnerabilidades conocidas y las que son validaciones específicas personalizadas. Estas últimas refieren a validaciones de datos de entrada y son las mencionadas en la sección 3.4.2 Validación de entradas, *Blacklist*, *Whitelist y Sanitization*. En dicha categoría se debería definir aquellas nuevas subcategorías que apliquen a los datos.

Para proteger a todo el sistema del 1.1 Caso de estudio de SQL Injection y en particular al atributo `cuerpo` de la clase `Noticia` de posibles ataques de Cross Site Scripting, el especialista en seguridad define en la política de seguridad del sistema las reglas expresadas en Código 14.

Para el caso de *Blacklist* se define una lista de cadenas de caracteres que la herramienta de *virtual patching* no debería dejar pasar hacia la aplicación. Cada una de estas puede ser una expresión

```
RULE RESOURCE SYSTEM
      CONSTRAINT PRIORITY 2 COMMUNICATION_DIRECTION IN SQL INJECTION IMPACT
                  SEVERE COMMENT "Constraint SQLINJECTION a nivel de sistema"
RULE RESOURCE PACKAGE_NAME "uy.com.leaguemanager.entity" CLASS_NAME Noticia
                                                          ATTRIBUTE cuerpo
      CONSTRAINT PRIORITY 1 COMMUNICATION_DIRECTION BOTH XSS IMPACT HIGH COMMENT
                              "Constraint XSS al atributo cuerpo de Noticia"
```

Código 14 - Ejemplos de reglas Default definidas para LeagueManager

regular para detectar un patrón que no se desea que sea ingresado. Al sanitizar una entrada se puede emplear distintas soluciones, una de ellas es escapar ciertos caracteres para que en caso de que puedan ser una amenaza, al aplicarles una transformación, dejen de serlo. Por ejemplo en HTML `<script>` es codificado como `&lt;script&gt;` los caracteres < y > que podrían ser una amenaza, se los puede codificar como `&lt;` y `&gt;` respectivamente. Por eso se permite ingresar una lista de cadenas de caracteres y sus respectivos valores codificados. También se permite que se ingrese sólo el primer valor indicando funciones particulares de la herramienta de *virtual patching*, por ejemplo `htmlspecialchars()` de PHP que realiza el cambio de caracteres especiales de HTML tal cual se describe anteriormente. La lista de valores es opcional, porque existe otra forma de uso que consiste en evitar que se escriba en los *logs* los valores del atributo especificado, por ejemplo para el caso de contraseñas y otros datos que puedan considerarse sensibles. Dependerá del generador de reglas de cada herramienta en particular como procesar estos datos.

Aplicado en el caso de estudio, se presenta en el Código 15 un ejemplo de *Blacklist* que bloquea los mensajes que lleguen a la aplicación con un carácter "`'`", para el atributo título de la clase Noticia. En la misma política de seguridad se agrega la regla de S*anitization* para evitar que el atributo password de la clase Usuario sea escrita en los *logs*, como muestra el Código 16.

```
RULE RESOURCE PACKAGE_NAME "uy.com.leaguemanager.entity" CLASS_NAME Noticia
                                                          ATTRIBUTE titulo
      CONSTRAINT PRIORITY 3 COMMUNICATION_DIRECTION IN IO_VALIDATION
            BLACKLIST "'" IMPACT MEDIUM COMMENT "Blacklist titulo de Noticia"
```

Código 15 - Blacklist para el atributo password de la clase Usuario

```
RULE RESOURCE PACKAGE_NAME "uy.com.leaguemanager.entity" CLASS_NAME Usuario
                                                          ATTRIBUTE contrasena
      CONSTRAINT PRIORITY 4 COMMUNICATION_DIRECTION IN IO_VALIDATION
               SANITIZATION IMPACT SEVERE COMMENT "Sanitization de contrasena"
```

Código 16 - Sanitization para el atributo contrasena de la clase Usuario

Además se define una regla *Whitelist* para el atributo edad de la clase `Persona` para permitir ingresar al sistema sólo valores enteros de hasta dos dígitos. En SPLang se definen cuatro de los 5 valores analizados en la sección 3.4.2 Validación de entradas. El atributo con el formato (`formatAttribute`) posee una expresión regular que sirve para detectar los valores válidos que acepta el recurso. Para el caso de la edad, la expresión regular definida es verdadera cuando el parámetro tiene uno o dos dígitos como muestra el Código 17. Se expresa el tipo de dato que corresponde al atributo (entero o *integer*), el dominio que acota el tipo de datos (cada dígito es un carácter del conjunto [0..9]) y el largo (dos debido que son dos dígitos) que puede ser útil para determinar el largo máximo permitido. Se descartaron los valores razonables que puede tener el atributo ya que no son un dato preciso para emplear a la hora de generar la regla, sino que es de utilidad para el desarrollador que escribe la política.

```
RULE RESOURCE PACKAGE_NAME "uy.com.leaguemanager.entity" CLASS_NAME Persona
                                                              ATTRIBUTE edad
      CONSTRAINT PRIORITY 3 COMMUNICATION_DIRECTION IN IO_VALIDATION WHITELIST
                  FORMAT "^\d{1,2}$" TYPE "Integer" DOMAIN "[0-9]" LENGTH 2
                              IMPACT MEDIUM COMMENT "Whitelist edad de Persona"
```

Código 17 - Whitelist para el atributo edad de la clase Persona

Como se mencionaba en la definición de las reglas (`ruleDefinition`), a un recurso se puede aplicar un conjunto de permisos, que sigue la definición de RBAC. Consiste en definir la relación del permiso con los roles, de manera que a un atributo u operación se le asigna un permiso para un conjunto de roles. Los tipos de permisos pueden ser extendidos, por ejemplo agregando permisos de escritura, lectura y ejecución que son de utilidad en un contexto de base de datos por ejemplo. Con esta definición se podrían generar reglas de permisos que aplican a todo el sistema, ya que un recurso puede ser el sistema. Por ejemplo en LeagueManager se decide que el atributo email de Persona no lo pueda acceder cualquier rol, salvo el rol de administrador. Para eso se debe definir en la política de seguridad lo expresado en Código 18.

```
RULE RESOURCE PACKAGE_NAME "uy.com.leaguemanager.entity" CLASS_NAME Persona
                                                              ATTRIBUTE email
      PERMISSION ALLOW ROLE "administrador"
      PERMISSION DENY ROLE "usuario"
```

Código 18 - Control de acceso para el atributo email de la clase Persona

El control y gestión de la definición de usuarios y los roles que posee cada uno se dejan por fuera del lenguaje porque la manera de implementarlo depende de cada herramienta que se emplee para realizar el *virtual patching*.

5.4.2 MapLang

El lenguaje para realizar el mapeo comienza con la definición de la versión del mismo, un identificador del modelo mapeado y una lista de dominios, los cuales contienen toda la información referente al mapeo entre los datos del modelo de la aplicación y los de la herramienta que implemente el *virtual patching*, que puede ser WEB, DB u otros. El generador de reglas de la herramienta en particular deberá conocer a que codominio pertenecen sus reglas para obtener esos mapeos específicos y no los restantes. En el Anexo II - MapLang se encuentran los detalles del lenguaje completo.

La definición de la regla de cada dominio, como muestra el Código 19, define su `id` que es una cadena de caracteres y el recurso “SISTEMA” que contiene la ruta a los recursos a proteger y servirá para diferenciar la aplicación a la cual se debe aplicar la política de seguridad. A modo de ejemplo la liga deportiva cuenta con LeagueManager y una aplicación con servicios para los proveedores de equipamientos de los clubes, ambas desplegadas en un único servidor web.

```
domain : 'DOMAIN' 'DOMAIN_ID' domainId ('SYSTEM' system)? (resourceMapping)+ ;
```

Código 19 - Regla domain de MapLang

En el Código 20 se aprecia la definición del recurso sistema para la aplicación LeagueManager que se encuentra desplegada en `liga.com/LeagueManager/`, y la de la aplicación para proveedores que se encuentra en `liga.com/ServiciosProveedores/`. Con respecto a bases de datos se las puede diferenciar por el esquema definido. Por último se define una lista de relaciones con los mapeos entre los recursos (`resourceMapping`).

```
DOMAIN DOMAIN_ID "WEB" SYSTEM "/LeagueManager/"
DOMAIN DOMAIN_ID "WEB" SYSTEM "/ServiciosProveedores/"
DOMAIN DOMAIN_ID "DB" SYSTEM "LeagueManagerDB"
DOMAIN DOMAIN_ID "DB" SYSTEM "Proveedores"
```

Código 20 - Ejemplos de recursos "sistema" para LeagueManager

Dado que las políticas de seguridad se definirán sobre los recursos del modelo de la aplicación, se necesita conocer a qué recursos del codominio se mapea cada uno, en general, se mapea a uno o más recursos del codominio. A modo de ejemplo, en el dominio WEB, el atributo password de la clase Usuario de LeagueManager aparece en varias páginas, en las que es posible autenticarse. La regla definida en ANTLR se aprecia en Código 21. El recurso o "*resource"* se define de la misma manera que en SPLang para que sea más sencillo identificar cuando se define el mismo recurso del modelo de aplicación en uno y el otro aunque no es requerimiento que sean iguales.

```
resourceMapping : 'RESOURCE' resource 'RELATED_RESOURCE_LIST'
                                                    (relatedResource)+ ;
```

Código 21 - Regla resourceMapping de MapLang

Por otra parte los recursos del codominio son más abstractos ya que se requiere poder identificar recursos de distintos contextos. Si bien brinda mayor libertad al especialista en seguridad para definirlos, requiere mayor esfuerzo por parte del especialista en el modelo de la aplicación a la hora de identificar los recursos. Se define cada recurso como un contenedor o `container` que sirve para identificar el lugar donde se encuentra el recurso y un identificador del recurso asociado (`relatedResourceId`), como muestra el Código 22.

```
relatedResource : 'CONTAINER_ID' containerId 'RESOURCE_ID' relatedResourceId ;
```

Código 22 - Regla relatedResource de MapLang

Como contenedor se hace referencia a la URI donde se encuentra el atributo para el caso del dominio WEB, el esquema y tabla de base de datos para el dominio DB, lo que demuestra la complejidad de dicha definición y que dependen de la herramienta de *virtual patching* a emplear y el contexto de la misma. Solo restaría para identificar al recurso en este contexto, el nombre o identificador del propio recurso. Estas reglas se definen como cadenas de caracteres.

En Código 23 se detalla un fragmento del mapeo realizado para la aplicación LeagueManager para dos dominios de herramientas de *virtual patching* diferentes "WEB" y "DB".

Para el recurso de tipo atributo llamado `password` de la clase `Usuario` se definen tres recursos en el dominio WEB: el de identificador `password` de la página `home.php`, `noticias.php` y `jugadores.php`. Esto se debe a que el login de la aplicación se puede efectuar en cualquiera de estas páginas. Para el dominio DB, para el mismo atributo se define el mapeo a la columna `PASSWORD` de la tabla `USUARIOS`. La definición de sistema en el dominio WEB es la URI que identifica a esa aplicación de las demás en el servidor. De la misma manera en el dominio DB, `LeagueManagerDB` es el identificador de la base de datos de la aplicación.

```
MapLang VERSION:1.0 MODEL "Modelo1 – LeagueManager"
      DOMAIN_LIST
            DOMAIN DOMAIN_ID "WEB" SYSTEM "/LeagueManager/"
                  RESOURCE PACKAGE_NAME "uy.com.leaguemanager.entity" CLASS_NAME
                                                Usuario ATTRIBUTE password
                  RELATED_RESOURCE_LIST
                        CONTAINER_ID "home.php" RESOURCE_ID "password"
                        CONTAINER_ID "noticias.php" RESOURCE_ID "password"
                        CONTAINER_ID "jugadores.php" RESOURCE_ID "password"
            DOMAIN DOMAIN_ID "DB" SYSTEM "LeagueManagerDB"
                  RESOURCE PACKAGE_NAME "uy.com.leaguemanager.entity" CLASS_NAME
                                                Usuario ATTRIBUTE password
                  RELATED_RESOURCE_LIST
                        CONTAINER_ID "USUARIOS" RESOURCE_ID "PASSWORD"
```

Código 23 - Definición de mapeos para LeagueManager

En el siguiente capítulo se describen los aspectos más relevantes de la implementación de DEPSA.

[Página dejada en blanco intencionalmente.]

6 Implementación

Este capítulo comienza con la descripción de las tecnologías utilizadas para la realización de DEPSA y LeagueManager. Además se presentan las particularidades de la implementación de los lenguajes SPLang y MapLang empleando una herramienta para el reconocimiento de lenguajes expresados mediante una gramática formal.

Se plantean las alternativas analizadas para el uso del CRS de OWASP que permiten mitigar las diferencias entre el enfoque del *framework* y el de ModSecurity, para posteriormente detallar su instalación y la implementación de la generación de reglas.

Por último se dedica una sección para describir los pasos necesarios para incorporar nuevos tipos de reglas a la implementación.

6.1 Tecnologías utilizadas

DEPSA fue desarrollado utilizando el entorno de desarrollo Eclipse Luna y el lenguaje de programación JAVA, con una máquina virtual openJDK 7u79. Como librería externa fue empleado ANTLR versión 4.4, para realizar el procesamiento de los lenguajes.

Durante el desarrollo del proyecto fue utilizado el compilador de Eclipse, pero se proveen compiladores realizados en Apache Ant versión 1.9.3.

El archivo de configuración generado por el prototipo fue ejecutado en un servidor Apache versión 2.4.7 sobre un sistema operativo Ubuntu Linux 14.04. En el servidor apache fue instalado el módulo de ModSecurity versión 2.7.

LeagueManager fue implementado utilizando el entorno de desarrollo Netbeans 8, en el lenguaje de programación PHP versión 5.5. La aplicación cuenta con accesos a una base de datos MySQL, y es desplegado en el mismo servidor apache donde se instaló el módulo de ModSecurity.

6.2 Implementación de los lenguajes definidos

Para el reconocimiento de ambos lenguajes se empleó la herramienta ANTLR (*ANother Tool for Language Recognition*) [56] que genera automáticamente un *lexer* o analizador lexicográfico y un *parser* o analizador sintáctico a partir de la descripción del lenguaje. El primero se encarga de convertir el texto ingresado en una secuencia de caracteres que tienen un significado, llamados *tokens* o componentes léxicos. Por otro lado el analizador sintáctico recibe como entrada los *tokens* agrupándolos jerárquicamente en una estructura del tipo árbol al tiempo que valida la sintaxis. Lo realiza con un enfoque *top-down* (recorre las derivaciones de arriba hacia abajo, buscando una que aplique) y de izquierda a derecha, tomando la derivación de más a la izquierda. En la Figura 34 se puede ver como a partir de los caracteres “SQL INJECTION”, el *lexer* los convierte en un conjunto de *tokens* (SQL e INJECTION) y luego el *parser* se encarga de agruparlo en un árbol.

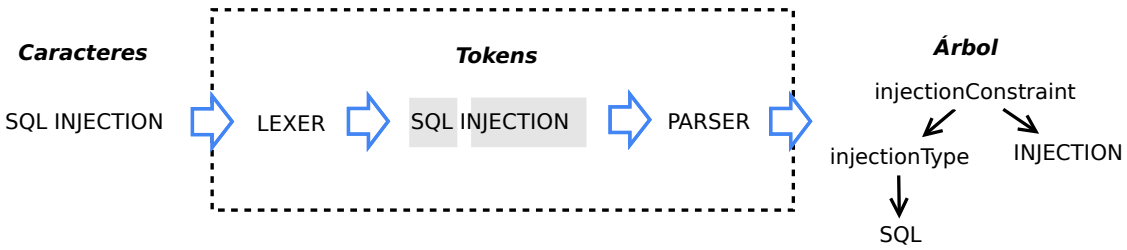


Figura 34 - Ejemplo de lexer y parser de ANTLR

ANTLR además provee dos maneras de recorrer los árboles del analizador sintáctico `ParseTreeListener` y `ParseTreeVisitor` generando automáticamente las interfaces y una implementación básica de los mismos para que solo sea necesario implementar aquellos métodos que se desea especificar cómo deben funcionar. El *listener* responde a los eventos que genera el *parser* al ingresar y al salir de un nodo del árbol. Posee la ventaja que al emplearlo se recorre el árbol automáticamente y cada vez que se visita un nodo del árbol se ejecutan dichas funciones. En cambio el *visitor* genera un método `visit` para cada regla pero permite que se controle el recorrido del árbol manualmente, teniendo el programador que realizar las llamadas a la función `visit` de los nodos hijos del árbol.

Para describir el lenguaje se necesita definir una gramática que es un conjunto de reglas que definen la sintaxis del lenguaje expresado en EBNF. Se profundizará en las principales características de MapLang y SPLang y la manera de extenderlo.

Como se observa en el Código 24, una regla del lenguaje para definir los tipos de inyecciones es `injectionConstraint` que consiste de una subregla `injectionType` y de un literal o nodo final `INJECTION`. A su vez se define `injectionType` como los literales `SQL`, `HTML` o `JAVASCRIPT`.

```
injectionConstraint     : injectionType 'INJECTION' ;
injectionType           : 'SQL'
                    | 'HTML'
                    | 'JAVASCRIPT'
                    ;
```

Código 24 - Ejemplo de regla de SPLang en ANTLR

Para que se cumpla la sintaxis del lenguaje, el texto ingresado debe ser '`SQL INJECTION`', '`HTML INJECTION`' o '`JAVASCRIPT INJECTION`'. Si no cumple con esta regla, continúa su intento con las demás existentes del lenguaje y si no encuentra coincidencia, informa del error y la posición del texto ingresado donde no puede reconocer una regla correcta.

Para cumplir con lo definido en la sección 5.2 Interfaces definidas, se genera una estructura de DTOs (Data Transfer Object son objetos que pasan datos entre procesos sin contener lógica ni validaciones) que van a ser instanciados a medida que se recorre el árbol y son resultado de los módulos `Mappings` y `SecurityPolicies`. De esta manera si en el futuro se desea cambiar ANTLR, sólo se debe procurar que la nueva herramienta para reconocer el lenguaje retorne la misma estructura de datos.

Como es necesario generar la estructura de datos de cada lenguaje, el método definido para recorrer el árbol que genera el analizador sintáctico es el `ParseTreeVisitor`, debido a que cuando se visita un nodo se genera el objeto correspondiente en la estructura y es el retorno de esa función. ANTLR genera una interfaz básica donde todas las operaciones deben retornar el mismo tipo de dato, por lo que se define que sea `Object` que es la clase de la cual heredan todas las clases en Java por defecto y no se requiere definir esta herencia.

Para instanciar los DTOs de cada nodo, se sobreescribe el método de la interfaz básica, se agrega el código que realiza dicha tarea y se recorren los nodos hijos manualmente. Por ejemplo en Código 25 se observa el código del método *visit* del nodo *Whitelist* del lenguaje que se definió en Código 17. Cuando el `ParseTreeVisitor` pasa por un nodo *Whitelist*, invoca a dicha función y se instancia un objeto de tipo *Whitelist* definido en la estructura `SecurityPolicies`. Se define que

el atributo `formatAttribute` del objeto contiene el objeto resultante de visitar el nodo `FormatAttribute`. De la misma manera se visitan los nodos `TypeAttribute`, `DomainAttribute` y `Length`, almacenando el resultado en los parámetros del objeto *Whitelist*. Por último se retorna el nuevo objeto creado, que debió ser invocado por un nodo padre de *Whitelist*, dado que el árbol se recorre manualmente. De esta manera se genera una estructura de datos que es independiente del lenguaje y el módulo que la consume no debe conocer las particularidades del recorrido del árbol que aquí se describe.

```
@Override
public Object visitWhitelist(WhitelistContext ctx) {
      Whitelist whitelist = new Whitelist();
      whitelist.setFormatAttribute((String) visit(ctx.formatAttribute()));
      whitelist.setTypeAttribute((String) visit(ctx.typeAttribute()));
      whitelist.setDomain((String) visit(ctx.domainAttribute()));
      whitelist.setLength((Integer) visit(ctx.length()));
      return whitelist;
}
```

Código 25 - Método visit del nodo Whitelist

Por defecto ANTLR envía los mensajes de error a la salida por defecto del sistema, pero permite definir un método particular. Para los errores de sintaxis se debe implementar una clase que extienda `org.antlr.v4.runtime.BaseErrorListener` y sobrescribir el método `syntaxError` que muestra el Código 26 y de esta manera poder manejar la excepción como se desee.

```
void syntaxError(Recognizer<?, ?> recognizer, Object offendingSymbol, int line,
                 int charPositionInLine, String msg, RecognitionException e)
```

Código 26 - Método syntaxError

En los módulos `SecurityPolicies` y `Mappings` se define la clase `ThrowingErrorListener` para lanzar una excepción que indique que ocurrió un error a los módulos que realizaron la invocación y en base a esto poder actuar. Para mantener información de la línea, la posición dentro de la línea, el mensaje y demás datos, se debe implementar una excepción particular que pueda guardar esos mensajes (`SPLangSyntaxErrorException` y `MapLangSyntaxErrorException`).

Luego de generadas las clases automáticas de ANTLR e implementar los métodos necesarios, sólo resta implementar la funcionalidad principal de cada módulo de lenguaje. Esta consiste en abrir el archivo con las definiciones de las políticas de seguridad o el mapeo especificado según corresponda, crear una instancia del *Lexer* autogenerado para dicho lenguaje y configurar el manejo de errores particulares que se nombraban anteriormente. Luego de obtener los *tokens*, se pasa al *Parser* para que genere un árbol. Se recorre el mismo a partir del nodo inicial del lenguaje, retornando la estructura de datos definida en sus interfaces.

6.3 CRS de OWASP

Como fue detallado en la sección 5.3.5 Particularidades del generador ModSecurity, existen distintos enfoques sobre como aplicar las reglas entre ModSecurity y el *framework*.

Dado que el *framework* no debía ser diseñado solamente contemplando ModSecurity, el enfoque de DEPSA consiste en definir las reglas a aplicar en los recursos y no la de definir excepciones. Entonces, para hacer uso del *Core Rule Set* de OWASP (CRS) en DEPSA, fueron evaluadas dos alternativas:

1. Incluir el CRS en el servidor donde se encuentra instalado el ModSecurity.
2. Incluir las reglas del CRS en los archivos generados por DEPSA.

La alternativa 1 se encuentra en un entorno restringido ya que las reglas del CRS aplican a todos los recursos sin agregar definiciones en la política de seguridad, por lo tanto, si algún recurso no tiene definida un regla en la política es necesario generar excepciones.

Una de las ventajas que presenta esta alternativa es que al surgir actualizaciones del CRS no es necesario modificar el archivo de configuración generado por DEPSA, esto es porque la definición está contenida en el servidor y no en el archivo, en donde solo hay referencia a las reglas definidas. Además, al no modificarse el código generado para el CRS, no se introducen nuevos errores y como desde el archivo de configuración se hace una referencia a las reglas del CRS contenidas en el servidor, esto genera archivos más pequeños y manejables. Como aspectos negativos se destacan que las políticas de seguridad no dependen solo de los archivos de configuración generados, sino que además se depende de la instalación y configuración de ModSecurity y CRS. Se agrega que en el caso de que a nivel del sistema se desee que no se aplique una regla, por ejemplo las reglas de *SQL injection*, pero sí se desea que se apliquen para un campo particular, es necesario agregar excepciones a las reglas de SQL *injection* para todo el sistema. Sin embargo al hacer esto el recurso que sí se quería que contara con la protección queda desprotegido, por lo que es necesario agregar nuevamente las reglas para ese recurso en particular. Esto no es posible en ModSecurity ya que no es posible agregar una regla sobre la que ya se creó una excepción. En la versión 2.6.0 de ModSecurity existía una configuración para agregar el recurso en la transacción, pero fue quitada en la siguiente ya que no funcionaba de forma correcta. La solución para este escenario es conocer todos los recursos con los que cuenta el sistema para agregar excepciones recurso por recurso o aplicar lo que se propone como alternativa 2 para la instalación del CRS.

En la alternativa 2, el servidor no cuenta con el CRS instalado, por lo es en un entorno permisivo. Para agregar las reglas del CRS se debe generar un mecanismo para obtener las reglas que están definidas y agregarlas al archivo de configuración generado por DEPSA.

Las ventajas de esta alternativa son que los datos que se encuentran en el archivo resultante coinciden con los que se encuentran definidos en la política de seguridad para el sistema, por lo que no se depende de la instalación y configuración de ModSecurity ni del CRS. Las desventajas son que al surgir actualizaciones del CRS se debe volver a generar el archivo de configuración, y puede ser necesario realizar algún cambio más, dependiendo de cómo se realice la actualización de las reglas contenidas en el CRS dentro de DEPSA. Otra desventaja es que al manipular las reglas que están definidas en el CRS, se corre el riesgo de introducir errores y como los archivos configuración contienen las reglas del CRS definidas para cada recurso provoca archivos de configuración de mayor tamaño y más difíciles de mantener.

Al realizar la implementación de DEPSA se optó por la opción de utilizar el CRS instalado dentro de ModSecurity por las razones antes planteadas y porque para poder extraer las reglas expuestas, era necesario realizar un análisis del CRS, que escapaba a los objetivos del proyecto.

6.4 Generación de reglas para ModSecurity

ModSecurity funciona como cualquier *firewall* tradicional: se define una lista de reglas y cuando

llega un mensaje hacia la aplicación, recorre dichas reglas de arriba hacia abajo y si se cumplen las condiciones especificadas en la regla se ejecutan las acciones que en ella se definan. Estas reglas se pueden definir en varios archivos, debiéndose indicar los archivos que contienen las reglas definidas y serán tomados en el orden de aparición, como se explica a continuación.

Luego de instalado ModSecurity, se encuentra el archivo `modsecurity.conf`, con la configuración por defecto, y estas son las primeras reglas que lee. Por defecto funciona en modo detección y solo guarda *logs* de la información de las reglas definidas, pero para realizar el *enforcement* es necesario cambiar el valor `SecRuleEngine On`.

Las reglas del CRS son instaladas en un directorio, agrupadas en subdirectorios (reglas activadas, reglas base, reglas experimentales, archivos lua, reglas opcionales, reglas de estudio de SpiderLabs y utilidades). Al igual que con ModSecurity, el CRS provee un archivo base de configuración (`modsecurity_crs_10_setup.conf`) y se lo debe incluir para activar cualquier otra regla del CRS.

Para activar todas las reglas de ModSecurity en el servidor donde se encuentre instalado, se debe generar un archivo de configuración, que para el caso de *Apache* HTTPD, es `${apache2}/mods-enabled/mod-security.conf`. Como se aprecia en el Código 27 se incluyen las reglas por defecto de ModSecurity, la regla base y las activadas del CRS y por último se deben agregar las reglas específicas que genera el *framework* para las aplicaciones que se desean proteger. En este caso en particular la aplicación es LeagueManager, el sistema operativo es Ubuntu 14.04 (las rutas de los archivos de configuración varían según el mismo) y las reglas específicas fueron agregadas al archivo `leaguemanager.conf`.

```
<IfModule security2_module>
    # Include all the *.conf files in /etc/modsecurity.
    # Keeping your local configuration in that directory
    # will allow for an easy upgrade of THIS file and
    # make your life easier
    IncludeOptional /etc/modsecurity/*.conf
    # OWASP CRS
    Include "/usr/share/modsecurity-crs/*.conf"
    Include "/usr/share/modsecurity-crs/activated_rules/*.conf"
    # Specific rules for LeagueManager app
    IncludeOptional /etc/modsecurity/specificrules/leaguemanager.conf
</IfModule>
```

Código 27 - Configuración de ModSecurity

Para activar una regla del CRS solo se debe crear un enlace simbólico en el directorio `${modsecurity-crs}/activated_rules/` y de esta manera sin copiar los archivos, ModSecurity va a configurar las reglas allí definidas. Al actualizarlas, las reglas activadas se sobreescriben y no es necesario otro mantenimiento en nuestra configuración particular.

Para DEPSA se deben crear enlaces en reglas activadas de las reglas de *XSS* y *SQLInjection* del directorio de reglas base (`modsecurity_crs_41_sql_injection_attacks.conf`, `modsecurity_crs_41_xss_attacks.conf`).

Las reglas particulares las generará el módulo ModSecurityGenerator y para ello se deben especificar las reglas definidas para un recurso, así como aplicar excepciones para que las reglas del

CRS no apliquen a los recursos que no la tienen definida.

En el manual de referencia de ModSecurity [15] se puede encontrar la lista de reglas que se pueden definir y una descripción de cómo emplearlas. La regla básica es *SecRule* y en ella se definen las variables a analizar y que operaciones aplicar, cuyo formato se presenta en Código 28. Si al evaluar la operación aplicada a los parámetros da verdadero, se aplican las acciones definidas, sino continúa con la siguiente regla en el archivo.

```
SecRule VARIABLES OPERACIONES [ACCIONES]
```

Código 28 - Regla SecRule de ModSecurity

Las acciones son opcionales debido a la existencia de una regla que sirve para configurar las acciones por defecto `SecDefaultAction`, y si ambas están definidas, se combinan las acciones de ambas. Por ejemplo en el Código 29 se definen las acciones `phase, log, auditlog, pass`. Si una regla no contiene la acción `phase` se aplica la acción por defecto, es decir `phase:2`, pero si tiene definido `phase:1` se considera esta acción definida específicamente. La lista de acciones es bastante amplia y se encuentra explicada en detalle en el manual de referencia antes mencionado.

```
SecDefaultAction "phase:2,log,auditlog,pass”
```

Código 29 - Ejemplo de regla SecDefaultAction de ModSecurity

A continuación se describen las reglas que el módulo ModSecurityGenerator genera, para cada tipo de regla definida en la política de seguridad.

Las reglas de la política de seguridad fueron definidas con 5.4.1 SPLang sobre un recurso del modelo de la aplicación, y estos recursos mapeados con 5.4.2 MapLang a recursos de los distintos dominios de las posibles herramientas de *virtual patching*. A ModSecurity se aplica la lista de recursos del dominio WEB y el módulo ModSecurityGenerator recibe directamente los recursos de dicho dominio, que pasan a ser las variables. Por ejemplo, para la especificación del mapeo del recurso `password` del Código 23, definido en el modelo de LeagueManager, se relaciona con tres recursos en el dominio WEB.

Para que todas las reglas definidas apliquen sólo al dominio de la aplicación en el servidor, tienen en común que se valida que la URI del REQUEST del protocolo HTTP se corresponda con la del recurso, que se encuentra en el atributo “*container*”. Luego se debe aplicar la regla definida sobre el atributo particular que se encuentra en el "*resource*" del recurso definido en el mapeo. La variable `ARGS` contiene todos los argumentos de la carga efectiva del mensaje y para identificar que se aplica al recurso de nombre `_resource_`, se debe escribir `ARGS:_resource_`.

Para que la acción definida se aplique si se cumplen más de una regla, se definen cadenas de reglas, mediante la acción `chain` que actúa como un operador lógico AND. En Código 30 se muestra la regla generada para el atributo `Titulo` de la página `noticias.php` del sistema `"/LeagueManager/"`.

```
SecRule REQUEST_URI "^/LeagueManager/noticias.php" "..., chain,..."
      SecRule ARGS:Titulo "..."
```

Código 30 - Regla para el atributo Titulo

Para el caso de la regla *Blacklist*, se recuerda que su definición en SPLang permitía ingresar una lista de expresiones regulares que definen los patrones que deben ser filtrados y evitar que lleguen a

la aplicación. Por ejemplo, para el recurso del modelo `Titulo` de la clase `Noticias` de LeagueManager, la regla *Blacklist* definida en el Código 15, la regla generada para ModSecurity se

```
SecRule REQUEST_URI "^/LeagueManager/noticias.php" "phase:2, id:20, log, chain,
                                deny, msg:'blacklist matched on Titulo resource'"
      SecRule ARGS:Titulo "(')"
```

Código 31 - Regla Blacklist generada para ModSecurity

presenta en Código 31, aplicada sobre el recurso correspondiente en el dominio "WEB".

En general la lista de expresiones regulares se incorpora a la regla de ModSecurity mediante la disyunción lógica inclusiva de los elementos de la lista que se logra con el caracter “|”. El id debe ser único, por lo que es autogenerado y se incrementa con cada regla que se escribe y de la misma manera aparecerá en las siguientes reglas.

En el caso de *Whitelist* la acción no es bloqueante para los datos que cumplen con el patrón definido en `formatAttribute`, que para el caso del ejemplo del Código 17 es una expresión regular que reconoce 2 dígitos. Sí se deben bloquear todos los datos que no cumplan con este patrón. En Código 32 la regla generada de id 5 identifica el patrón definido y si los datos ingresados lo cumplen, se salta la regla que bloquea cualquier otro dato, identificada con el `id` 6. Para realizar el salto, en ModSecurity se emplea la acción `skipAfter` que motiva que la transacción actual continúe luego de la directiva `SecMarker` con la etiqueta indicada, en este caso `whitelistMarker5`.

```
SecRule REQUEST_URI "^/LeagueManager/jugadores.php" "phase:2, id:5, nolog,
                                      chain, skipAfter:whitelistMarker5, pass"
      SecRule ARGS:Edad "^\d{1,2}$"
SecRule REQUEST_URI "^/LeagueManager/jugadores.php" "phase:2, id:6, log, chain,
                              deny, msg:'whitelist not matched on Edad resource'"
      SecRule ARGS:Edad ".*"
SecMarker whitelistMarker5
```

Código 32 - Regla Whitelist generada para ModSecurity

Otra opción para la validación de datos de entrada era la sanitización. En ModSecurity se puede sanitizar los argumentos impidiendo que éstos sean escritos en los logs mediante la acción `sanitiseMatched`, como muestra el Código 33, definida para el recurso password de la clase Usuario en el ejemplo del Código 16.

```
SecRule REQUEST_URI "^/LeagueManager/noticias.php" "phase:2, id:18, chain,
                                                 nolog, pass, sanitiseMatched"
      SecRule ARGS:password
```

Código 33 - Regla Sanitization generada para ModSecurity

Notar que dado el mapeo del Código 23, el *framework* generará las tres reglas necesarias para cada uno de los recursos definidos en el dominio WEB. Esta es una de las ventajas de definir la política de seguridad sobre el modelo y que se aplique a todos los recursos mapeados mediante 5.4.2 MapLang.

Como fue especificado al inicio de esta sección, las reglas del CRS se deberán configurar en ModSecurity de manera que se encuentren activadas, y se desactivará sólo para los recursos que no posean definida una de esas reglas en la política de seguridad. Para realizar esta tarea existe una

acción `ctl` que sirve para cambiar la configuración para la transacción que se esté ejecutando. Del conjunto de configuraciones posibles, `ruleRemoveTargetByTag` sirve para quitar un recurso de todas las reglas que tengan definido un *tag*. Las reglas del CRS poseen *tags* de acuerdo a su categoría y de esta manera se evita escribir muchas reglas para desactivar todo el conjunto de reglas que incluye una categoría. Sí se especifica solo `ARGS`, como muestra el Código 34, se desactiva Cross Site Scripting para toda la aplicación, sino se debe escribir la regla para cada uno de los recursos como se especifica en el Código 35.

```
SecRule REQUEST_URI "@rx ^/LeagueManager/.*" "phase:1,id:21,t:none,pass,nolog,
                        ctl:ruleRemoveTargetByTag=OWASP_CRS/WEB_ATTACK/XSS;ARGS"
```

Código 34 - Regla que desactiva XSS para toda la aplicación

```
SecRule REQUEST_URI "@rx ^/LeagueManager/noticias.php" "phase:1,id:21,pass,
        nolog,ctl:ruleRemoveTargetByTag=OWASP_CRS/WEB_ATTACK/XSS;ARGS:password"
```

Código 35 - Regla que desactiva XSS sólo para el recurso password

Otro *tag* empleado es `OWASP_CRS/WEB_ATTACK/SQL_INJECTION` que sirve para la regla de s*ql injection*.

6.5 Creación de nuevos tipos de reglas

En esta sección se describirán los pasos necesarios para incorporar nuevos tipos de reglas en DEPSA. La extensión de los lenguajes ya fue descrita en la sección 5.4 Lenguajes definidos. Se detalla el procedimiento que debe seguirse para realizar la modificación de `PreGenerator` y `ModSecurityGenerator`.

Cada módulo define los tipos de reglas de forma independiente como se vio en la sección 5.3.3 Extensibilidad de reglas. Para poder instanciar clases que no existen en el momento de desarrollar el módulo se utiliza la funcionalidad de Java, *reflection* [57], que permite invocar clases de forma dinámica a partir de su nombre.

Al generar un nuevo tipo de regla dentro de SecurityPolicies, existirá una nueva clase `RulePermissionRole` que la representará. Se deberá crear en `PreGenerator` una nueva clase que extienda `PreGeneratedSecurityPolicy` dentro del directorio que se muestra en el Código 36, que es donde `PreGenerator` buscará clases que puedan instanciarse a partir del nuevo `RulePermissionRole` definido. Al implementar la función `makeMe` debe validar que se trate de la `RulePermissionRole` correcta. En caso contrario `PreGenerator` indica que no existe una implementación para el tipo de regla definido.

```
${DEPSA}/CommonsPreGenerator/src/commonspregenerator/datatypes/policies
```

Código 36 - Directorio donde agregar nuevas reglas a nivel PreGenerator

Luego se debe actualizar `RuleResumeTable` para determinar la compatiblidad de esta nueva regla con las ya existentes. Para esto se crea una nueva clase que extienda `RulePossibleType`, dentro del directorio que se indica en Código 37. Deben implementarse las funciones `amICompatibleWithOther` y `isOtherCompatibleWithMe`.

```
${DEPSA}/PreGenerator/src/pregenerator/rulespossibletype
```

Código 37 - Directorio donde agregar la compatibilidad con las nuevas reglas

Por último se debe agregar en todos los generadores específicos para que puedan generarla. Para estos casos es necesario referirse a la documentación de cada generador en particular.

Para el caso de ModSecurityGenerator se debe agregar en el directorio que se indica en el Código 38 una clase que extienda `ModSecuritySecurityPolicy`, cuyo nombre debe coincidir con el nombre del `PreGeneratedSecurityPolicy` que representa, reemplazando el PGDT por MSDT. Por ejemplo si el pregenerador retorna una clase `PGDTAllow`, el generador debe tener una clase `MSDTAllow` que lo implemente. Al implementar la función `makeMe` se valida que se reciba una instancia del tipo correcto de PreGeneratedSecurityPolicy.

```
${DEPSA}/ModSecurityGenerator/src/modsecuritygenerator/datatypes/policies
```

Código 38 - Directorio donde agregar nuevas reglas a nivel ModSecurityGenerator

El siguiente capítulo expone las conclusiones del trabajo realizado.

[Página dejada en blanco intencionalmente.]

7 Conclusiones

Se concluye que los objetivos del proyecto fueron alcanzados al presentar una solución capaz de realizar el *enforcement* de una política de seguridad definida sobre el modelo de la aplicación a proteger. La política de seguridad se aplica sin necesidad de modificar la aplicación debido a que el *framework* genera un conjunto de reglas que configuran una herramienta de *virtual patching* para mitigar las vulnerabilidades detectadas. Además se definió un procedimiento iterativo e incremental que permite a un especialista en seguridad junto con un especialista en el modelo de la aplicación definir o modificar la política de seguridad.

En el proyecto se abordan las áreas de políticas de seguridad, relaciones entre dominios, vulnerabilidades de seguridad, herramientas de *virtual patching*, extensiones de UML y uso de interfaces con los usuarios. Cada una de estas áreas abarcan una vasta cantidad de contenido, lo que permitió que durante el proyecto se dedicara tiempo en aspectos que no se pudieron concretar, dados los tiempos disponibles para cumplir con las metas del proyecto, por lo que se decidió limitar el alcance en cada una de las áreas.

Fue creado el lenguaje SPLang para definir las políticas de seguridad sobre el modelo de la aplicación. El lenguaje es extensible, por ejemplo donde se agrupan distintos tipos de reglas, como las específicas que puede definir un especialista en seguridad, las reglas que sirven para que la herramienta de *virtual patching* aplique una solución predefinida para ciertas vulnerabilidades conocidas como el caso de OWASP CRS, y la definición base del control de acceso basado en roles (RBAC). Se estudiaron herramientas empleadas por los desarrolladores para representar los modelos, y si bien queda como trabajo a futuro incorporar el lenguaje en dichas herramientas, se consideró como se identifican los recursos.

MapLang cumple con la necesidad de identificar la relación de los recursos en el modelo de la aplicación y los distintos dominios donde las herramientas de *virtual patching* aplican la política de seguridad. La definición de los recursos del modelo de la aplicación en MapLang coincide con la de SPLang para que sea más sencillo determinar la correspondencia entre ellos, y quitarle el trabajo de unificarlos al módulo `PreGenerator`. La definición de los recursos de los dominios de las herramientas de *virtual patching* fue implementado con el fin de que no sólo sirva para representar los recursos de un WAF, sino para otros dominios como el de base de datos.

La solución presentada posee la suficiente abstracción como para cambiar la herramienta de *virtual patching*, incorporar nuevas y generar archivos de configuración para todas ellas partiendo de la misma política de seguridad. Esto agregó dificultad en la definición de todos sus componentes por ejemplo porque se debió considerar los dominios utilizados por las herramientas de *virtual patching* para definir el mapeo de dominios, en lugar de ser solo el dominio web de ModSecurity.

La diferencia de enfoques entre el proyecto y el uso del CRS de OWASP con ModSecurity presentó otra dificultad. Se observó que colocar reglas sobre el modelo de la aplicación, definiendo explícitamente qué reglas se desea aplicar, no coincide con el enfoque tomado por ModSecurity el cual se basa en colocar excepciones sobre las reglas generales, definidas en el CRS de OWASP. Esto tuvo impacto en el diseño del módulo `ModSecurityGenerator` que debió contemplarlos.

La herramienta ANTLR simplificó la tarea de reconocer los lenguajes y genera automáticamente varias opciones de recorrerlos. Esto permite crear una estructura de datos que cumpla con la interfaz definida para desasociar esta implementación con el resto de la aplicación.

La utilización de funcionalidades avanzadas de Java, simplificó la definición de un mecanismo para incorporar nuevos tipos de reglas y generadores de otras herramientas de *virtual patching*, sin

necesidad de modificar lo implementado. En particular, la técnica *reflection*, permite instanciar clases de forma dinámica a través del nombre de la misma. Esta funcionalidad, por ejemplo hizo posible que los módulos invoquen representaciones de los tipos de reglas, independientemente del momento en que hayan sido agregados. Esto aumentó el nivel de abstracción de la solución al permitir que nuevos tipos sean soportados por el sistema, únicamente creando una representación de la misma dentro de dicho módulo.

Con el caso de estudio fue posible validar la solución implementada por DEPSA, al definir una política de seguridad a partir del modelo de LeagueManager utilizando SPLang y las relaciones entre los elementos del modelo y los de ModSecurity, empleando MapLang. A partir de estas definiciones DEPSA generó una configuración para ModSecurity que mitigó las vulnerabilidades detectadas en LeagueManager. Además se pudo comprobar la interacción entre ModSecurity y el enfoque tomado por el proyecto de definir las reglas que se deben aplicar a cada recurso.

El equipo de trabajo adquirió conocimientos en todas las áreas abordadas por el proyecto, entre ellas se destacan vulnerabilidades conocidas que son frecuentes en el desarrollo del software, el uso del *firewall* ModSecurity para agregar restricciones de seguridad en una aplicación sin necesidad de modificar el código fuente y el uso de la herramienta ANTLR para procesar lenguajes.

A continuación se enumeran los trabajos a futuro que extienden el alcance actual del proyecto, profundizando en áreas reconocidas durante el desarrollo del trabajo.

7.1 Trabajos a futuro

Durante la realización del proyecto se estudiaron perfiles de UML que aportan seguridad al modelo de la aplicación, pero quedó fuera del alcance del mismo la vinculación con los diagramas de UML y el lenguaje SPLang. Se propone la creación o extensión de un perfil de UML para agregar la política de seguridad al diagrama de clases y obtener el archivo .splang. Definir el metalenguaje siguiendo los estándares de UML para generar una interfaz gráfica de usuario o modificar las existentes para que incorporen la definición de SPLang y así simplificar la definición de la política de seguridad, como buscan los trabajos de UMLSec y demás perfiles estudiados.

Así como se observaron las ventajas de simplificar la tarea de definir la política de seguridad, lo mismo puede considerarse para la definición de los mapeos, por lo que sería de utilidad asistir a los usuarios a generar dichos mapeos mediante una interfaz de usuario amigable. Por ejemplo que recuerde los recursos de cada dominio e incluso sugiera posibles mapeos. Investigar herramientas que contribuyan para realizar esta tarea de forma semiautomática o automática, por ejemplo configurar la herramienta en modo detección y luego de un tiempo de haber recabado información acerca de los recursos de un dominio, sugerir relaciones.

Para validar la solución se implementaron sólo algunas reglas del CRS de OWASP, y se debería extender para poder aplicarlas en su totalidad. Existen más grupos de reglas y algunas no aplican a los atributos u operaciones, sino que se deben aplicar al sistema como es el caso de la detección de violación del protocolo HTTP o de posibles negaciones de servicios (DoS). Además de incorporar las nuevas reglas al lenguaje, se deben implementar los controles para que estas reglas no sean aplicadas a atributos u operaciones, porque el lenguaje no lo restringe ya que es una particularidad de ModSecurity.

Las reglas que conforman la política de seguridad podrían incluir puntajes y de esta manera poder ejecutarlas en el modo detección de anomalías con puntajes de ModSecurity. Se puede definir una categoría para acumular puntaje en distintas reglas mediante la acción de las reglas de ModSecurity [17], por ejemplo para recursos que permiten código HTML, se define el contador “reglas_html”

como `setvar:tx.reglas_html` y si se cumple lo que se desea especificar, se incrementa su valor con `setvar:tx.reglas_html=+5`. Se debe brindar la posibilidad de expresar en la política de seguridad, el puntaje que indica que se debe tomar una acción disruptiva, así como configurar los puntajes que se acumulan de acuerdo al nivel de severidad definida para la regla, en lugar de escribir un número fijo. ModSecurity considera la severidad 0 como la mayor y 6 el nivel de menor severidad y le asigna valores de 5 y 0 respectivamente.

En los lenguajes MapLang y SPLang se asume que las expresiones regulares cumplen con PCRE (*Perl Compatible Regular Expressions*). Se deberían validar que se cumpla y detectar incompatibilidades entre reglas que aplican a un recurso evaluando las expresiones regulares que se definen en las mismas.

Los *firewalls* son configurados tradicionalmente con dos tipos de políticas, la restrictiva y la permisiva. Se considera que puede ser útil agregar esta configuración para que el usuario defina de qué manera debe funcionar el *firewall* con su política.

Si bien se definieron reglas en el lenguaje SPLang que consideran el control de acceso a los recursos de los distintos usuarios del sistema, se debe implementar un módulo para su gestión. En este caso se debe proveer una base de datos para gestionar los usuarios y los roles así como una API para poder validar si un usuario posee un rol que lo habilite a usar un recurso. Para el caso de ModSecurity es posible realizar estos controles mediante reglas *`SecRuleScript`* para ejecutar *scripts* de Lua, que incluso se pueden conectar a bases de datos.

El framework puede soportar varios generadores específicos de reglas, por lo que interesa desarrollar módulos para otros *firewalls* o herramientas que sirvan para realizar *virtual patching* de una aplicación a partir de la política de seguridad definida. Basándose en las estructuras de datos generadas por los módulos encargados de reconocer los lenguajes definidos, se podría generar reglas particulares para dichas herramientas y notificar al módulo generador que se desea usar el nuevo generador específico.

Como fue detallado, al instalar el CRS de OWASP en ModSecurity el enfoque consiste en aplicar las reglas a todo el sistema y luego especificar las excepciones que se deseen para evitar que dichas reglas apliquen a un recurso en particular. Se propone la creación de un mecanismo que permita definir reglas del CRS para algún recurso particular y que no sea basándose en excepciones. Una solución posible consiste en que DEPSA contenga las reglas del CRS y las defina para cada recurso, lo que requiere de un mecanismo para actualizarse cuando las reglas del CRS sean modificadas. Otra solución sería que el CRS de OWASP provea el mecanismo para que las reglas se definan para los recursos en particular en lugar de para todo el sistema. Por último se plantea que ModSecurity permita agregar recursos a reglas definidas previamente. Esta solución además simplificaría la implementación de DEPSA para los `DataTypeValidation`, permitiendo definir una única regla con la definición del datatype y agregarle los recursos posteriormente.

Desarrollar una interfaz de usuario amigable que integre todo el proceso desde la generación de la política de seguridad a partir del modelo de la aplicación, permitiendo importar los diagramas en UML con el modelo de la aplicación y agregarle las reglas a los recursos, asistencia para definir mapeos entre los diferentes dominios, configuraciones de base para los diferentes generadores de reglas seleccionados, como el modo de ejecución de ModSecurity y permitir seleccionar cuales generadores emplear.

[Página dejada en blanco intencionalmente.]

Referencias

[1] “Measuring The Information Society Report.” - *ITU*. Accessed September 18, 2015. http://www.itu.int/en/itu-d/statistics/pages/publications/mis2014.aspx.

[2] Sommerville, Ian. *Software Engineering*. Boston: Pearson, 2011.

[3] Devanbu, Premkumar T., and Stuart Stubblebine. “Software Engineering for Security.” *Proceedings Of the Conference on The Future of Software Engineering - ICSE '00*, 2000. doi:10.1145/336512.336559.

[4] “Automated Quality Characteristic Measures.” - *CISQ Consortium For IT Software Quality*. Accessed September 18, 2015. http://it-cisq.org/standards/automated-quality-characteristic-measures/.

[5] “Common Vulnerabilities And Exposures - Terminology.” - *CVE*. Accessed September 18, 2015. https://cve.mitre.org/about/terminology.html.

[6] “Internet Security Threat Report 2015” - *Symantec*. Accessed September 18, 2015. https://know.elq.symantec.com/lp=1542.

[7] “OWASP Virtual Patching Survey Results.” *Trustwave Holdings, Inc*. Accessed September 18, 2015. https://www.trustwave.com/resources/spiderlabs-blog/owasp-virtual-patching-survey-results/.

[8] “Virtual Patching Best Practices.” - *OWASP*. Accessed September 18, 2015.https://www.owasp.org/index.php/Virtual_Patching_Best_Practices.

[9] Barnett, Ryan. “WAF Virtual Patching Challenge: Securing WebGoat with ModSecurity”

[10] “OWASP Top Ten Project.” - *OWASP*. Accessed September 18, 2015. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[11] “2011 CWE/SANS Top 25 Most Dangerous Software Errors.” - *CWE/SANS*. Accessed September 18, 2015. http://cwe.mitre.org/top25/.

[12] “Virtual Patching Cheat Sheet.” - *OWASP*. Accessed September 18, 2015. https://www.owasp.org/index.php/Virtual_Patching_Cheat_Sheet.

[13] “Intrusion Detection FAQ: What Is the Difference between an IPS and a Web Application Firewall?” - *SANS*. Accessed September 18, 2015. https://www.sans.org/security-resources/idfaq/ips-web-app-firewall.php.

[14] “ModSecurity: Open Source Web Application Firewall.” - *ModSecurity: Open Source Web Application Firewall*. Accessed September 18, 2015. http://www.modsecurity.org/.

[15] “Reference Manual.” - *SpiderLabs/ModSecurity*. Accessed September 18, 2015. https://github.com/spiderlabs/modsecurity/wiki/reference-manual.

[16] “OWASP ModSecurity Core Rule Set Project.” - *OWASP*. Accessed September 18, 2015. https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project.

[17] “Advanced Topic Of the Week: Traditional vs. Anomaly Scoring Detection Modes.” - *ModSecurity Blog*. Accessed September 18, 2015. http://blog.modsecurity.org/2010/11/advanced-topic-of-the-week-traditional-vs-anomaly-scoring-detection-modes.html.

[18] Scott, David, and Richard Sharp. "SPECTRE: A Tool for Inferring, Specifying and Enforcing Web-Security Policies".

[19] Almorsy, Mohamed, John Grundy, and Amani S. Ibrahim. “MDSE@R: Model-Driven Security Engineering At Runtime.” *Cyberspace Safety And Security Lecture Notes in Computer Science*, 2012, 279–95. doi:10.1007/978-3-642-35362-8_22.

[20] *Trusted Computer System Evaluation Criteria*. Ft. Meade, MD: Dept. of Defense, Computer Security Center, 1985.

[21] “Unified Modeling Language™ (UML®) Resource Page.” - *Unified Modeling Language (UML)*. Accessed September 18, 2015. http://www.uml.org/.

[22] Hayati, Pedram, Nastaran Jafari, S. Mohammad Rezaei, Saeed Sarenche, and Vidyasagar Potdar. “Modeling Input Validation In UML.” *19th Australian Conference On Software Engineering (Aswec 2008)*, 2008. doi:10.1109/aswec.2008.4483260.

[23] “About OMG®.” - *The Object Management Group*. Accessed September 18, 2015. http://www.omg.org/gettingstarted/gettingstartedindex.htm.

[24] “Introduction To OMG's Unified Modeling Language® (UML®).” - *OMG UML*. Accessed September 18, 2015. http://www.omg.org/gettingstarted/what_is_uml.htm.

[25] “OMG Unified Modeling Language (UML) Vendor Directory.” - *OMG UML*. Accessed September 18, 2015. http://uml-directory.omg.org/.

[26] “UML Infrastructure.” - *OMG UML*. Accessed September 18, 2015. http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/.

[27] Jürjens, Jan. “Towards Development Of Secure Systems Using UMLsec.” *Fundamental Approaches To Software Engineering Lecture Notes in Computer Science*, 2001, 187–200. doi:10.1007/3-540-45314-8_14.

[28] Lodderstedt, Torsten, David Basin, and Jürgen Doser. “SecureUML: A UML-Based Modeling Language For Model-Driven Security.” *≪UML≫ 2002 — The Unified Modeling Language Lecture Notes In Computer Science*, 2002, 426–41. doi:10.1007/3-540-45800-x_33.

[29] Peterson, M.j., J.b. Bowles, and C.m. Eastman. “UMLpac: An Approach For Integrating Security into UML Class Design.” *Proceedings Of the IEEE SoutheastCon 2006*, 2006. doi:10.1109/second.2006.1629362.

[30] Sandhu, Ravi, David Ferraiolo, and Richard Kuhn. “The NIST Model for Role-Based Access Control.” *Proceedings Of the Fifth ACM Workshop on Role-Based Access Control - RBAC '00*, 2000. doi:10.1145/344287.344301.

[31] Scott, David, and Richard Sharp. “Abstracting Application-Level Web Security.” *Proceedings Of the Eleventh International Conference on World Wide Web - WWW '02*, 2002. doi:10.1145/511446.511498.

[32] Milner, R. *The Definition of Standard ML: Revised*. Cambridge, MA: MIT Press, 1997.

[33] “Extensible Markup Language (XML).” - *W3C*. Accessed September 18, 2015. http://www.w3.org/xml/.

[34] “RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format.” - *Internet Engineering Task Force (IETF)*. Accessed September 18, 2015. https://tools.ietf.org/html/rfc7159.

[35] “Web Application Description Language.” - *W3C*. Accessed September 18, 2015. http://www.w3.org/submission/wadl/.

[36] “Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.” - *W3C*. Accessed September 18, 2015. http://www.w3.org/tr/wsdl20/.

[37] “About W3C.” - *W3C*. Accessed September 18, 2015. http://www.w3.org/consortium/.

[38] “XML Tutorial.” - *w3schools*. Accessed September 18, 2015. http://www.w3schools.com/xml/default.asp.

[39] “W3C XML Schema Definition Language (XSD)” - *W3C*. Accessed September 18, 2015. http://www.w3.org/TR/xmlschema11-1/.

[40] “XSL Languages.” - *w3schools*. Accessed September 18, 2015. http://www.w3schools.com/xsl/xsl_languages.asp.

[41] “The Extensible Stylesheet Language Family (XSL).” - *W3C*. Accessed September 18, 2015. http://www.w3.org/style/xsl/.

[42] “XSL Transformations (XSLT) Version 1.0.” - *W3C*. Accessed September 18, 2015. http://www.w3.org/tr/xslt.

[43] “XML Path Language (XPath)Version 1.0.” - *W3C*. Accessed September 18, 2015. http://www.w3.org/tr/xpath/.

[44] “XQuery 1.0: An XML Query Language (Second Edition).” - *W3C*. Accessed September 18, 2015. http://www.w3.org/tr/xquery/.

[45] “JAXP Reference Implementation — Project Kenai.” JAXP Reference Implementation — Project Kenai. Accessed September 18, 2015. https://jaxp.java.net/.

[46] “Introducing JSON.” - *JSON*. Accessed September 18, 2015. http://json.org/.

[47] “What Are RESTful Web Services? - The Java EE 6 Tutorial.” - *Oracle*. Accessed September 18, 2015. http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html.

[48] “Team Comment On the ‘Web Application Description Language’ Submission.” - *W3C*. Accessed September 18, 2015. http://www.w3.org/submission/2009/03/Comment.

[49] “Web Services Description Language (WSDL) Version 1.2: Bindings.” - *W3C*. Accessed September 18, 2015. http://www.w3.org/tr/2002/wd-wsdl12-bindings-20020709/.

[50] “Learn REST: A Tutorial. Documenting REST Services: WSDL And WADL”. Accessed September 18, 2015. http://rest.elkstein.org/2008/02/documenting-rest-services-wsdl-and-wadl.html.

[51] "On Mapping between UML and Entity-Relationship Model". Accessed September 18, 2015. https://static.aminer.org/pdf/PDF/000/635/065/on_mapping_between_uml_and_entity_relationship_model.pdf.

[52] "Model Mapping Approach Based on Ontology Semantics". Accessed September 18, 2015. http://ojs.academypublisher.com/index.php/jnw/article/viewFile/jnw080919671974/7712.

[53] “ISO/IEC 14977” - *ISO/IEC*. Accessed September 18, 2015. http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf.

[54] "Extensible Markup Language (XML) 1.0 (Fifth Edition)." - *W3C*. Accessed September 18,

2015. http://www.w3.org/tr/rec-xml/#sec-notation.

[55] "PCRE - Perl Compatible Regular Expressions." - *PCRE - Perl Compatible Regular Expressions*. Accessed September 18, 2015. http://www.pcre.org/.

[56] Parr, Terence. *The Definitive ANTLR 4 Reference*. n.d., 2012.

[57] “(The Java™ Tutorials) Trail: The Reflection API.” - *Oracle*. Accessed September 18, 2015. https://docs.oracle.com/javase/tutorial/reflect/.

Glosario

Analizador lexicográfico

Primera fase de un compilador. Recibe como entrada el código fuente e identifica los Tokens que lo conforman en orden.

Analizador sintáctico

Fase de un compilador posterior al análisis léxico. Toma los tokens generados por el anterior y crea estructuras para que puedan ser analizadas.

Código Fuente

Conjunto de líneas de texto escritas en un lenguaje de programación que describen el funcionamiento del programa. Puede ser interpretado directamente por la máquina o requerir ser pasado por un compilador o algún otro traductor que lo lleve a otro lenguaje, que la misma pueda comprender.

Compilador

Es un programa que traduce el código fuente de una aplicación a otro lenguaje de programación.

Componente Léxico

Ver Token

Disruptivo

Palabra derivada de la expresión inglesa “disruptive”. Se utiliza para nombrar algo que produce una ruptura brusca. En el contexto de las herramientas de virtual patching, al hablar de reglas disruptivas, se hace referencia a reglas de firewalls que cortan la ejecución de las validaciones de sus reglas, ya sea porque toma la decisión de dejar pasar la información o bloquearla.

Enforcement

Palabra de la lengua inglesa. Como sustantivo implica aplicar, ejecutar o vigilar. Hace referencia a exigir el cumplimento de leyes establecidas. En el marco de este proyecto implica asegurarse que las políticas definidas se cumplan.

Firewall

Es una pieza de software o hardware que analiza la información que entra y sale a un sistema. Acto seguido puede bloquear, loguear o tomar acciones con esta información en función de su configuración.

Framework

Palabra de la lengua inglesa, significando marco de trabajo. Se define como un conjunto de conceptos, prácticas y criterios estandarizados para resolver un problema particular. En informática es una estructura de soporte en la cual otros proyectos de software pueden apoyarse.

Gramática

Ciencia que estudia los componentes de un lenguaje y las combinaciones entre estos.

Lenguaje

Sistema utilizado para comunicarse de manera estructurada. Poseen un contexto de uso, componentes y reglas que determinan como se deben combinar los mismos.

Lexer

Programa que genera analizadores lexicográficos.

Mapear

Establecer una relación entre 2 o más elementos de dominiós distintos.

Modelos en alto nivel

Esquema conceptual que modela la realidad de un problema en alto nivel sin incluir particularidades de la implementación.

Parser

Ver Analizador sintáctico.

Patrón de diseño

Son plantillas de soluciones probadas y documentadas a problemas comunes. Cuando se definen se le asigna un nombre, descripción del problema que resuelven, descripción abstracta de la solución y las consecuencias y ganancias de aplicar el mismo.

Reingeniería de sistemas

Modificación de un producto de software o componentes del mismo, utilizando ingeniería inversa en la etapa de análisis para comprender cómo se comporta el mismo y luego ingeniería directa en la reconstrucción del mismo. Se busca con la misma mejorar la mantenibilidad, reutilización o comprensión de la misma.

Robustez

Atributo de calidad que hace referencia a la resistencia ante cambios en el entorno.

Sanitizar

Remover elementos que puedan generar problemas a un nivel seguro, no necesariamente removiendo la totalidad de los mismos. En seguridad informática hace referencia a remover información considerada sensible, confidencial o peligrosa, ya sea eliminando la misma o reemplazandola.

Sistemas de información

Conjunto de componentes que interaccionan entre sí para satisfacer las necesidades de información de una organización.

Sociedad de la información

Sociedad en la cual las tecnologías que dan acceso y permiten modificar la información cumplen un rol esencial en sus actividades.

Stakeholders

Palabra de la lengua inglesa. personas o entorno que se puede ver afectado positiva o negativamente por la culminación del proyecto.

Tecnologías de la información y comunicación

Conjunto de tecnologías creadas para gestionar información y transmitirla. Incluyen desde tecnologías para almacenar, procesar y recuperar la información, hasta las necesarias para enviar y recibir información entre dos o más componentes.

Token

También llamado componente léxico. Cadena de caracteres que representa la unidad básica a partir de la cual se desarrolla la traducción de un programa.

[Página dejada en blanco intencionalmente.]

Notas

Anexos

Proyecto de grado

Definición y Enforcement de Políticas de Seguridad sobre Aplicaciones

(DEPSA)

Tutores

Felipe Zipitría – Rodrigo Martínez

Rodrigo de la Fuente

Luis González

Juan Pírez

[Página dejada en blanco intencionalmente.]

Índice

1 Anexo I - SPLang....4
2 Anexo II – MapLang....8
3 Anexo III - Contratos de operaciones....10
4 Anexo IV - Diagramas de comunicación....15
4.1 Generador::generate....15
4.2 PreGeneratorMain::execute....16
4.3 PreGeneratorMain::preGenerateDataTypeValidationList()....17
4.4 ModSecurityGenerator::generate....19
4.5 Imapping::map....23
4.6 IsecurityPolicies::securityPolicies....24

1 Anexo I - SPLang

```
/**
 * Define a grammar called SPLang
*/
grammar SPLang ;
init : 'SPLang VERSION:1.0' (dataTypeValidation)* (ruleDefinition)+ ;


/** COMMON TYPES **/
// fragment annotation indicates ANTLR that this rules are always part of
another rules
fragment
LETTER : [a-zA-ZñÑ] ;


fragment
NUMBER : [0-9] ;


// expression for numbers
NUMBERS : NUMBER+ ;


// expression for a word with numbers, letters and some special characters _
- . /
NAMES : (NUMBER|LETTER|[_-./])+ ;


// expression for a string
STRING_TYPE : '"' ('\\"'|.)*? '"' ;


/** Constraints that could be applied to different resources and share this
policy  **/
dataTypeValidation : 'DATA_TYPE_VALIDATION' 'ID' dataTypeValidationId 'NAME'
                     name 'IO_VALIDATION_CONSTRAINT' ioValidationConstraint ;


dataTypeValidationId : NUMBERS ;


name : NAMES ;


/** Rules (the world rule conflicts with antlr4, so we add 'Definition' **/
ruleDefinition : 'RULE' 'RESOURCE' resource (rulePermissionRole)+ ;


// with the ampersand annotation ANTLR makes a visitor for each subrule with //
this name (resourceSYSTEM, etc)
resource : 'SYSTEM'                                     #resourceSYSTEM
         | 'PACKAGE_NAME' packageName 'CLASS_NAME' className 'ATTRIBUTE'
           attributeName                                #resourceATTRIBUTE
```

```
        | 'PACKAGE_NAME' packageName 'CLASS_NAME' className 'OPERATION'
          operationName '(' (parameters)? ')'      #resourceOPERATION
        ;

packageName : STRING_TYPE ;
className : NAMES ;
attributeName : NAMES ;
operationName : NAMES ;
parameters : parameter (',' parameter)*;
parameter : NAMES ;

rulePermissionRole : 'CONSTRAINT' constraint
                   | permissionRole
                   ;

constraint : 'PRIORITY' priority 'COMMUNICATION_DIRECTION'
             communicationDirection constraintDesc 'IMPACT' impact 'COMMENT'
             comment ;

priority : NUMBERS ;
comment : STRING_TYPE ;
impact : 'NONE'    #impactNONE
       | 'LOW'     #impactLOW
       | 'MEDIUM'    #impactMEDIUM
       | 'HIGH'    #impactHIGH
       | 'SEVERE'    #impactSEVERE
       ;
communicationDirection  : 'IN'       #communicationDirectionIN
                        | 'OUT'      #communicationDirectionOUT
                        | 'BOTH'    #communicationDirectionBOTH
                        ;

constraintDesc : defaultConstraint
               | specificConstraint
               ; // add more constraints | ...

defaultConstraint : xssConstraint
                  | csrfConstraint
                  | injectionConstraint
                  ; // add more default constraints | ...

xssConstraint : 'XSS' ;
csrfConstraint : 'CSRF' ;
```

```
injectionConstraint : injectionType 'INJECTION' ;
injectionType : 'SQL'                #injectionSQL
              | 'HTML'          #injectionHTML
              | 'JAVASCRIPT'    #injectionJAVASCRIPT
              ; // add | ...

specificConstraint : 'IO_VALIDATION' ioValidationConstraint ;

ioValidationConstraint : blacklist   #ioValidationConstraintBLACKLIST
                       | sanitization #ioValidationConstraintSANITIZATION
                       | whitelist #ioValidationConstraintWHITELIST
                       | 'DATA_TYPE_VALIDATION_ID' dataTypeValidationId
                         #ioValidationConstraintDATATYPEVALIDATIONID
                       ;
// blacklist definition
blacklist : 'BLACKLIST' (STRING_TYPE)+ ;

// sanitization definition
sanitization : 'SANITIZATION'  (sanitizationValues)* ;

sanitizationValues : STRING_TYPE ',' STRING_TYPE ;

// whitelist definition
whitelist : 'WHITELIST' 'FORMAT' formatAttribute 'TYPE' typeAttribute 'DOMAIN'
            domainAttribute 'LENGTH' length ;

formatAttribute : STRING_TYPE ;
typeAttribute : STRING_TYPE ;
domainAttribute : STRING_TYPE ;
length : NUMBERS ;

// RBAC permission model
permissionRole : 'PERMISSION' permission (role)+ ;

permission : 'ALLOW'     #permissionALLOW
           | 'DENY'      #permissionDENY
           ; // add permissions | ...

role : 'ROLE' roleName ;

roleName : STRING_TYPE ;
```

```
/** skip spaces, tabs, newlines **/
WS : [ \t\r\n]+ -> skip ;
```

2 Anexo II – MapLang

```
/**
 * Define a grammar called MapLang
*/
grammar MapLang ;
init : 'MapLang VERSION:1.0' 'MODEL' modelId 'DOMAIN_LIST' (domain)+ ;

fragment
LETTER : [a-zA-ZñÑ] ;

fragment
NUMBER : [0-9] ;

NAMES : (NUMBER|LETTER|[_-./])+ ;

STRING_TYPE : '"' ('\\"'|.)*? '"' ;

modelId : STRING_TYPE ;

domain : 'DOMAIN' 'DOMAIN_ID' domainId ('SYSTEM' system)? (resourceMapping)+ ;

domainId : STRING_TYPE ;

system : STRING_TYPE ;

resourceMapping : 'RESOURCE' resource 'RELATED_RESOURCE_LIST'
 (relatedResource)+ ;

resource : 'PACKAGE_NAME' packageName 'CLASS_NAME' className 'ATTRIBUTE'
attributeName #resourceATTRIBUTE
         | 'PACKAGE_NAME' packageName 'CLASS_NAME' className 'OPERATION'
operationName '(' (parameters)? ')' #resourceOPERATION
         ;

packageName : STRING_TYPE ;
className : NAMES ;
attributeName : NAMES ;
operationName : NAMES ;
parameters : parameter (',' parameter)*;
parameter : NAMES ;
```

```
relatedResource : 'CONTAINER_ID' containerId 'RESOURCE_ID' relatedResourceId ;

containerId : STRING_TYPE ;

relatedResourceId : STRING_TYPE ;

/** skip spaces, tabs, newlines **/
WS : [ \t\r\n]+ -> skip ;
```

3 Anexo III - Contratos de operaciones

Generator::generate

Operación
generate(String policiesPath, String mappingsPath, List<SpecificGeneratorDT> generators)
Descripción
Invoca al pregenerador solicitandole la lista de políticas explotadas para los archivos de políticas y mapeos recibidos por parámetros. Una vez obtenidas estas políticas invoca a los generadores que se le haya indicado en la variable generators para que generen sus reglas para las herramientas correspondientes. Devuelve una lista con todas las advertencias que se hayan obtenido en el proceso.
Pre-Condiciones
• policePath es una ruta válida donde se encuentra un archivo .splang • mappingsPath es una ruta válida donde se encuentra un archivo .maplang • generators posee una lista de SpecificGenerators válidos.
Post-Condiciones
• Cada generador invocado ha generado su archivo de reglas • Retorna una lista con las advertencias generadas en todo el proceso.
Parámetros de entrada
• policiesPath: Ruta al archivo que contiene las políticas de seguridad • mappingsPath: Ruta al archivo que contiene los mapeos del sistema. • generators: representación de los generadores a utilizar.
Retorno
List<SpecificGeneratorWarning>: Lista que contiene todoas las advertencias generadas durante el proceso.
Excepciones
• GeneratorNotFoundException • UnmappedResourceException • RuleNotFoundException • UserMessageException • IncompatibleTypeException • SPLangSyntaxErrorException • MapLangSyntaxErrorException • IncompatibleRulesException • SPLangDuplicatedIdException • SPLangInvalidIdReferenceException • SPLangFileNotFoundException • SPLangIOException • MapLangFileNotFoundException

• MapLangIOException • InvalidDataTypeException

PreGeneratorMain::execute

Operación
execute(String policiesPath, String mappingsPath)
Descripción
Invoca el módulo de SecurityPolicies y el de Mappings para obtener las estructuras de datos que representan la política de seguridad y los mapeos, a partir de las rutas recibidas por parámetro. Si alguno de estos módulos ha detectado la existencia de un error, interrumpe su ejecución e informa del problema. En caso de que todo sea correcto se procesan las estructuras generadas por dichos módulos. Para esto se recorren las reglas de la política de seguridad definidas para un elemento del modelo de la aplicación y se asigna cada regla a todos los recursos relacionados según lo indique Mappings. En caso de no existir un mapeo para dicho elemento, o que se aplique una regla al sistema y la representación del mismo no fue definido para algún dominio, se considera un error bloqueante y se informa del problema. Se valida la definición de los DataTypeValidation para que sean del tipo correcto y que no haya una definición recursiva. Se crea la estructura con la relación entre los recursos y las reglas que se deben aplicar para cada dominio. Se verifica según el Error: Reference source not found entre las reglas asignadas a los recursos y al sistema, y genera advertencias en caso de que no se cumplan.
Pre-Condiciones
• policePath es una ruta válida donde se encuentra un archivo .splang • mappingsPath es una ruta válida donde se encuentra un archivo .maplang
Post-Condiciones
• Devuelve una estructura con las políticas explotadas en los distintos dominios mapeados.
Parámetros de entrada
• policiesPath: Ruta al archivo que contiene las políticas de seguridad • mappingsPath: Ruta al archivo que contiene los mapeos del sistema.
Retorno
PreGeneratedPolicies: Estructura que posee las políticas explotadas en los distintos dominios mapeados y la lista de advertencias con las incompatibilidades entre reglas.
Excepciones
• SPLangDuplicatedIdException • SPLangInvalidIdReferenceException • SPLangSyntaxErrorException • SPLangFileNotFoundException • SPLangIOException • MapLangSyntaxErrorException

- MapLangFileNotFoundException
- MapLangIOException
- RuleNotFoundException
- IncompatibleRulesException
- UnmappedResourceException
- InvalidDataTypeException

ModSecurityGenerator::generate

Operación
GeneratedPolicies generate(PreGeneratedPolicies preGeneratedPolicies, HashMap<String, Object> parameters)
Descripción
Genera las reglas para las políticas indicadas en preGeneratedPolicies obteniendo únicamente el/los dominio/s correspondiente/s a su herramienta. Retorna las reglas generadas junto con las advertencias generadas en el proceso.
Pre-Condiciones
• preGeneratedPolicies es una estructura válida generada por el pregenerador.
Post-Condiciones
• Reglas generadas para la herramienta correspondiente generadas y retornadas por parámetro.
Parámetros de entrada
• preGeneratedPolicies: estructura que contiene todas las políticas que deben aplicarse a los recursos del sistema. • parameters: Parámetros auxiliares que pueden ser de utilidad para generadores particulares.
Retorno
GeneratedPolicies: estructura que contiene las reglas generadas y las advertencias obtenidas en el proceso.
Excepciones
• preGeneratedPolicies

Imapping::map

Operación
map(String path)
Descripción
Recibe la ruta al archivo que contiene el mapeo descrito en MapLang y valida que el archivo exista, realiza el reconocimiento léxico y analiza que la sintaxis sea correcta. En caso de encontrarse errores se generan mensajes con información sobre la ubicación del problema. Genera una estructura de datos que corresponda con el tipo Mappings representado los mapeos contenidos en el archivo pero sin las peculiaridades de la herramienta que realiza el

reconocimiento del lenguaje.
Pre-Condiciones
-
Post-Condiciones
-
Parámetros de entrada
• path: ruta absoluta a un archivo .maplang
Retorno
Estructura de datos Mapping con el identificador del modelo, y la colección de posibles codominios donde se puede mapear el recurso. Cada elemento contiene la descripción del sistema y la correspondencia entre cada recurso del modelo de la aplicación con los del codominio especificado.
Excepciones
• MapLangSyntaxErrorException: se detecta un error de sintaxis en el archivo. Contiene la línea y la posición dentro de la línea para identificar dónde se encuentra el error así como el mensaje de error generado por ANTLR en este caso que sirven para corregir el mismo. • MapLangFileNotFoundException: no existe el archivo en “path”. • MapLangIOException: error al leer el archivo ubicado en “path”.

ISecurityPolicies::securityPolicies

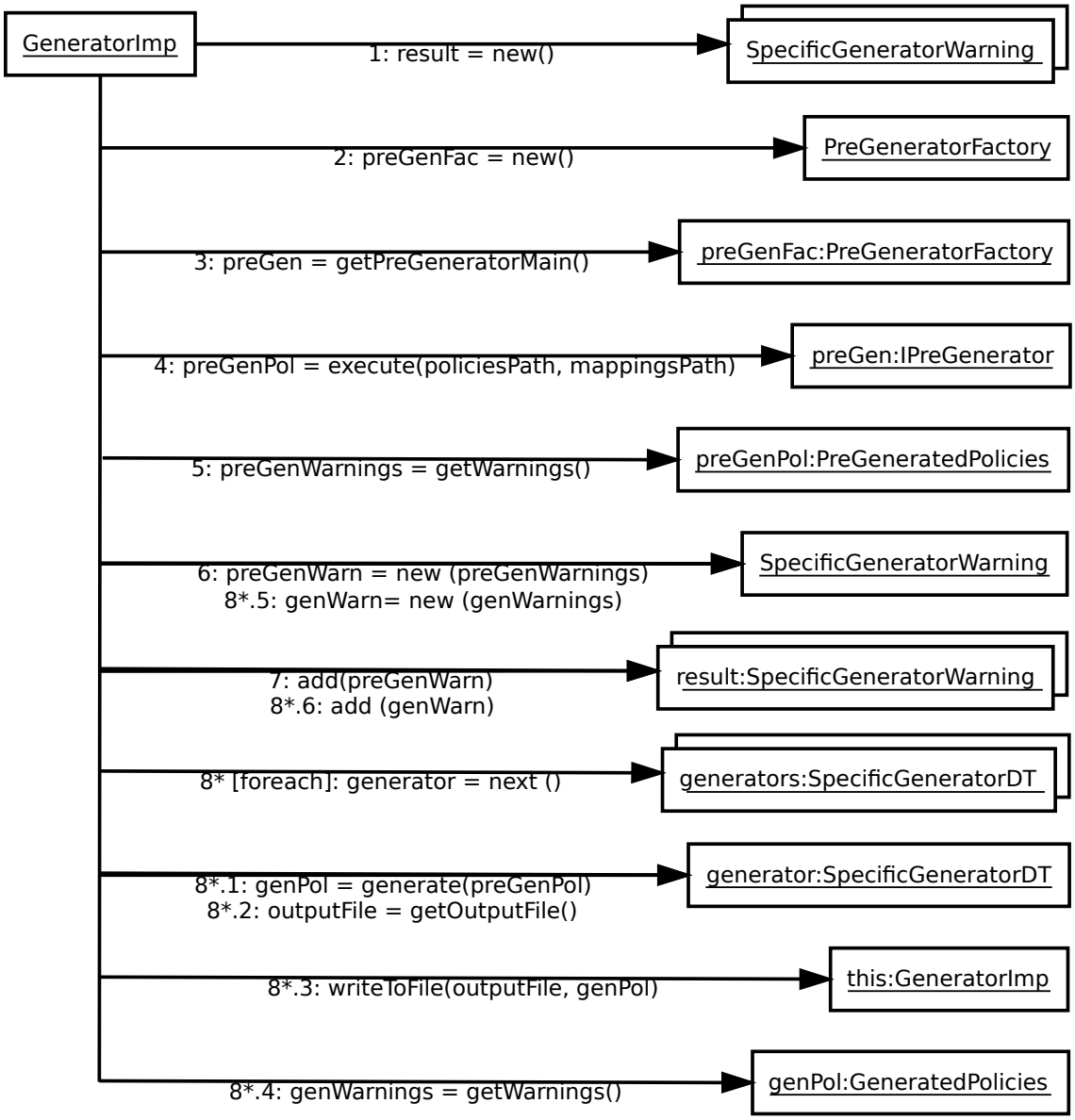
Operación
securityPolicies(String path)
Descripción
Recibe la ruta al archivo que contiene la política de seguridad descrita en SPLang, valida que el archivo exista y realiza el reconocimiento léxico y analiza que la sintaxis sea correcta. En caso de encontrarse errores se generan mensajes con información sobre la ubicación del problema. La validación incluye controles sobre los conjuntos de políticas definidos en el lenguaje, corroborando que sus identificadores sean únicos, así como que todos los conjuntos referenciados por las reglas hayan sido declarados. Genera una estructura de datos que corresponda con el tipo SecurityPolicies representado la política contenida en el archivo pero sin las peculiaridades de la herramienta que realiza el reconocimiento del lenguaje.
Pre-Condiciones
-
Post-Condiciones
-
Parámetros de entrada
• path: ruta absoluta a un archivo .splang

Retorno
Estructura de datos que representa la política de seguridad definida.
Excepciones
• SPLangSyntaxErrorException: se detecta un error de sintaxis en el archivo. Contiene la línea y la posición dentro de la línea para identificar dónde se encuentra el error así como el mensaje de error generado por ANTLR en este caso que sirven para corregir el mismo. • SPLangDuplicatedIdException: existen dos dataTypesValidations con el mismo identificador. • SPLangInvalidIdReferenceException: dentro de las reglas se encuentra una referencia a un DataTypeValidation inexistente, debido a que no existe uno con dicho id. • SPLangFileNotFoundException: no existe el archivo en “path”. • SPLangIOException: error al leer el archivo ubicado en “path”.

4 Anexo IV - Diagramas de comunicación

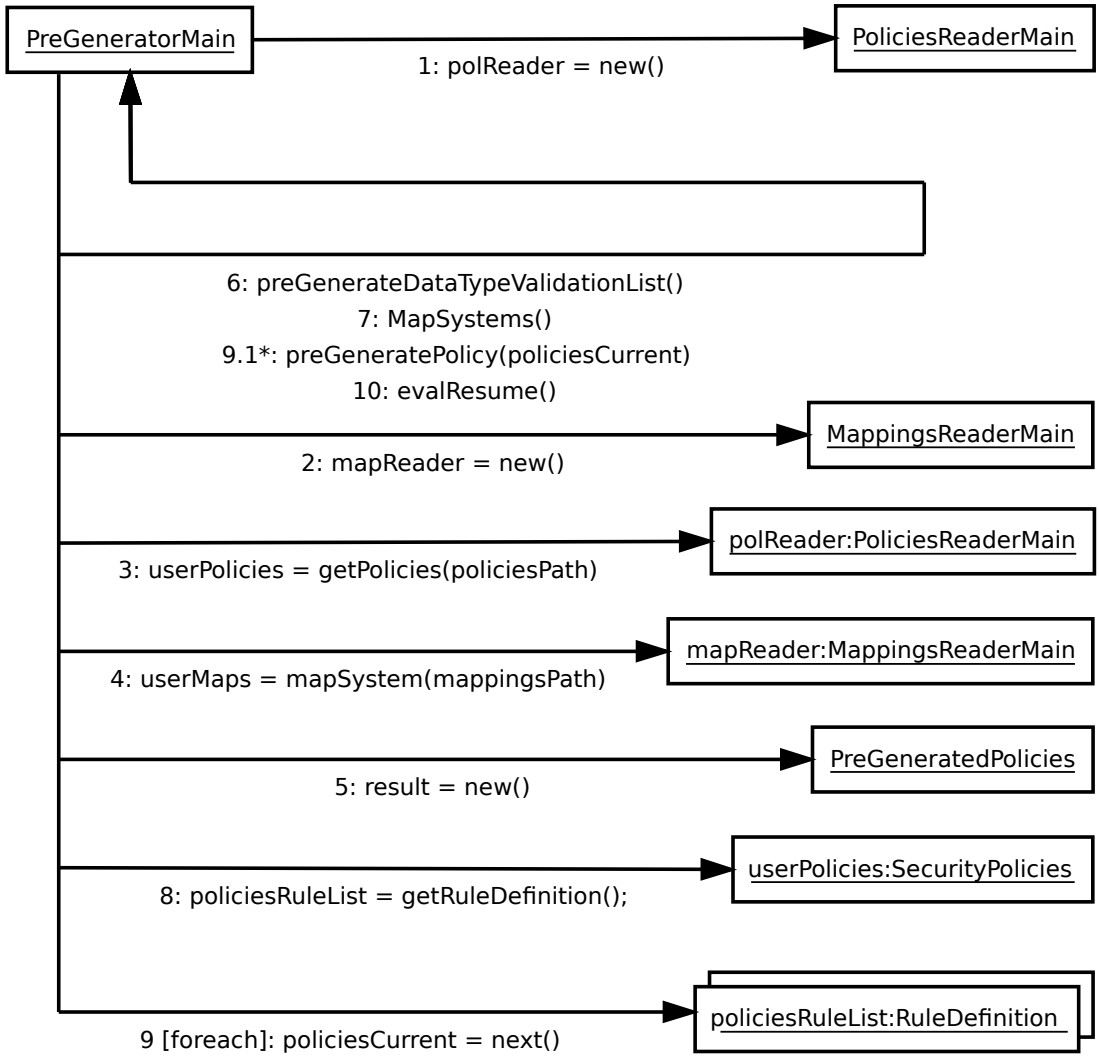
4.1 Generador::generate

List<SpecificGeneratorWarning> GeneratorImp::generate(String policiesPath, String mappingsPath, List<SpecificGeneratorDT> generators)

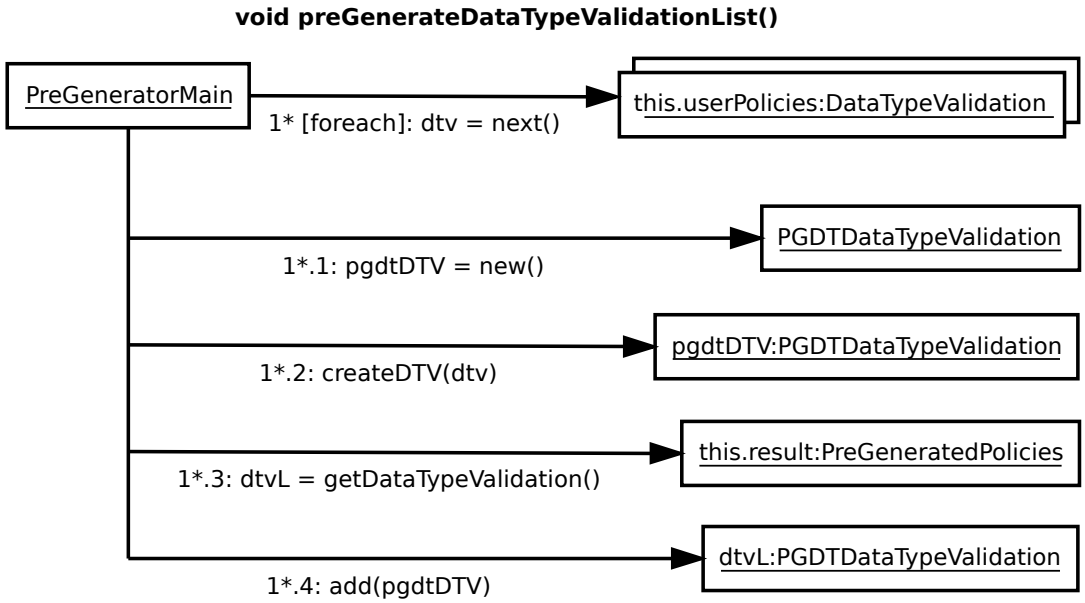


4.2 PreGeneratorMain::execute

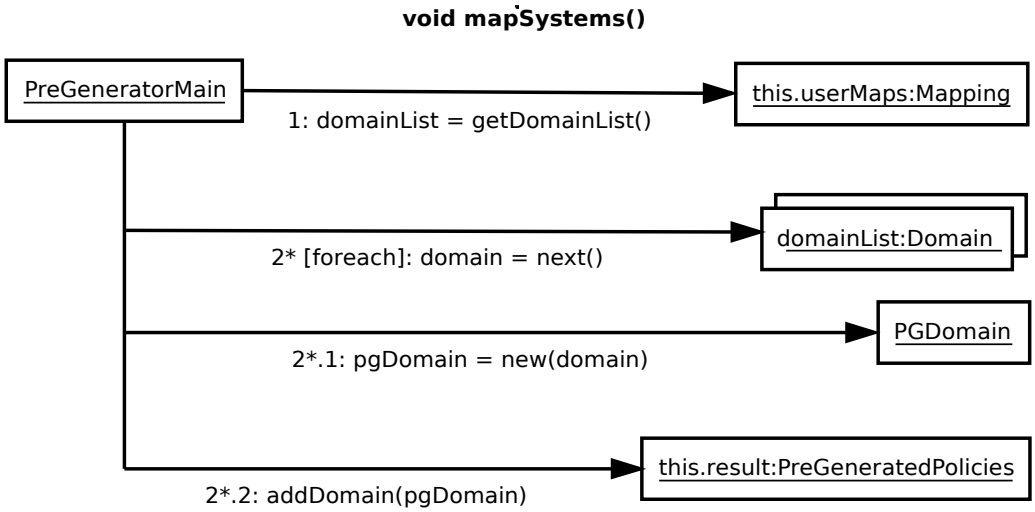
PreGeneratedPolicies execute(String policiesPath, String mappingsPath)



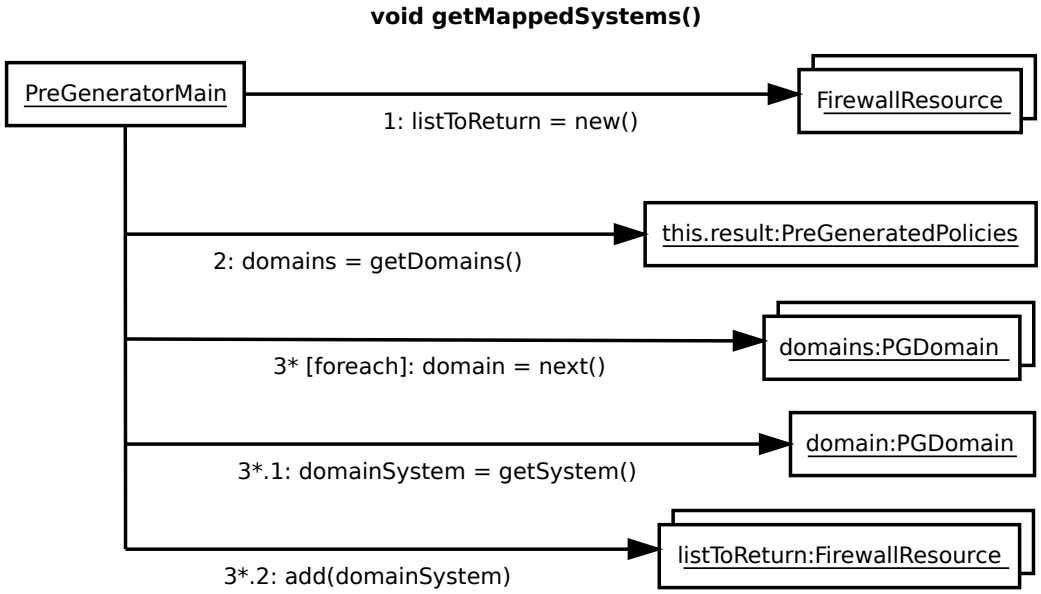
4.3 PreGeneratorMain::preGenerateDataTypeValidationList()



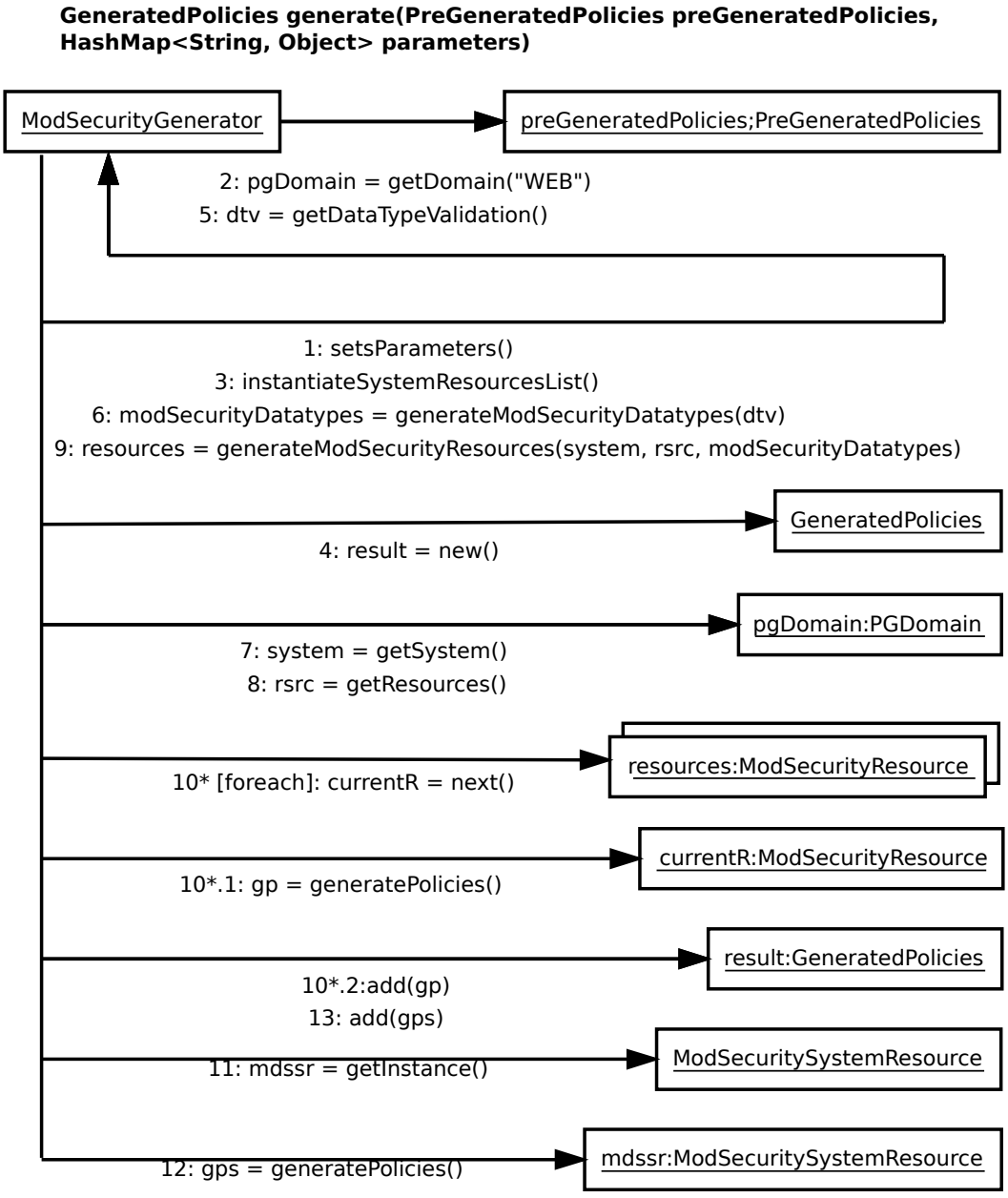
4.3.1 PreGeneratorMain::mapSystems()



4.3.2 PreGeneratorMain::getMappedSystems()

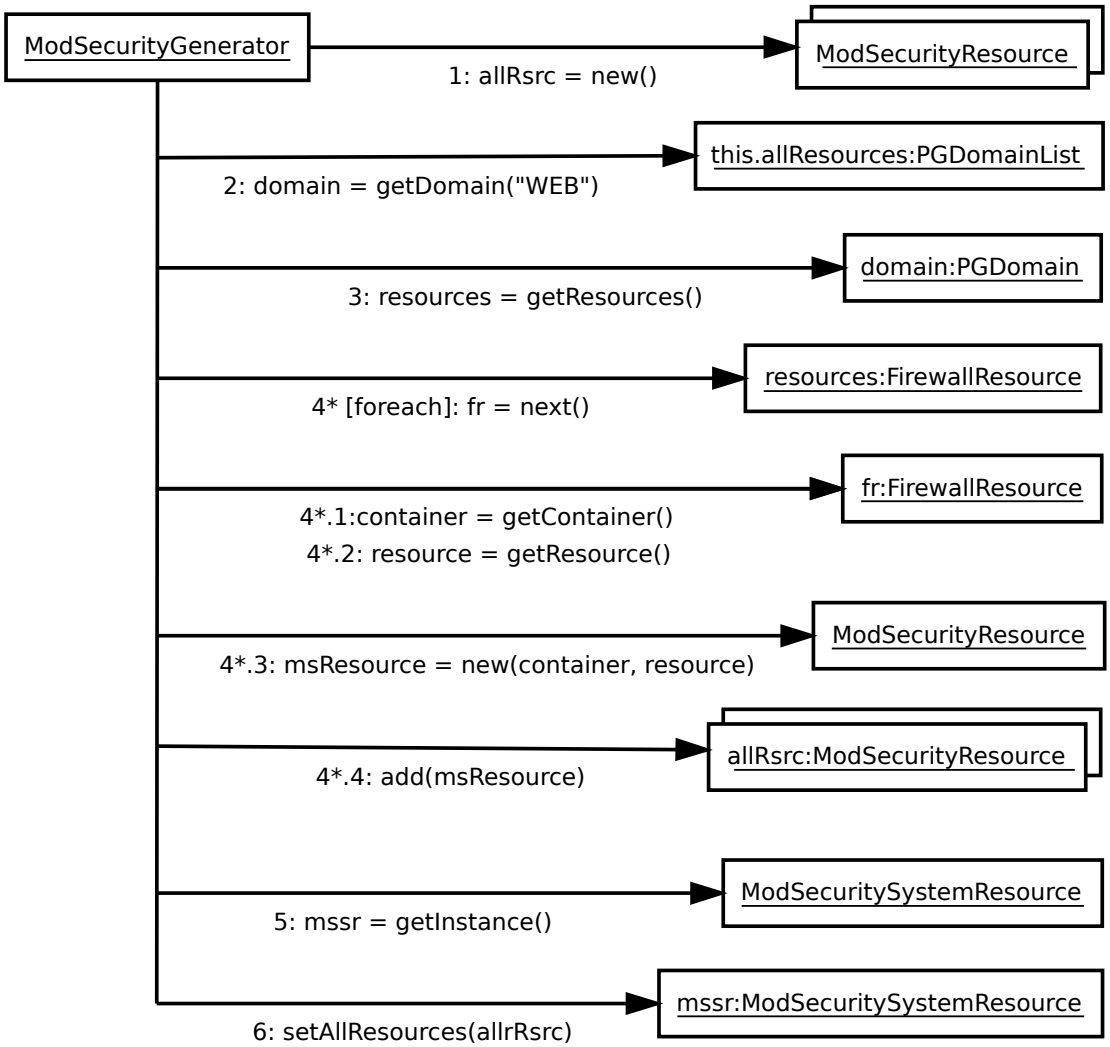


4.4 ModSecurityGenerator::generate



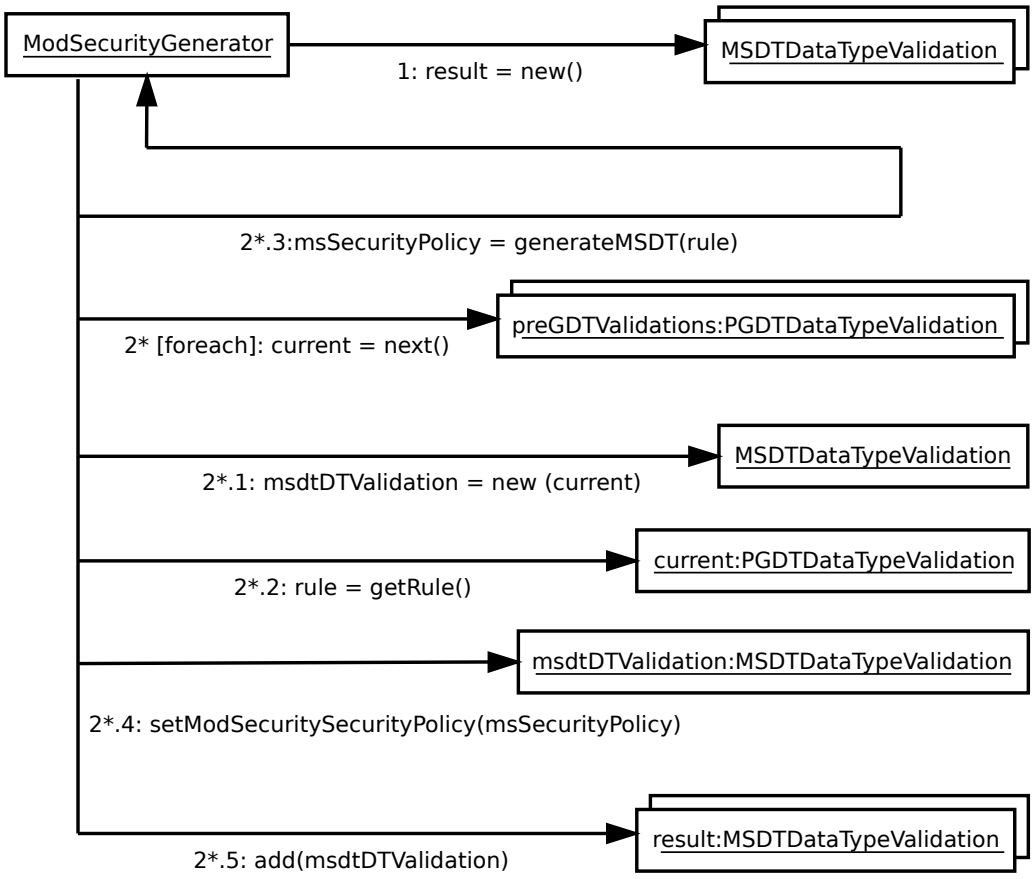
4.4.1 ModSecurityGenerator::instantiateSystemResourcesList()

void instantiateSystemResourcesList()



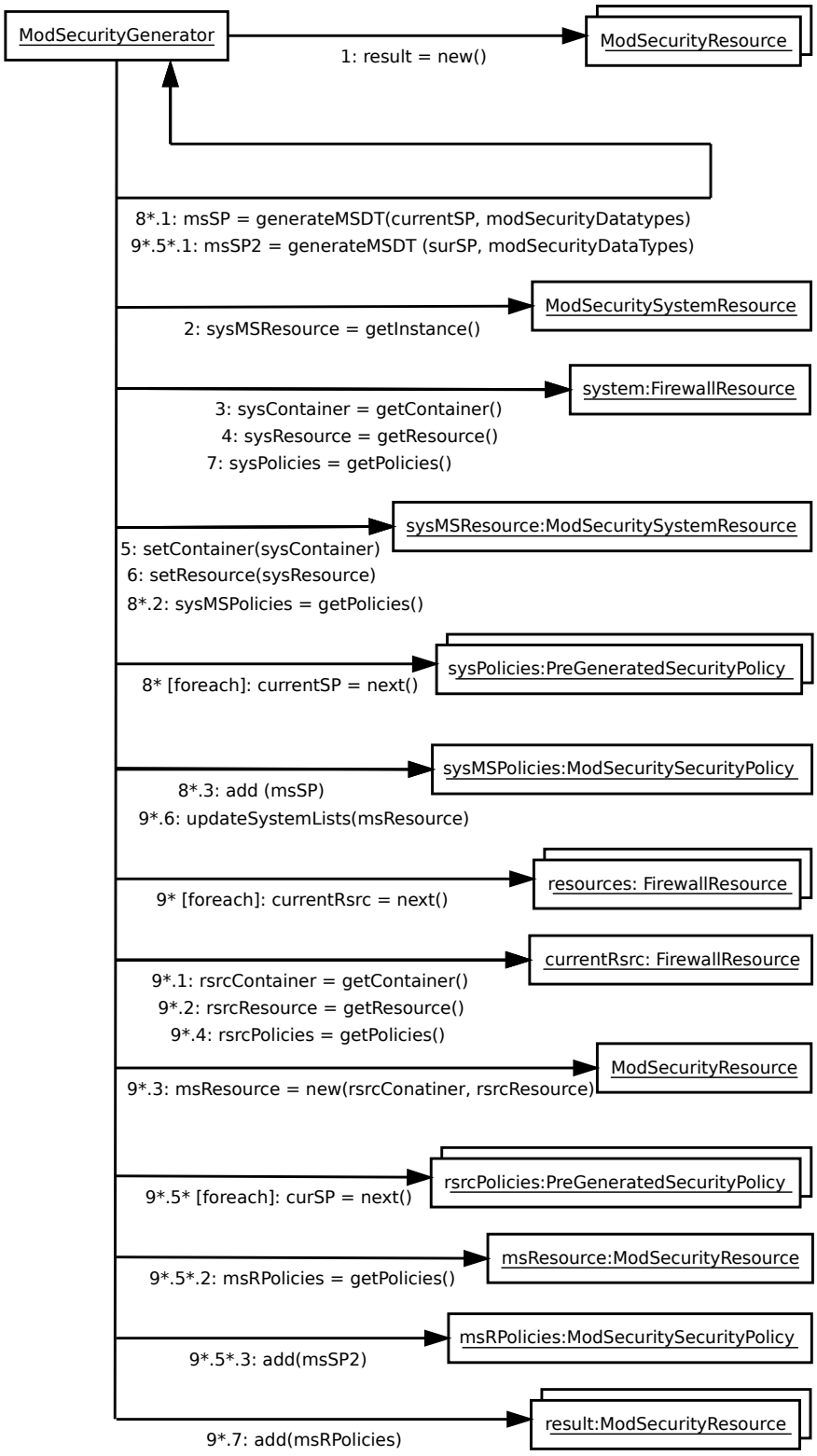
4.4.2 ModSecurityGenerator::generateModSecurityDatatypes

Hashtable <String, MSDTDataTypeValidation> generateModSecurityDatatypes (Hashtable<String, PGDTDataTypeValidation> preGDTValidations)



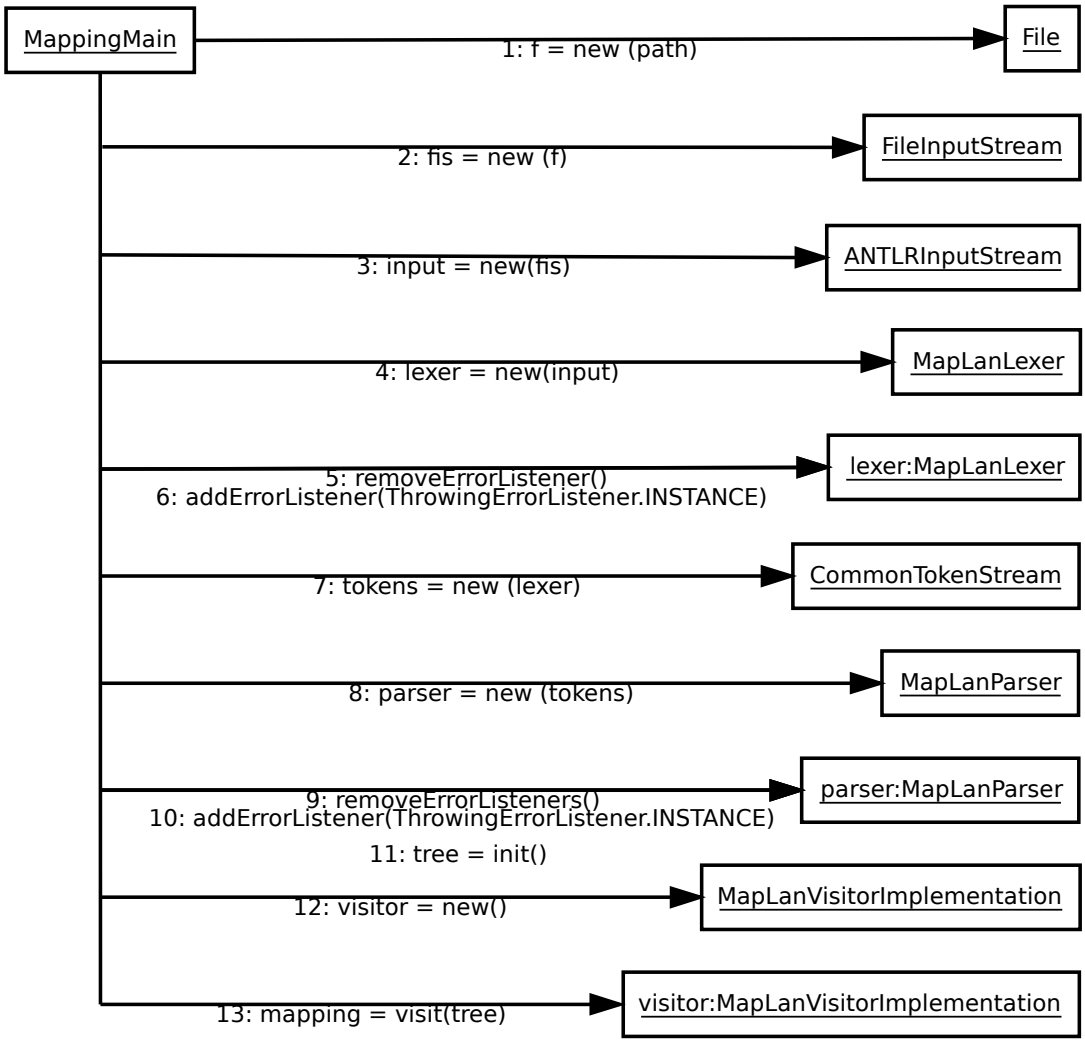
4.4.3 ModSecurityGenerator::generateModSecurityResources

**List<ModSecurityResource> generateModSecurityResources(
FirewallResource system, List<FirewallResource> resources,
Hashtable<String, MSDTDataTypeValidation> modSecurityDatatypes)**

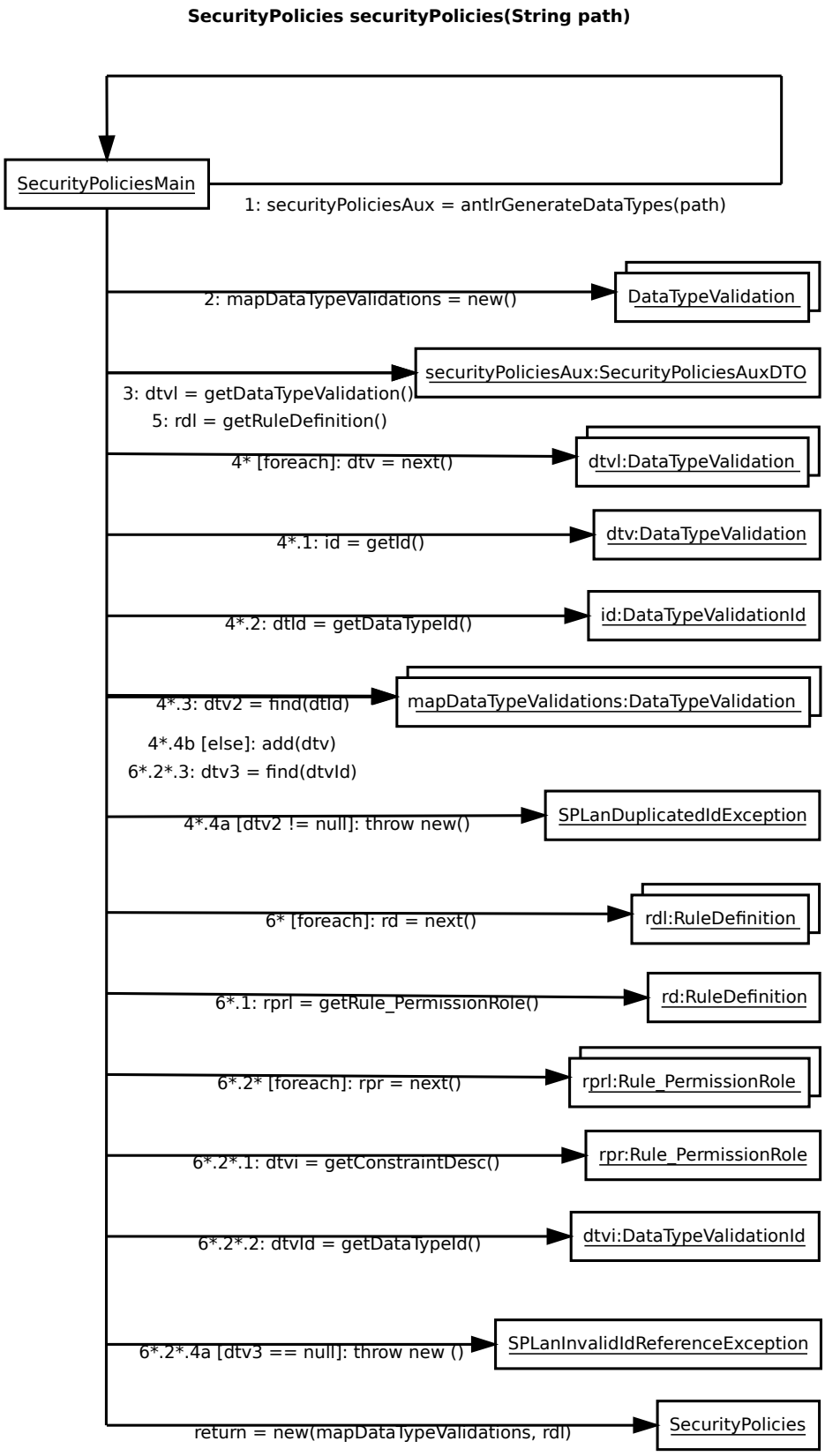


4.5 Imapping::map

Mapping::map(String path)



4.6 IsecurityPolicies::securityPolicies



Manual de usuario

Proyecto de grado

Definición y Enforcement de Políticas de Seguridad sobre Aplicaciones

(DEPSA)

Tutores

Felipe Zipitría – Rodrigo Martínez

Rodrigo de la Fuente

Luis González

Juan Pírez

Índice

1 DEPSA....1
1.1Compilación....1
1.2Ejecución....1
1.3Aplicación de las reglas....1
2 LeagueManager....3
2.1Instalación....3
2.2Acceso a la aplicación....3
3 Entorno de ejecución....4
3.1Stack LAMP....4
3.2ModSecurity....5
3.3CRS de OWASP....6
Referencias....7

1 DEPSA

A continuación se describen los pasos necesarios para compilar y ejecutar la aplicación DEPSA y así generar el archivo de configuración para ModSecurity.

1.1 Compilación

Para compilar el proyecto se proveen *script* para su ejecución con ant, que puede ser instalado en ubuntu ejecutando:

```
sudo apt-get install ant.
```

Los *script* de ant contienen las tareas `compile`, `dist` y `clean`.

La tarea `compile` realizará una compilación de todos los módulos, delegando la responsabilidad de compilar cada módulo a los archivos *build* definidos en cada módulo.

La tarea `dist` invocará a la tarea `compile` y luego generará los archivos *jars* de todos los módulos colocando dentro de la carpeta `dist`, los archivos generados para la distribución.

La tarea `clean` eliminará los archivos generados por la tarea `compile` y por la tarea `dist`.

En caso de no especificar una tarea en la invocación de ant, se realiza `clean` y `dist`.

1.2 Ejecución

Para generar la configuración para ModSecurity DEPSA se invoca con:

```
java –jar depsa_v1.0.0.jar generate {ruta al archivo con la politica
de seguridad} {ruta al archivo con los mapeos} {archivo donde se
almacenará la configuración}
```

Para chequear la sintaxis del archivo con la política de seguridad DEPSA se invoca con:

```
java –jar depsa_v1.0.0.jar checkPolicies {ruta al archivo con la
politica de seguridad}
```

Para chequear la sintaxis del archivo con los mapes DEPSA se invoca con:

```
java –jar depsa_v1.0.0.jar checkMappings {ruta al archivo con los
mapeos}
```

1.3 Aplicación de las reglas

Para aplicar las reglas generadas por depsa, lo recomendado es copiar el archivo con la configuración en la carpeta:

```
/etc/modsecurity/
```

y despues en el archivo

```
/etc/apache2/mods-enabled/security2.conf
```

agregar una regla para que incluya el archivo generado.

```
    /etc/modsecurity/specificrules/{archivo generado}
```

2 LeagueManager

Se describen en este capítulo los pasos necesario para instalar LeagueManager y los usuarios existentes para acceder a las funcionalidades de administración.

2.1 Instalación

Desplegar la carpeta `LeagueManager` en un servidor apache. Configurar la base de datos MySQL para que contenga un esquema llamado "`leaguemanager`" y que un usuario de nombre "`leaguemanager`" y contraseña "`leaguemanager`" tenga todos los permisos sobre ella.

Ejecutar sobre el esquema `leaguemanager` los *scripts* `structure.sql` y `data.sql` ubicados en la carpeta `SQL` que se encuentra dentro de la carpeta `LeagueManager`.

2.2 Acceso a la aplicación

Para acceder a LeagueManager acceder a la URL donde se encuentra el servidor y agregar `/LeagueManager`.

Para acceder a las funcionalidades de administración se cuenta con dos usuarios: "`otabarez`" con contraseña "`celeste`" y "`admin`" con contraseña "`admin`"

3 Entorno de ejecución

A continuación se describen los pasos necesarios para crear en un equipo con el sistema operativo Ubuntu 14.04, el entorno de ejecución para realizar el enforcement de la política de seguridad definda para LeagueManager.

Para esto se instala el Stack Lamp [1], que consiste en la instalación del servidor Web Apache, un motor de base de datos MySQL, PHP y de forma opción opcional phpMyAdmin [2], una interfaz gráfica para manipular la base de datos de MySQL. Además al servidor apache se le instala el módulo de ModSecurity [3] y las reglas del CRS defindo por OWASP [4].

3.1 Stack LAMP

Para instalar un servidor apache ejecutar en linea de comandos:

```
sudo apt-get update
sudo apt-get install apache2
```

Puede comprobarse la correcta instalación ingresando a la URL http://127.0.0.1.

Luego de instalar apache, se instala MySQL ejecutando:

```
sudo apt-get install mysql-server php5-mysql
```

Soliticará que se ingrese una contraseña de root para realizar la administración de MySQL.

Luego de completar la instalación, se deben ejecutar los siguientes comandos para completar la configuración.

```
sudo mysql_install_db
sudo mysql_secure_installation
```

Al completar la configuración de mysql, proseguiremos instalando PHP, para esto ejecutar:

```
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
```

Reiniciar Apache para que surjan efecto los cambios.

```
sudo service apache2 restart
```

Con esto queda instalado el entorno Apache – MySQL – PHP.

Para comodidad, es posible la instalación de phpMyAdmin para la configuración de la base de datos de MySQL. Para esto ejecutar:

```
sudo apt-get install phpmyadmin apache2-utils
```

Durante la instalación phpMyAdmin seguir los siguientes pasos

- Seleccionar servidor Apache2
- Seleccionar *yes* cuando consulte si se desea configurar la base de datos usando dbconfig-common.
- Ingresar la contraseña de MySQL ingresada.
- Ingresar una contraseña para administrar phpMyAdmin.

Luego de la instalación, agregar phpMyAdmin a la configuración de apache. Para esto editar el archivo `apache2.conf` con:

```
sudo nano /etc/apache2/apache2.conf
```

y agregar el arhivo de configuración de phpMyAdmin al final del archivo

```
Include /etc/phpmyadmin/apache.conf
```

Reiniciar el servidor apache con:

```
sudo service apache2 restart
```

3.2 ModSecurity

Para instalar ModSecurity ejecutar:

```
sudo apt-get install libapache2-modsecurity
```

Luego de realizada la instalación, reiniciar el servidor apache.

```
sudo service apache2 restart
```

Si se desea que ModSecurity bloquee los pedidos que coincidan con las reglas, se debe modificar el archivo modsecurity.conf con:

```
sudo nano /etc/modsecurity/modsecurity.conf
```

y modificar la linea:

```
SecRuleEngine DetectionOnly
```

por:

```
SecRuleEngine On
```

Con esto ModSecurity se encargará de bloquear los pedidos que cumplan alguna de las reglas configuradas.

3.3 CRS de OWASP

Para instalar el CRS de OWASP ejecutar:

```
sudo apt-get install libapache2-mod-security2 libapache2-mod-evasive
```

Para habilitar las reglas del CRS editar el archivo:

```
/etc/apache2/mods-enabled/security2.conf
```

y agregar:

```
IfModule security2_module>
    # Include all the *.conf files in /etc/modsecurity.
    # Keeping your local configuration in that directory
    # will allow for an easy upgrade of THIS file and
    # make your life easier
    IncludeOptional /etc/modsecurity/*.conf
    # OWASP CRS
    Include "/usr/share/modsecurity-crs/*.conf"
    Include "/usr/share/modsecurity-crs/activated_rules/*.conf"
</IfModule>
```

Es necesario crear links simbólicos con las reglas ha activar. Para esto se usan el comandos:

```
sudo ln -s /usr/share/modsecurity-crs/base-
rules/modsecurity_crs_41_sql_injection_attacks.conf
/usr/share/modsecurity-crs/activated-rules/

sudo ln -s /usr/share/modsecurity-crs/base-
rules/modsecurity_crs_41_xss_attacks.conf /usr/share/modsecurity-
crs/activated-rules/
```

Con eso esta configurado para utilizar las reglas definidas en el CRS.

Referencias

[1] “How To Install Linux, Apache, MySQL, PHP (LAMP) Stack on Ubuntu 14.04.” *DigitalOcean*. Accessed September 19, 2015. https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-14-04.

[2] “How To Install And Secure PhpMyAdmin on Ubuntu 12.04.” *DigitalOcean*. Accessed September 19, 2015. https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-phpmyadmin-on-ubuntu-12-04.

[3] “How To Set Up mod_security With Apache on Debian/Ubuntu.” *DigitalOcean*. Accessed September 19, 2015. https://www.digitalocean.com/community/tutorials/how-to-set-up-mod_security-with-apache-on-debian-ubuntu.

[4] “How To Install mod_security and mod_evasive on an Ubuntu 14.04 VPS.” *RoseHostingcom Linux VPS Hosting Blog*. Accessed September 19, 2015. https://www.rosehosting.com/blog/how-to-install-mod_security-and-mod_evasive-on-an-ubuntu-14-04-vps/.