Proyecto de Grado

Ingeniería en Computación

Integración del WAF ModSecurity y la plataforma para inteligencia de amenazas MISP

Guillermo Franco

guillermo.franco@fing.edu.uy

Tutores:

Gustavo Betarte

Rodrigo Martinez

Marcelo Rodriguez

Instituto de Computación

Facultad de Ingeniería

Universidad de la República

Montevideo — Uruguay

Diciembre, 2021

Resumen

En los últimos años se ha observado un aumento en el uso de plataformas para compartir información sobre ataques cibernéticos, tales como la plataforma Malware Information Sharing Platform (MISP), una de las plataformas de código abierto más populares. Por otro lado, también se popularizó el uso de Web Application Firewalls (WAFs) tales como ModSecurity, un WAF de código abierto que se ha convertido en el standard de facto. El ritmo acelerado con el que surgen nuevas amenazas cibernéticas supone un trabajo constante para el administrador de seguridad, ya que debe actualizar la defensa de sus sistemas conforme surjan nuevas amenazas. En este trabajo se analizan las distintas vias de comunicación entre ModSecurity y MISP y se implementa una integración entre ambas tecnologías, con el propósito de actualizar la defensa del WAF automáticamente a partir de información obtenida de MISP, la cual se encuentra en constante actualización. Particularmente, la solución implementada logra actualizar la defensa en tiempo real y es fácilmente extensible para casos de uso no contemplados en este trabajo.

Palabras claves: ModSecurity, MISP, TISP, WAF

Índice general

Capítulo 1

Introducción

En los últimos años se ha observado un aumento en la cantidad y complejidad de los ataques cibernéticos[1], en particular los ataques web. Esto ha provocado que las soluciones de defensa deban ser continuamente actualizadas para mantenerse funcionales. Este es uno de los motivos que provocó el surgimiento de distintas plataformas para compartir información sobre amenazas cibernéticas como MISP, así como provocó que se popularizara el uso de Web Application Firewalls como ModSecurity.
MISP es una plataforma para compartir información sobre amenazas cibernéticas en forma de indicadores de compromiso (artefactos observados sobre un ataque). Una instancia MISP puede obtener y compartir información sincronizándose con un repositorio compatible con MISP (otras instancias MISP, CSVs, etc.).
ModSecurity es un Web Application Firewall, se encarga de aplicar una serie de reglas a una comunicación HTTP para cubrir ataques cibernéticos comunes. Las reglas son definidas por el usuario y le permiten realizar determinadas acciones sobre una solicitud HTTP entrante en función de los elementos de la misma, como bloquearla en caso de que algún elemento de la solicitud sea interpretado como malicioso.

En la siguiente sección se plantea el problema a resolver junto con los objetivos, finalmente se describe la estructura de este documento.

1.1. Problema y objetivos

Dado un WAF (ModSecurity) y una plataforma para compartir indicadores de compromiso (MISP), se busca que el WAF sea "reforzado" mediante información obtenida en tiempo real de la plataforma, con el objetivo de mejorar la protección del servidor web que se está protegiendo. Con cada solicitud entrante, el WAF debe comunicarse con la plataforma para evaluar si algún elemento de la solicitud se encuentra catalogado como malicioso para un conjunto de casos de uso previamente definidos.

Para esto se definieron una serie de requerimientos generales así como un conjunto de casos de uso. Se planteron dos requerimientos generales: por un lado, la solución debe ser extensible, es decir, se deben poder soportar casos de uso ajenos a los establecidos sin necesidad de modificar el código existente. Por otro lado, el procesamiento debe ocurrir en tiempo real (cada solicitud entrante debe ser evaluada con información obtenida de la plataforma en tiempo real). Con respecto a los casos de uso, el sistema desarrollado debe poder detectar una solicitud proveniente de una IP catalogada como maliciosa, una solicitud con archivos adjuntos catalogados como

maliciosos, una solicitud para acceder a un archivo catalogado como malicioso y una solicitud de acceso a una URL catalogada como maliciosa.

Se exploraron distintas vías de comunicación para integrar ambas tecnologías y se eligió la opción más adecuada teniendo en cuenta los requerimientos y casos de uso planteados. Para esto se analizaron las interfaces de comunicación de las tecnologías planteadas así como tecnologías externas para ser utilizadas en la comunicación. Luego la opción elegida fue implementada.

Se logró implementar una solución sin utilizar tecnologías ajenas al ecosistema de ModSecurity y MISP, cumpliendo los requerimientos planteados así como todos los casos de uso. La solución desarrollada puede ser extendida fácilmente para soportar casos de uso distintos a los planteados sin la necesidad de modificar el código ya existente, en particular no es necesario recompilar el código desarrollado para ModSecurity.

1.2. Estructura del documento

En el capítulo 2 se introducen las principales tecnologías utilizadas en el proyecto, particularmente ModSecurity y MISP y algunos conceptos generales, así como una breve sección de antecedentes. En el capítulo 3 se definen los requerimientos del proyecto y se presenta un análisis de los distintos métodos existentes para integrar ModSecurity y MISP. En el capítulo 4 se plantea la arquitectura general de la solución implementada así como el flujo entre los distintos componentes desarrollados. En el capítulo 5 se describen con detalle los componentes implementados, así como el conjunto de reglas y directivas implementadas para ModSecurity. Finalmente, en el capítulo 6 se plantean las conclusiones y el trabajo futuro.
En el anexo se describe un template de objeto comúnmente usado a modo de ejemplificar la definición de los mismos.

Capítulo 2

Estado del arte

En este capítulo se introducen las tecnologías principales utilizadas en el proyecto (ModSecurity y MISP) junto con sus interfaces de comunicación, así como conceptos generales tales como indicadores de compromiso. Finalmente, en la sección antecedentes se presenta brevemente otro proyecto de integración ModSecurity - MISP.

2.1. ModSecurity

Un Web Application Firewall (WAF)[2] es un firewall que protege aplicaciones web analizando el tráfico a nivel de la capa 7 del modelo OSI[3]. Se encarga de monitorear, filtrar y bloquear tráfico HTTP desde y hacia una aplicación web. Puede ser considerado como un proxy reverso.

ModSecurity[4] es un módulo WAF de código abierto y multiplataforma. Permite obtener más visibilidad sobre el tráfico HTTP en el contexto de aplicaciones web y provee un poderoso lenguaje de reglas así como una API para implementar protecciones avanzadas. Aplica un set de reglas a una comunicación HTTP[1]y generalmente estas reglas cubren ataques comunes tales como XSS, inyecciones SQL, etc. Puede ser integrado con herramientas como el OWASP CRS (detallado más adelante), el cual intenta cubrir ataques comunes como los mencionados con mínima intervención del administrador.
Principalmente se utiliza para los siguientes casos de uso:

- Control de acceso y monitoreo de seguridad de aplicaciones en tiempo real
- Hardening de aplicaciones web: permite reducir la superficie de ataque reduciendo aspectos de HTTP, tales como métodos de petición, etc.
- Evaluación continua de seguridad pasiva: es una variación de monitoreo en tiempo real, donde en vez de centrarse en el comportamiento de terceros se centra en el comportamiento del sistema. Funciona como sistema de alerta temprana
- Logging de tráfico HTTP completo

Uno de los aspectos más útiles para el bloqueo de solicitudes maliciosas es el control de acceso y monitoreo de seguridad de aplicaciones web en tiempo real: ModSecurity proporciona acceso al stream de tráfico HTTP en tiempo real, junto con la habilidad

[1] A menos que se especifique lo contrario se usa 'HTTP' como sinónimo de HTTP(s)

de inspeccionarlo, bloquearlo, etc. También permite trackear elementos del sistema en el tiempo mediante el mecanismo de persistencia.

Se rige por 4 principios rectores:

- Flexibilidad: ModSecurity provee un poderoso lenguaje de reglas que permiten conseguir exactamente lo que se necesita, aplicándolas solamente dónde se necesitan
- Pasividad: ModSecurity no interactúa en las transacciones (a menos que se especifique lo contrario)
- Previsibilidad: ModSecurity no intenta ser perfecto pero sí previsible: entendiendo como funciona es posible identificar sus puntos débiles y trabajar alrededor de ellos
- Calidad sobre cantidad: el equipo de ModSecurity decidió no implementar muchas funcionalidades para poder incluir y mantener funcionalidades de buena calidad

La flexibilidad es uno de los principios más importantes para la defensa: el lenguaje de reglas provisto permite aplicar acciones en función de la solicitud y la respuesta con un gran nivel de granularidad. Es un lenguaje especializado diseñado para trabajar con los datos transaccionales de HTTP, de tal forma que las operaciones comunes son simples mientras que las complejas son posibles.

2.1.1. Reglas

En ModSecurity todo gira en torno a 2 elementos: la configuración y las reglas. La configuración le indica a ModSecurity cómo procesar los datos; las reglas deciden qué hacer con los datos ya procesados.

Una regla es una directiva de ModSecurity al igual que cualquier otra. La diferencia se encuentra en que esta directiva es mucho más poderosa que las demás en términos de lo que es capaz de representar. Esta compuesta por 4 componentes:

- **Variables:** indican las partes de una transacción HTTP con las que la regla trabaja. ModSecurity extrae información de cada transacción y la disponibiliza a través de variables. En otras palabras, indican a la regla en dónde buscar.
- **Operadores:** especifican cómo una variable debe ser analizada, por ejemplo mediante expresiones regulares. Sólo un operador es permitido por regla.
- **Transformaciones:** son funciones que realizan modificaciones sobre la entrada antes de que los operadores sean ejecutados. Usualmente son usados para contrarrestar evasiones de los controles del cliente normalizando la entrada (ej. eliminar caracteres extraños de un email proporcionado), pero pueden ser usados para decodificar datos. Una regla puede especificar varias transformaciones.

- **Acciones:** especifican qué se debería hacer en caso de que la regla aplique.

Una regla tiene la siguiente estructura:

```
SecRule VARIABLES "OPERADOR" "TRANSFORMACIONES, ACCIONES"
```

Listing 2.1: Estructura general de regla ModSecurity

A modo de ejemplo, tomemos la siguiente regla:

```
SecRule REQUEST_URI "@streq /index.php" "id:1,phase:1,t:lowercase,deny"
```

Listing 2.2: Regla de ejemplo

La única variable involucrada en la regla del listing 2.2 es `REQUEST_URI`. Esta variable guarda la URL completa incluyendo la query string. El único operador es `streq`, el cual realiza una comparación entre strings, devolviendo true si el parámetro es idéntico al input (`REQUEST_URI` == '/index.php'). La acción `id` asigna un ID único a la regla y la acción `phase` coloca a la regla en una de las 5 posibles fases (1: Request Headers, 2: Request Body, 3: Response Headers, 4: Response Body, 5: Logging). La transformación `lowercase` convierte todos los caracteres a minúscula, y la acción `deny` detiene el procesamiento de reglas e intercepta la transacción.

Esta regla captura cada solicitud HTTP y obtiene únicamente la URI de la misma. Luego transforma la URI a minúscula (`t:lowercase`), para luego chequear que el valor transformado sea igual a '/index.php'. De serlo, ModSecurity rechaza la solicitud, es decir, detiene el procesamiento de más reglas e intercepta la solicitud.

2.1.2. Modos de detección

Tradicionalmente, en ModSecurity existen dos modos de detección: el modo de sólo detección, en el cual no se bloquea ninguna solicitud independientemente de la regla activada, y el modo de bloqueo, donde sí se bloquea la solicitud en caso de que la regla activada lo indique. Existe un submodo de detección no nativo llamado *Anomaly Scoring*[5][6][7], el cual es implementado por una librería externa.

Para utilizar el modo *Anomaly Scoring* es necesario contar con CRS. El *Core Rule Set*[8] (CRS) es un set de reglas genéricas de detección de ataques para ser utilizadas junto con ModSecurity. También proporciona un sistema de puntajes para calificar la severidad de una regla [9]. Cada regla puede adquirir 1 de los 8 niveles de severidad, donde un valor menor indica una mayor severidad.
En el modo de detección *Anomaly Scoring*, este sistema de puntajes funciona en forma paralela con otro sistema de puntajes: dentro del ciclo de vida de una solicitud, CRS lleva registro de todas las reglas que fueron ejecutadas junto con un puntaje (puntaje de anomalía) que es incrementado (o no) luego de cada regla. Cada regla aumenta el puntaje en función de la severidad de la misma.
Este puntaje de anomalía es utilizado para interceptar la solicitud: si el puntaje excede determinado threshold se intercepta la solicitud. Se cuenta con un threshold para las solicitudes y otro para las respuestas.

ModSecurity cuenta con otro modo de detección, el modo tradicional. Este es un modo más básico, donde no se registra un puntaje de anomalía y cada regla se encuentra autocontenida: las reglas individuales no tienen estado y son independientes a las demás, no tienen información sobre la activación de reglas anteriores. Esto provoca que cada regla sea la encargada de decidir si se captura o no la solicitud.
Este modo es menos flexible que el anterior y más estricto: normalmente provoca una mayor cantidad de falsos positivos/negativos. Tiene la ventaja de que consume menos recursos.

2.2. Indicadores de Compromiso

Los IoC (Indicadores de Compromiso)[10] son artefactos observados sobre un atacante: sus técnicas, tácticas, procedimientos o herramientas e infraestructura asociadas. Estos indicadores pueden ser observados en una combinación de niveles de red o host y pueden, con distintos grados de confianza, ayudar a identificar la ocurrencia de una intrusión o de actividad asociada a intrusiones. Estos IoCs son usados por los equipos de ciberseguridad para proteger sus redes y equipos. Algunos ejemplos de IoCs pueden incluir:

- direcciones IP
- nombres de dominio
- valores *TLS Server Name Indication*
- información sobre certificados
- firmas tales como patrones de código binario y strings
- hashes de binarios/scripts maliciosos
- herramientas de ataque
- técnicas de ataque

Los IoCs pueden ser de diferente calidad. Un IoC sin contexto (como por ejemplo el actor al cual se relaciona, la última vez que se observó en uso, su tiempo de vida esperado u otros IoCs relacionados) no aporta mucha información para la defensa de la red: un defensor podría realizar distintas acciones con un IoC, como monitorearlo, bloquearlo, loguearlo, etc. en función del contexto. Sin este contexto asociado la usabilidad de un IoC se reduce considerablemente. Por otro lado, un IoC con un contexto asociado es mucho más útil para un defensor de la red, el cual luego puede tomar una decisión informada sobre cómo usarlo para proteger la misma.
Una de sus ventajas es que son escalables y fáciles de implementar. Son fáciles de compartir debido a la estandarización de formatos de IoCs, lo que motivó la creación de distintas plataformas para compartirlos. Cuando la implementación automática de los IoC funciona bien, las organizaciones y los usuarios obtienen protección general

con un mínimo de intervención humana, permitiendo a las organizaciones focalizar sus defensas contra riesgos particulares.

2.3. MISP

En los últimos años, las organizaciones han demostrado un aumento en su voluntad de compartir información y conocimiento sobre vulnerabilidades/amenazas/incidentes/mitigación de ataques con el objetivo de protegerse colectivamente ante los sofisticados ciberataques de hoy en día[11]. Con este fin, fueron creadas distintas plataformas bajo el amplio término *"Threat Intelligence Sharing Platform"* (TISP).
Muchas de estas plataformas comparten esa información mediante IoCs, por lo que utilizan primariamente los observables e indicadores[12]. Los observables son propiedades o eventos medibles en el ámbito de computadoras y redes, solo especifican "hechos" sin un contexto para interpretarlos (ej. una dirección IP). Por otro lado, los indicadores son observables dentro de un contexto particular. Ese contexto es el que le otorga significado al observable. Por ejemplo, una dirección IP es un observable mientras que IPs particulares pueden ser indicadores de una intención maliciosa.
La arquitectura consiste en 8 conceptos de amenaza cibernética y tiene en cuenta sus relaciones. Estos conceptos son:

- Ciber Observables (ej. Direcciones IP, nombres de archivo, hashes)
- Indicadores (transmiten patrones observables específicos combinados con información contextual destinada a representar artefactos y/o comportamientos de interés dentro de un contexto de seguridad cibernética)
- Incidentes
- Tácticas/técnicas/procedimientos de adversarios (incluye patrones de ataque, *kill chains*[13], etc.)
- Objetivos de explotación (ej. vulnerabilidades, debilidades)
- Vías de acción (ej. respuesta a incidentes, estrategias de mitigación)
- Campañas de ciberataque: agrupación de comportamientos que describe un conjunto de actividades o ataques maliciosos que ocurren durante un período contra un conjunto específico de objetivos. Son comúnmente atribuidas a un conjunto de actores
- Actores de ciberataques: individuos, grupos u organizaciones que operan con intenciones maliciosas (ej. un conjunto de IPs determinadas)

MISP[14] (Malware Information Sharing Platform), tal como lo indica su nombre, es una plataforma para compartir información sobre malware, y en particular permite compartir IoCs. Fue desarrollado por NATO para facilitar el seguimiento y el análisis de malware poco usual. Posee algunas características que lo convierten en un buena plataforma de intercambio para este problema en particular, entre ellas:

- BD eficiente de IoC/indicadores que permite almacenar información tanto técnica como no técnica sobre muestras de malware, incidentes, atacantes, etc.
- Correlación automática para encontrar relaciones entre atributos e indicadores de malware, campañas de ataque o análisis
- Modelo de datos flexible donde objetos complejos pueden ser expresados y vinculados
- Funcionalidad para compartir, incorporada para facilitar el intercambio de datos usando distintos modelos. MISP puede sincronizar automáticamente eventos y atributos entre diferentes instancias de MISP
- Permite persistir datos en formatos estructurados
- Permite exportar datos en varios formatos
- Permite importar datos desde OpenIOC, GFI sandbox, ThreatConnect CSV o formato MISP
- API flexible para integrar MISP con soluciones personalizadas. Esta es una de las características más importantes en este caso de uso, ya que facilita la integración con ModSecurity
- Taxonomía ajustable para clasificar/etiquetar eventos según clasificación personalizada

2.3.1. Eventos MISP

En MISP la información es representada a través de eventos. Un evento es el registro de una muestra, el cual se encuentra compuesto por sus atributos conectados. Tienen múltiples propiedades, entre las más importantes:

- **Date**: fecha en la cual ocurrió el incidente
- **Threat Level (TL):** nivel de amenaza, puede ser:
  - Undefined (4): sin definir
  - Low (3): Malware general
  - Medium (2): Amenazas persistentes avanzadas (APT por sus siglas en inglés)
  - High (1): APTs sofisticadas y ataques 0day
- **Analysis**: indica la etapa del análisis en la que se encuentra el evento, puede ser:
  - Initial: el análisis se encuentra apenas comenzando
  - Ongoing: se encuentra en progreso
  - Completed: fue completado
- **Event Info**: nombre o título del evento, descripción breve y concisa

2.3.2. Atributos MISP

Los eventos se encuentran compuestos por atributos[15] (o indicadores[2], como son llamados en otras plataformas). En MISP estos eventos pueden ser indicadores de la red (ej. direcciones IP), indicadores del sistema (ej. un string en memoria) y otros datos no necesariamente relacionados al mundo cibernético (ej. detalles de cuentas bancarias).
Un atributo es descrito por su tipo (ej. MD5, url). Todo atributo se encuentra en una categoría (ej. *Network activity*), lo cual lo coloca en un contexto determinado.
No todos los atributos disponibles en MISP son aplicables en el contexto de una aplicación web: no tendría sentido, por ejemplo, tener en cuenta los atributos pertenecientes a la categoría *Persistence mechanism*, la cual refiere a mecanismos usados por el malware para iniciar al momento de bootear.

Un atributo aislado no aporta mucha información sobre un evento, por más que se disponga de su tipo y categoría. Es por esto que se utilizan combinaciones de los mismos en forma de objetos.

2.3.3. Objetos MISP

Los objetos MISP[16][17] permiten combinaciones avanzadas de atributos, y un evento puede tener varios objetos asociados, de la misma forma que puede tener varios atributos asociados. La creación de estos objetos y los atributos que lo componen está basada en casos de usos reales de ciber seguridad y prácticas existentes en "information sharing".
Los objetos se encuentran asociados a templates de objeto, los cuales funcionan a modo de plantilla para crear objetos y son identificados por su UUID. Estos templates son descritos en formato JSON y se componen de los siguientes atributos:

- **name**: nombre del template
- **meta-category**: la categoría a la que pertenece el template (file, network, financial, misc, internal)
- **description**: resumen de la descripción del template
- **version**: número de version, es utilizado por MISP para saber cuándo un template debe ser actualizado
- **attributes**: objeto JSON listando todos los atributos que componen al template. Cada uno de estos atributos debe referenciar una definición de atributo en MISP (ej "ip-src"), y puede contener información extra como descripción, orden en que figurará en la interfaz de usuario (UI), etc.
- **required**: lista de los atributos mínimos requeridos para poder describir al objeto

[2]No confundir con IoC

- **requiredOneOf**: lista que contiene los atributos donde al menos uno debe estar presente para poder describir al objeto
- **uuid**: id que identifica al template de objeto globalmente, es utilizado en los objetos para referenciar al template

Emplear objetos MISP sobre atributos aislados cuenta con la ventaja de aportar contexto. Es más útil para un defensor conocer que cierta IP formó parte de un ataque DDoS que comenzó en una fecha dada a simplemente saber que dicha IP se encuentra catalogada como maliciosa. A modo de ejemplo se puede observar el template de objeto MISP para un ataque DDoS en el anexo 7.1.
Estos objetos pueden ser compartidos como cualquier otro atributo de MISP, por más que otras instancias MISP no cuenten con el template al cual se encuentra asociado.
A diferencia de los objetos MISP, otros estándares pueden llegar a ser muy estáticos al no permitir nuevas versiones de objetos ya existentes o la creación y distribución de los mismos por parte de los usuarios. Las amenazas cibernéticas, así como sus indicadores, tienen una naturaleza dinámica, por lo que nuevos indicadores pueden requerir un trabajo considerable para estandarizar. El objetivo de los objetos MISP es permitir la actualización dinámica de la definición de los mismos en sistemas de intercambio distribuidos.
Estos objetos permiten agregar nuevos formatos de indicadores combinados basados en su uso sin que sea necesario modificar el código base de MISP. La definición de los mismos puede ser propagada junto con sus indicadores.

2.3.4. MISP feeds

MISP puede obtener información sobre amenazas mediante los feeds[18]. Esta funcionalidad permite a MISP obtener eventos desde un servidor externo sin necesidad de un acuerdo previo. Pueden ser usados como fuente de correlaciones para todos los eventos y atributos propios sin la necesidad de que sean importados directamente en el sistema, por lo que permite correlaciones y comparaciones rápidas entre distintos feeds. Pueden encontrarse en tres formatos distintos: MISP, CSV y texto desestructurado, y pueden ser obtenidos mediante URLs o archivos locales.
La instalación estandar de MISP incluye un conjunto de feeds OSINT[3] por defecto, los cuales contienen información sobre dominios maliciosos, blacklists de IPs, URLs maliciosas, hashes maliciosos, etc.

2.4. Interfaces de comunicación ModSecurity-MISP

Los mecanismos y tipos de ataques cambian constantemente. No es suficiente tener un conjunto de aplicaciones web protegidas contra todas las amenazas de hoy bajo el mismo WAF si mañana los ataques mutan y las defensas quedan obsoletas. Es por

[3]Open Source Intelligence, práctica de obtener información desde fuentes públicas para ser utilizadas en un contexto de inteligencia

esto que se busca una integración entre ModSecurity y MISP de forma que MISP sea capaz de indicarle a ModSecurity si se detectó una nueva amenaza para que pueda mejorar su función como WAF y proteger a las aplicaciones web de forma más eficaz. Para esto es necesario evaluar las distintas interfaces de comunicación entre ambas tecnologías.

En la siguiente sección se explorarán los distintos métodos de comunicación de las plataformas. NOTA: no se considerarán los métodos manuales.

2.4.1. Comunicación con MISP

A continuación se plantean distintos métodos de comunicación con la plataforma MISP.

MISP API

MISP provee una API[19] HTTP para integrarla con soluciones personalizadas. Esta API permite automatizar la interacción con MISP, permitiendo manipular eventos, tags, atributos, avistamientos y otros recursos. Particularmente permite realizar búsquedas sobre los eventos/atributos con un alto nivel de granularidad. Por ejemplo, permite realizar una consulta para obtener todos los eventos con una IP de origen dada.
Actualmente, MISP incluye PyMISP[20], una librería escrita en python para obtener/agregar/actualizar atributos de eventos, así como manejar muestras de malware o buscar atributos. Es una interfaz para interactuar con la API.

Tomemos el siguiente flujo a modo de ejemplo: con cada nueva solicitud entrante, ModSecurity realiza una consulta contra MISP para consultar si algunos elementos de esa solicitud se encuentran marcados como maliciosos, y en caso de que así lo sean bloquear esa solicitud.
Esta API se destaca por las capacidades de filtros sobre eventos, atributos etc. En particular expone 2 endpoints para el filtrado:

- Filtrado de atributos: `POST /attributes/restSearch`
- Filtrado de eventos: `POST /events/restSearch`

Los filtros (la query) se envían en el cuerpo de la solicitud en formato JSON y permiten combinaciones de atributos mediante ANDs, ORs y NOTs.
Lista de filtros soportados relevantes:

- **returnFormat**: Permite seleccionar el formato de respuesta de la query, entre ellos json y xml
- **type**: tipo de atributo (ej. ip-src)
- **value**: buscar por el valor indicado en el campo `value` de un atributo

- **threat_level_id**: indica el nivel de amenaza
- **category**: categoría del atributo
- **from** y **to**: permiten filtrar eventos con fecha comprendida entre estos parámetros
- **withAttachments**: permite codificar los elementos adjuntos en base64 en el campo `data` de cada atributo
- **publish_timestamp**: restringe los resultados a aquellos publicados a partir de cierto timestamp
- **timestamp**: funciona igual que publish_timestamp pero usa el timestamp de la ultima edicion

MISP export

MISP permite exportar datos como sus eventos en distintos formatos, como STIX (XML, JSON, STIX 2.0), Open IOC, texto y otros formatos. La API también permite exportar datos en estos formatos.

MISP import

Además de permitir la exportación de datos, MISP permite importar datos en los formatos anteriormente mencionados. Al igual que con export, se pueden importar datos desde la API.

Módulos MISP

MISP permite extender sus funcionalidades mediante módulos[21] con el objetivo de facilitar la extensión de la plataforma sin modificar sus componentes centrales. Pueden ser ejecutados tanto en el mismo servidor MISP como en un servidor remoto. Existen tres tipos de módulos:

- Módulos de expansión: enriquecen datos que ya se encuentran en MISP
- Módulos de importación: importan datos a MISP
- Módulos de exportación: exportan datos desde MISP

Se puede interactuar con los módulos mediante una API REST HTTP accedida desde `http://misp_host/query`. El módulo accedido así como sus parámetros son especificados en el cuerpo de la solicitud.
A modo de ejemplo, el módulo DNS se encarga de traducir hostnames a direcciones IP. Puede ser invocado de la siguiente forma:

```
curl -s http://misp_host/query \
  -X POST \
  -H "Content-Type:application/json" \
  -data @body.json
```

body.json:

```
{ "module": "dns", "hostname": "www.circl.lu" }
```

A continuación un ejemplo de respuesta:

```
{
  "results":[
    {
      "values":[
        "149.13.33.14"
      ],
      "types":[
        "ip-src",
        "ip-dst"
      ]
    }
  ]
}
```

Todos estos módulos deben ser desarrollados en python. Para desarrollar un módulo se deben implementar al menos 3 funciones:

- introspection: función que devuelve un diccionario de los atributos soportados por el módulo
- handler: función que acepta un documento JSON para expandir sus valores y devolver un diccionario de los mismos
- version: función que devuelve un diccionario con la versión y los metadatos asociados, incluyendo configuraciones potencialmente requeridas por el módulo

2.4.2. Comunicación desde ModSecurity

A continuación se plantean distintos métodos de comunicación con ModSecurity.

Logs de ModSecurity

Una de las razones de la creación de ModSecurity fue la falta de visibilidad que proveían las herramientas existentes, es por esto que ModSecurity cuenta con amplias capacidades de logging y debugging[22][23][24].
Existen 2 tipos de logs:

- auditoría: por cada transacción bloqueada, ModSecurity provee logs detallados sobre la transacción, asi como el por qué del bloqueo. Es el log principal
- debug: provee información extensiva de todo lo que hace ModSecurity (en caso de que el modo debug se encuentre activo)

El log de auditoría loguea todos los ataques, así como todos los potenciales ataques que ocurren. Cada entrada del log se encuentra particionada en distintas secciones, lo cual facilita su escaneo para encontrar la información buscada. El cuadro 2.1 ilustra las secciones generadas.

Sección	Descripción
A	Header del log de audit (mandatorio)
B	Headers de la solicitud
C	Cuerpo de la solicitud
D	Reservado
E	Cuerpo de respuesta
F	Headers de respuesta
G	Reservado
H	Trailer del log de audit, contiene datos adicionales
I	Alternativa al punto C, cuerpo de la solicitud compacto, excluye archivos
J	Información sobre los archivos subidos
K	Contiene una lista de todas las reglas que matchean para la transacción
Z	Límite final

Cuadro 2.1: Secciones del log de auditoría

De esta forma, quedan separados elementos como los headers de una solicitud, el cuerpo de la solicitud, el cuerpo de la respuesta, los headers de la respuesta, etc.

Acción 'exec'

La acción *exec* es una de las acciones disponibles dentro de una regla (ver sección 2.1.1). Mediante esta acción ModSecurity puede ejecutar scripts externos a partir de la activación de una regla. Estos scripts tienen acceso a todas las variables del entorno, por lo que se emplea la directiva *setenv* para setear las variables de interés a partir de las variables de ModSecurity. Ej:

```
SecRule REQUEST_URI "@streq /index.php" "id:1, phase:1, t: lowercase, deny,
    setenv:bodypost=%{REQUEST_BODY}, exec:/scripts/script.sh"
```

Listing 2.3: Regla de ejemplo: exec

Esta regla es similar a la presentada en la sección 2.1.1, solo que además de rechazar la solicitud invocará a un script (/scripts/script.sh). Dicho script sólo podrá acceder a la variable `REQUEST_BODY` a través de la variable de entorno `bodypost`. El script debe imprimir algo en stdout, de lo contrario ModSecurity asumirá que el script falló. Exceptuando este detalle, *exec* no afecta la activación de una regla.

La aplicación de esta acción puede resultar en un fork del proceso, impactando negativamente en la performance.
Este compromiso en la performance puede ser evitado si se utiliza la regla SecRuleScript (detallada en la siguiente sección) junto con un script escrito en Lua[25].
Puede ser utilizado para obtener eventos de MISP a través de la API dentro del script.
También puede ser utilizado para popular MISP con eventos de ModSecurity, aunque esto puede resultar un poco tedioso debido a que se debería especificar explícitamente qué tipo de eventos y con qué características interesa registrar. Esto se puede hacer también a través de la API.

Regla SecRuleScript

La regla SecRuleScript sigue la siguiente sintaxis:

```
SecRuleScript "/path/to/script.lua" "[ACTIONS]"
```

Listing 2.4: Sintaxis de SecRuleScript

A diferencia de las reglas usuales (SecRule) en esta regla no hay ni *targets* ni *operators*, sino una ruta hacia un script. La regla delega la decisión sobre su activación al script, y éste puede obtener cualquier variable del contexto de ModSecurity y usar cualquier operador de Lua para evaluarlos. Se utiliza Lua ya que ModSecurity contiene un procesador de Lua.
El segundo parámetro `[ACTIONS]` es una lista de acciones, su significado es idéntico al significado de las acciones de una SecRule. A diferencia de la acción *exec*, es el script que decidirá si la regla es activada. Estos scripts son compilados en la etapa de configuración y son guardados en memoria, mejorando la performance ya que no genera un fork del proceso de ModSecurity (a diferencia de la acción `exec`). Esto presenta la desventaja de que si un script es modificado, se necesita reiniciar el servidor web sobre el que corre ModSecurity para que el cambio surta efecto.

Al igual que con `exec`, puede ser utilizado en los 2 sentidos (MISP→ModSecurity | ModSecurity→MISP).

Extensión nativa

Dado que ModSecurity forma parte de Apache, no necesita implementar su propio método de extender el lenguaje de reglas, sino que este es extensible mediante el desarrollo de módulos Apache en C. Existen 4 "puntos de extensión", permitiendo agregar variables, operadores, transformaciones y procesadores del cuerpo de la solicitud[26]. Cada punto de extensión es desarrollado dentro de su propio módulo, y puede ser extendido de la siguiente forma:

Transformaciones: para las transformaciones es necesario implementar una única función, la cual es invocada por ModSecurity. Por convención la función debe tener el mismo nombre que el nombre que se quiera usar para invocar la transformación desde ModSecurity. Entre los parámetros de la función se encuentran punteros apuntando

a los strings de input y output, por lo que se transforma el string proporcionado y se almacena el resultado en la memoria apuntada por el puntero de output.

Operadores: este punto puede resultar un poco más difícil de implementar que el anterior, ya que se necesitan dos funciones: además de una función como la que es necesario implementar para el caso de las transformaciones, hay una función adicional (opcional) a modo de inicialización, la cual se ejecuta en tiempo de configuración para preparar lo que necesite la función principal del operador y poder usarlo en tiempo de ejecución. Esto puede provocar mejoras significativas en la performance. A diferencia de las transformaciones, en los operadores se tiene acceso a las estructuras de ModSecurity directamente (en vez de trabajar con strings), lo que permite una mayor flexibilidad. Estas estructuras almacenan elementos como todos los datos de la transacción y la estructura de las variables. El resultado (si lo hubiese) es devuelto mediante un puntero presente en esa estructura.

Variables: para crear una nueva variable es necesario implementar una función donde se cree una nueva estructura del tipo variable de ModSecurity con los datos requeridos y luego almacenarla en la colección de variables.

Procesadores del cuerpo de la solicitud: todo procesador debe implementar 3 funciones: inicialización, procesamiento de datos y finalización. La inicialización y finalización son invocadas una única vez por cada solicitud, pero el procesamiento puede ser invocado repetidas veces, cada vez con un chunk del cuerpo.

En todos estos puntos planteados, además de las funciones mencionadas es necesario declarar ciertas estructuras y funciones para que el código sea reconocido como un módulo Apache.

Al ser una extensión nativa, esta estrategia es más performante que un script en Lua, a la vez que disminuye la complejidad de la arquitectura al contar con menos componentes. Por otro lado, el desarrollo puede resultar más engorroso si no se dispone de conocimientos previos sobre programación en Apache. A diferencia de Lua, es necesario compilar el código de la extensión.
Puede ser utilizado para la integración bidireccional de forma muy similar a `exec`.

Inclusión de archivo de reglas

Se pueden tener varios archivos de reglas independientes para luego ser incluidos en ModSecurity. Esto permite, por ejemplo, generar un archivo de reglas a partir de información sobre amenazas obtenidas de MISP y que éstas sean incluidas dinámicamente cada cierto intervalo de tiempo.
Hoy en día no es posible modificar reglas y que estas sean aplicadas sin tener que reiniciar el servidor web sobre el que corre ModSecurity, pero se está trabajando en solucionarlo[4].
Este método presenta el inconveniente de que no funciona en tiempo real.

[4]https://github.com/SpiderLabs/ModSecurity/issues/1970

2.5. Antecedentes

Actualmente existe un proyecto que logró integrar ModSecurity con MISP. En 2018 se creó el proyecto "Honeypot" de OWASP[27] con el objetivo de identificar ataques emergentes sobre aplicaciones web y generar reportes detallados de las amenazas para la comunidad, facilitando la protección ante estos ataques.
Para esto, se implementó una forma de comunicar ModSecurity con MISP de forma unidireccional (ModSecurity → MISP). Esto se logró enviando los logs de ModSecurity al stack ELK[28], con el propósito de filtrar los logs relevantes de los no relevantes. Este stack se encuentra compuesto por Elasticsearch, un potente motor de búsqueda, Logstash, un procesador y agregador de datos y Kibana, un visualizador de datos. Se utilizó Filebeat (o mlogc de forma alternativa con la misma finalidad) para parsear los datos de los logs y enviarlos a Logstash. No es necesario especificar cómo parsear los logs, basta con especificar que se encuentran en formato Apache y filebeat se encarga del resto.

Luego de haber populado el stack con los logs, la información relevante es procesada por un script python para obtener los datos de interés y ser enviada mediante PyMISP a la plataforma MISP.

Este proyecto fue desarrollado a modo de prueba de concepto pero plantea una arquitectura prometedora tal como se muestra en la figura 2.1. A pesar de eso no resuelve problemas fundamentales como el filtrado de los eventos relevantes de ModSecurity en base a su severidad.

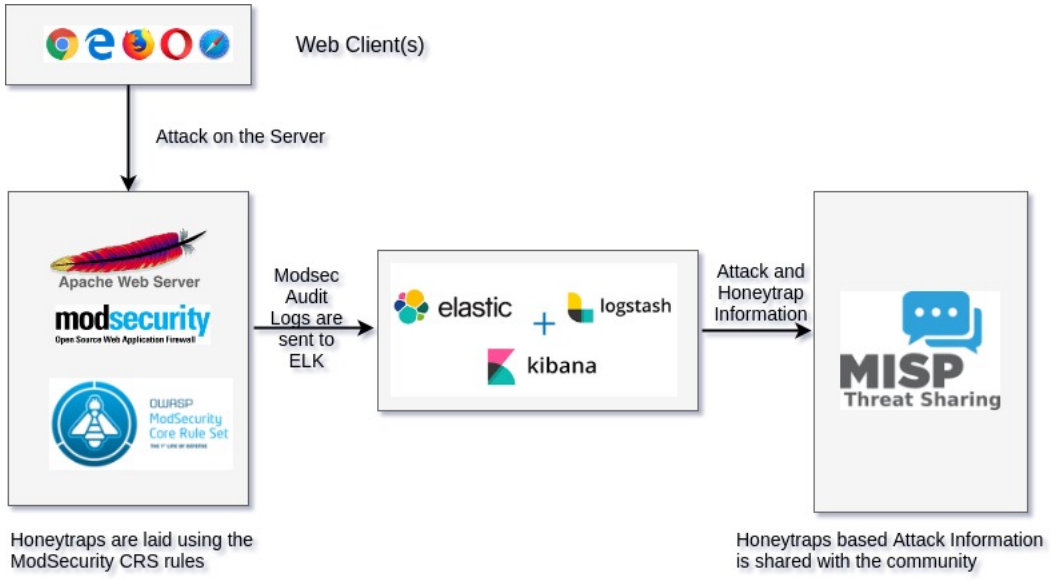

Figura 2.1: Diagrama arquitectura, obtenido del repositorio del proyecto[29]

Capítulo 3

Análisis

En este capítulo se detallan los requerimientos del proyecto y los casos de uso ante los cuales el sistema desarrollado debe responder correctamente. Partiendo de los requerimientos planteados se analizan los mecanismos de comunicación entre ModSecurity y MISP en base a las interfaces de comunicación planteadas en el estado del arte, para luego seleccionar el mecanismo que más se adapta a los requerimientos. También se analiza cómo identificar los eventos maliciosos relevantes en MISP. Finalmente se menciona cómo satisfacer cada caso de uso empleando el mecanismo de comunicación seleccionado.

3.1. Requerimientos

Luego de analizar el estado del arte se plantearon distintos requerimientos y se definió el alcance del proyecto. Se detallan los requerimientos a continuación.

3.1.1. Integración unidireccional MISP → ModSecurity

En el estado del arte se presentaron las interfaces de comunicación de ambas tecnologías, y debido a las capacidades de las interfaces de cada tecnología, se observó que es posible una integración bidireccional entre MISP y ModSecurity (ambas tecnologías permiten entrada y salida de datos). Aún así se decidió implementar la integración en un sólo sentido: MISP → ModSecurity. Esto se debe principalmente a que el objetivo (reforzar un WAF a partir de información obtenida mediante una plataforma TISP) puede ser alcanzado con la integración en el sentido elegido: ModSecurity puede ser reforzado por información obtenida desde MISP sin que este último sea alimentado por eventos de ModSecurity. Por otro lado, como se mencionó en 2.5, ya existe una integración de este tipo basándose en la estrategia de logs de ModSecurity.
Con una integración de este tipo (MISP → ModSecurity) se sacrifican algunas funcionalidades, como la posibilidad de alimentar a MISP con información relevante para la red monitoreada. A modo de ejemplo, en el caso de que un nodo de la red sea víctima de un ataque DoS dirigido de una IP no registrada por MISP y ModSecurity llegue a detectarlo, podría informar a los otros nodos sobre este ataque, permitiéndoles bloquear la IP ofensora antes de ser víctimas del mismo.
Con este cambio se dejan de lado algunas dificultades relacionadas a la integración en el sentido inverso, como la elección de qué tipos de eventos se quieren registrar en MISP.

3.1.2. Tiempo real / performance

ModSecurity debe poder obtener en tiempo real información sobre amenazas (en forma de eventos) desde MISP para incorporarla a su defensa. Es decir, ModSecurity debe poder comunicarse con MISP mediante algún medio para obtener la información necesaria sobre amenazas de tal forma que pueda utilizarla para identificar elementos maliciosos en solicitudes entrantes y así bloquearlas. En particular, debe poder obtener información sobre amenazas relacionadas a los casos de uso establecidos y poder reaccionar ante ellas (bloquearlas). Es por esto que la solución debe ser lo más performante posible, evitando retrasar significativamente la respuesta a una solicitud HTTP externa.

3.1.3. Extensible

La solución debe ser extensible, permitiendo soportar casos de uso distintos a los planteados con mínimas modificaciones. Idealmente se deben poder soportar nuevos casos de uso sin la necesidad de modificar el código existente.

3.1.4. Parametrizable

La solución debe ser parametrizable: se debe poder configurar la solución implementada sin necesidad de modificar el código. En particular se debe poder configurar la conexión al servidor MISP (credenciales y URL), idealmente se debe poder configurar cada caso de uso de forma independiente.

3.2. Casos de uso

Se plantean algunos casos de uso para los que el sistema debe responder correctamente, es decir, recuperar la información de interés en función del caso de uso desde MISP y bloquear las solicitud catalogadas como maliciosas a partir de la información obtenida.

3.2.1. Blacklistear conjunto de IPs

ModSecurity debe poder bloquear solicitudes provenientes de IPs catalogadas como maliciosas a partir de información obtenida de MISP.
Consideramos que una IP es “maliciosa” cuando se encuentra asociada a un evento MISP como IP de origen.

3.2.2. Subida/descarga de archivos maliciosos con firma conocida

ModSecurity debe poder bloquear solicitudes que contengan adjuntos archivos con firma conocida y catalogada como maliciosa o que intenten acceder a archivos con esas mismas características. En el caso de la descarga sólo deberán ser considerados archivos que se encuentren bajo determinada URL por motivos de performance.
Consideramos que una firma asociada es maliciosa cuando existe un evento MISP que contenga un atributo de tipo hash y con el mismo valor (o similar en caso de fuzzy hashes) que el hash del archivo adjunto.

3.2.3. URLs maliciosas

ModSecurity debe poder bloquear toda solicitud cuya URL contenga algún elemento malicioso, ya sea la ruta o los parámetros presentes en la query string. Al igual que en los casos de uso anteriores, consideraremos que un elemento de la URL es malicioso si existe un evento en MISP que contenga un atributo de tipo URL/URI con los mismos elementos.

En todos los casos planteados se puede obtener la información (IPs, firmas, URLs) de los eventos MISP y particularmente de los objetos MISP. Como se detalló en 2.3.3, estos objetos se encuentran compuestos por atributos y cuentan con la ventaja de que tienen su estructura bien definida. También se pueden encontrar a estos atributos "sueltos" (sin información extra asociada), aunque estos no proporcionan un contexto y puede provocar el bloqueo de falsos positivos.

3.3. Estrategias de integración

A continuación se evaluan distintas estrategias para integrar la instancia ModSecurity con la instancia MISP. En particular se evaluarán 3 de las estrategias planteadas en 2.4.2: reglas nativas, extensión de ModSecurity mediante scripts y extensión de ModSecurity nativa. La estrategia de logs es útil exclusivamente para exportar datos, por lo que no será tenida en cuenta.

3.3.1. Reglas nativas

Conociendo los casos de uso con anticipación, se puede generar manualmente un archivo de reglas para contemplarlos. Por ejemplo, ModSecurity provee distintas reglas para comparar atributos de una solicitud (IP, URL, etc.) contra un archivo de texto para realizar alguna acción como capturar la solicitud. Estas reglas pueden ser almacenadas en un servidor central y ser consumidas mediante `SecRemoteRules`[30], una directiva de ModSecurity que permite cargar reglas desde un archivo alojado en un servidor remoto.
Presenta la ventaja de que no es necesario alterar o extender componentes de ModSe-

curity. Por otro lado, se necesita que esos archivos sean constantemente actualizados por un agente externo a ModSecurity/MISP, impactando en la performance negativamente. La principal desventaja es que por más que esta estrategia se aproxime a ser de tiempo real, no llega a serlo. Es por no cumplir este requerimiento que no se profundizará sobre esta estrategia en este análisis.

3.3.2. Extensión de ModSecurity

A continuación se describen los distintos métodos existentes para extender ModSecurity.

Mediante scripts (Lua)

Como se mencionó en 2.4.2, ModSecurity permite escribir reglas en Lua, un lenguaje de scripting rápido y eficiente en términos de memoria.
Se puede utilizar de 2 formas: escribiendo reglas (utilizando la directiva `SecRuleScript`) y escribiendo scripts que sean ejecutados ante el match de una regla (`exec`). Dado que lo que nos interesa es poder decidir sobre la activación de una regla, no consideraremos `exec`.

Se necesitan crear tantos scripts como casos de uso se quieran soportar, dado que no hay forma de indicarle al script qué caso de uso se encuentra siendo analizado. Cada script es el encargado de comunicarse con la instancia MISP para obtener información sobre solicitudes maliciosas.

Como se mencionó en 2.4.2, ModSecurity incluye un procesador de Lua. Por más que Lua sea un lenguaje de scripting, el procesador de Lua de ModSecurity provoca que todos los scripts de Lua sean compilados en tiempo de configuración y cacheados en memoria, por lo que no se incurre en una penalización al interpretarlo. Por otro lado, al ser escrito en C nunca será más rápido que C, siendo preferible la extensión nativa en lo que respecta a la performance.

Con respecto a la utilización de Lua en ModSecurity, a pesar de que éste lenguaje ha sido utilizado en producción por distintas organizaciones a lo largo de los años, aún se encuentra marcado como experimental.[31]

Extensión nativa

Como se mencionó en 2.4.2, esta estrategia es la más eficiente ya que no es necesario invocar ningún proceso externo a ModSecurity (sin tener en cuenta la API de MISP) y el código es compilado a lenguaje de máquina y no interpretado. Al ser un módulo Apache se cuenta con acceso a las librerías APR y APR-Util[32], las que contienen funcionalidades tales como acceso a sockets de red y protocolos, pools de memoria, threads, etc.

ModSecurity puede ser extendido mediante la implementación de transformacio-

nes, operadores, variables y procesadores del cuerpo de la solicitud. Se puede implementar un único operador de ModSecurity el cual se encargue de comunicarse con la instancia MISP para determinar si se debe o no bloquear determinada solicitud entrante.

A dicho operador se le debe proporcionar por parámetro un identificador del caso de uso ("blacklist_ip", "malicious_file", "malicious_url"). Se deben definir tantas reglas como casos de uso se quieran tener en cuenta, donde cada regla debe tener un identificador de caso de uso distinto y variables/transformaciones acordes.

3.3.3. Conclusiones

Tanto la estrategia de extender ModSecurity mediante scripts como la de extenderlo de forma nativa satisfacen todos los requerimientos planteados. Ambas pueden acceder a todas las variables de ModSecurity, además de proporcionar la habilidad de poder programar las reglas (a diferencia de la estrategia de reglas nativas). Se diferencian en la forma en que implementan las reglas: en ambas se necesitan escribir tantas reglas como casos de uso se deseen soportar, pero la extensión nativa los diferenciará mediante parámetros del operador y la extensión mediante scripts los diferenciará con distintos scripts.

Con respecto al lenguaje, la extensión nativa debe ser implementada en C, mientras que la de scripts en Lua, un lenguaje más accesible. Dado que ModSecurity se encuentra escrito en C, una integración nativa simplificaría la estructura del código.

En cuanto a la performance, los scripts escritos en un lenguaje distinto a Lua serán más lentos que la extensión nativa y los scripts en Lua, y la extensión nativa será tanto o más rápida que los scripts en Lua.

Es por estos motivos que que la extensión nativa es una mejor opción que la extensión mediante scripts, ya que ambas pueden satisfacer todos los casos de uso pero la nativa destacará en el requerimiento de tiempo real, permitiendo centralizar el código en un único módulo C y simplificando la arquitectura.

3.4. Identificación de eventos en MISP

Se pueden utilizar los feeds MISP para obtener información sobre amenazas, y luego aplicar un filtro para descartar eventos no relevantes. Llamamos "no relevantes" a eventos que no se corresponden con ninguno de los casos de uso analizados. Esto presenta el problema de cómo diferenciar los eventos relevantes de los no relevantes. Como solución ingenua se puede descartar los objetos que no contengan atributos relevantes, como puede serlo `ip-src` en el caso de blacklistear lista de IPs. Esto presenta el inconveniente de generar un gran número de falsos positivos, ya que la presencia de ese atributo no indica necesariamente que la IP sea maliciosa.
Para solucionar el problema se pueden filtrar los eventos en función de sus objetos y no de sus atributos, en particular en función del template de objeto asociado a esos

objetos.

3.4.1. Templates de objeto

Para identificar qué eventos de MISP son relevantes para nuestros casos de uso se pueden utilizar templates de objeto previamente definidos. Al momento de filtrar los eventos en MISP se pueden descartar los eventos que no contengan objetos asociados a los templates que consideremos relevantes.

Se puede emplear un único template genérico (ej. ataque a aplicaciones web) o un template distinto por caso de uso.

Template único

En este caso se tiene un único template el cual contiene todos los atributos relacionados a un ataque a aplicaciones web (ip de origen, url, método HTTP, headers HTTP, etc.). El principal desafío es cómo distinguir el caso de uso dado un objeto asociado a ese template.
Se puede evaluar qué atributos se encuentran presentes en el objeto para identificar el caso de uso. Por ejemplo, si el atributo `ip-src` se encuentra presente entonces sabemos que el evento se corresponde al caso de uso de blacklistear IPs. Es responsabilidad del equipo que genere los eventos no generar eventos con el atributo `ip-src` si el mismo no se corresponde al caso de uso de blacklistear IPs. La ventaja de este template único es que no se deben definir casos de uso a priori: el template cubre la gran mayoría de los casos de uso de ataques a aplicaciones web.

La principal desventaja es su falta de flexibilidad. Si en un futuro se decide incorporar nuevos atributos al template se debe generar un nuevo template. En caso de sustituir el viejo template por el nuevo se deben modificar todas las referencias al mismo. Eso provoca que se ignoren los eventos con objetos asociados al viejo template. También se puede dar soporte a ambos templates, pero entonces deja de ser un template único.

Dado que ciertos atributos son exclusivos de determinados casos de uso, este enfoque no permite agregar los mismos a los otros objetos. Por ejemplo, para el caso de uso de un archivo malicioso adjunto en la solicitud, puede ser relevante registrar la IP de donde se originó la solicitud para poder correlacionarla con otros eventos de la comunidad MISP. El problema es que el atributo `ip-src` se encuentra reservado para el caso de uso de blacklistear IPs, por lo que no se podría usar. Como alternativa se puede establecer una precedencia entre los atributos, como por ejemplo `ip-src` < `MD5`. De esta forma cualquier objeto con un atributo del tipo `MD5` presente es asociado al caso de uso de blacklistear IPs, tenga o no el atributo `ip-src` presente.

Este template debe estar compuesto por (al menos) los siguientes atributos:

- `ip-src`
- `url`

- `http-method`
- `user-agent`
- `cookie`
- atributos relacionados a archivos adjuntos:
  - `pattern-in-file`: patrón en el archivo que identifica el malware
  - `pattern-filename`: patrón en el nombre de archivo
  - `attachment`: binario del archivo
  - hashes: solo los soportados por ModSecurity (`MD5`, `sha1`, `ssdeep`)

Se puede seleccionar un único tipo de hash y descartar los otros para solo tener que soportar el hash elegido. Esto evita que ModSecurity deba realizar transformaciones extra sobre los archivos adjuntos y perjudique la performance. En la sección 3.6.3 se profundizará sobre este tema.

Un template por caso de uso

Como alternativa al template único se puede tener un template para cada caso de uso. Por ejemplo, para el caso de blacklistear IP se puede tener un template con un único atributo `ip-src`. Con este enfoque no es necesario inspeccionar los atributos del objeto para poder identificar el caso de uso, ya que el mismo template lo identifica. Esto permite poder agregar más atributos de los estrictamente necesarios, permitiendo agregar información adicional al evento que permita correlacionarlo con otros eventos, a diferencia de lo planteado en el caso de un único template.
La principal desventaja es que se deben conocer todos los casos de uso de interés para poder definir los templates.

Para cada caso de uso se deben crear distintos templates con los siguientes atributos:

Atributos del template "blacklistear un conjunto de IPs":

- `ip-src`

donde `ip-src` es requerido.

Atributos del template "subida de archivos maliciosos con firma conocida":

- `pattern-in-file`
- `pattern-filename`
- `attachment`
- `MD5`

- `sha1`
- `ssdeep`

donde al menos un atributo de los listados es requerido.

Atributos del template "URLs maliciosas":

- `url`

donde `url` es requerido.

Conclusión

Un template por caso de uso permite un control más granular de los casos de uso y a diferencia del template único no es necesario un procesamiento extra del objeto para determinar el caso de uso asociado, además de permitir enriquecer los eventos correlacionándolos con otros. Es por esto que preferimos este tipo de templates sobre el template único.

3.5. Comunicación con MISP

En cada solicitud entrante, ModSecurity se comunica con la instancia MISP para comparar los atributos relevantes de la solicitud contra los atributos de los eventos relevantes de MISP. Como ya se mencionó en 2.4.1, MISP provee una API para integrarla con soluciones personalizadas. Esta API permite, entre otras cosas, listar eventos y atributos. Se puede utilizar dicha API para obtener toda la información necesaria de MISP y a partir de la misma capturar la solicitud.
Mediante la API mencionada se pueden filtrar eventos por sus atributos, ya sean atributos directos del evento o atributos de un objeto asociado al mismo. Sin embargo, no se puede filtrar en función del template de objeto, es por esto que ese filtrado debe realizarse por fuera de la API. Puede realizarse tanto del lado de MISP mediante módulos MISP como de ModSecurity. Como ya se mencionó en 2.4.1, estos módulos permiten expandir las funcionalidades de MISP y la comunicación con los mismos ocurre mediante una API REST.

En el caso de no utilizar un módulo MISP, ModSecurity se comunicará directamente con la API MISP mediante el lenguaje de queries. También debe almacenar el/los UUIDs de el/los templates de objeto. Esto simplifica la arquitectura al prescindir de un componente, pero vuelve a ModSecurity más complejo añadiéndole elementos del dominio de MISP (UUIDs, queries y manejo de la respuesta). En caso de que sea necesario actualizar cualquiera de esos elementos es necesario reiniciar el servidor para que los cambios surtan efecto.

Implementar un módulo MISP permite centralizar todo el procesamiento de los eventos en un único lugar, además de abstraer a ModSecurity del lenguaje de con-

sultas que utiliza MISP y centralizar la lógica. Esto permite definir los templates de interés así como todas las consultas al core de MISP del lado de MISP. Cualquier modificación de los templates o de las consultas no impacta directamente a ModSecurity, evitando modificar su configuración y reiniciar las instancias, además de tener que realizar una única modificación sobre el módulo MISP en vez de una modificación por cada instancia de ModSecurity.
Los UUIDs de los templates pueden ser abstraídos del código del módulo para ser cargados en tiempo de configuración por MISP mediante variables de ambiente.

Es por estos motivos que la comunicación a través del módulo MISP es la mejor opción.

3.6. Variables de ModSecurity involucradas

En cada caso de uso, el operador de ModSecurity obtiene la informacion necesaria de la solicitud entrante mediante las variables proporcionadas por ModSecurity. En las siguientes secciones se analiza cuales variables son necesarias para satisfacer cada caso de uso.

3.6.1. Blacklistear conjunto de IPs

En este caso la variable de interés es `REMOTE_ADDR`, la cual almacena la dirección IP del cliente remoto que originó la solicitud.

3.6.2. URLs maliciosas

En ModSecurity existen distintas variables que referencian a la URL accedida o a parte de ella: `REQUEST_URI`, `REQUEST_FILENAME` y `REQUEST_BASENAME`.

- `REQUEST_URI`: almacena la URL completa de la solicitud, incluyendo la query string pero sin incluir dominio ni protocolo. Ej: `/app/index.php?p=X`
- `REQUEST_FILENAME`: igual que `REQUEST_URI` pero sin incluir la query string. Ej: `/app/index.php`
- `REQUEST_BASENAME`: almacena solo el nombre del archivo de `REQUEST_FILENAME`. Ej: `index.php`

La regla utiliza la variable `REQUEST_URI`. `REQUEST_FILENAME` y `REQUEST_BASENAME` fueron descartadas por no incluir la query string.

3.6.3. Subida/descarga de archivos maliciosos con firma conocida

Los archivos adjuntos a una solicitud son accedidos desde la variable de ModSecurity `FILES_TMP_CONTENT`. Dicha variable guarda una colección que almacena el contenido de los archivos. El contenido de la respuesta HTTP es accedido desde la variable `RESPONSE_BODY`.
Para obtener la firma es necesario aplicar alguna transformación de ModSecurity: `MD5`, `SHA-1`. El hash `ssdeep` se puede obtener mediante un operador.

A diferencia de los hashes clásicos como SHA-1 y MD5, existen los llamados fuzzy hashes como ssdeep. Estos hashes sirven para comparar archivos similares no necesariamente idénticos. Muchas veces los atacantes modifican levemente su malware, por lo que no se detectaría el malware modificado empleando el método de hashing clásico. Con fuzzy hashing se obtiene cierto valor de similitud comparando los hashes, y si ese valor excede determinado threshold se asume que ambos archivos forman parte de la misma familia de malware.
MISP soporta los algoritmos `ssdeep`[33] y `tlsh`[34].

Como se mencionó previamente, se puede conservar un único tipo de hash y descartar los demás. Idealmente se emplearía un fuzzy hash debido a que es más robusto que los hashes clásicos: es posible detectar la amenaza aún cuando un atacante modifique el archivo malicioso conocido. ModSecurity soporta el hash `ssdeep`, pero a diferencia de `MD5` y `SHA-1` no se encuentra disponible como una transformación sino como un operador: al mismo se le proporciona el threshold objetivo y la ruta a un archivo conteniendo una base de datos de hashes `ssdeep`. El operador compara el hash del archivo contra los hashes de la base de datos y en caso de encontrar un hash similar en base al threshold proporcionado activa la regla.
Debido al funcionamiento del operador no hay forma de que se comunique con MISP en tiempo real y no se adapta a la solución planteada, debiendo contemplar este caso de uso con una solución aparte. Se evaluó la implementación de una transformación para obtener el hash `ssdeep` y poder integrarlo a la solución planteada, pero se descartó la idea por complejizar el desarrollo y exceder los objetivos planteados.
Se optó por conservar únicamente el hash MD5, ya que es el más rápido de los hashes presentados. Si bien no es un hash adecuado para aplicaciones criptográficas es útil para checksums.

Capítulo 4

Diseño

En este capítulo se plantea la arquitectura general del sistema así como la arquitectura del módulo MISP desarrollado. También se detalla cómo transcurre la comunicación entre los distintos componentes implementados.
Se detalla el diseño del operador de ModSecurity, de ahora en más `@MISP`, mencionando entre otras cosas la sintaxis/semántica de sus parámetros y la estructura del archivo de configuración. Finalmente se definen los templates de objeto a utilizar.

4.1. Arquitectura general y flujo

La arquitectura general de la solución desarrollada se compone de 3 componentes: la instancia de ModSecurity, el core de MISP y el módulo MISP "modsec", el cual funciona como interface entre la instancia de ModSecurity y el core de MISP. Tanto el core de MISP como el módulo MISP se encuentran contenidos dentro de la instancia MISP. La comunicación entre los distintos componentes ocurre enteramente sobre HTTP.
En el diagrama de la figura 4.1 se puede visualizar la arquitectura. Los bloques representan las instancias y los módulos involucrados, las flechas representan la comunicación HTTP entre ellas/os. Tanto el módulo MISP como el MISP core se encuentran dentro de la misma instancia MISP: esta es la configuración más usual pero podrían encontrarse en instancias distintas.

La comunicación ocurre entre el operador @MISP y el módulo modsec y entre el módulo modsec y el core de MISP. La misma comienza cuando una regla de ModSecurity conteniendo el operador @MISP es activada a raíz de una solicitud HTTP entrante.
En cada caso de uso, el operador @MISP envía al módulo MISP el identificador del caso de uso así como la información necesaria (obtenida de la variable de la regla) para filtrar los eventos MISP. El módulo se encarga de comunicarse con el core de MISP para obtener información sobre los eventos con las características proporcionadas por @MISP, para luego descartar los eventos irrelevantes, es decir, aquellos sin objetos asociados a un template de interés (el template correspondiente al caso de uso analizado). El flujo es representado por el diagrama en la figura 4.2, donde se describen los pasos con un mayor nivel de detalle.

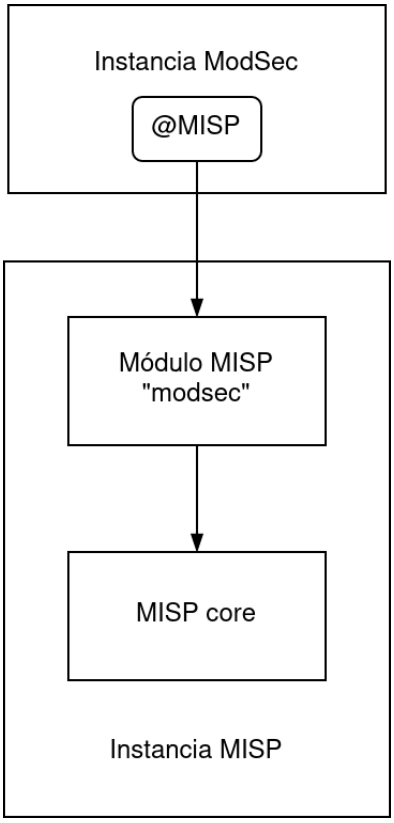

Figura 4.1: Diagrama arquitectura

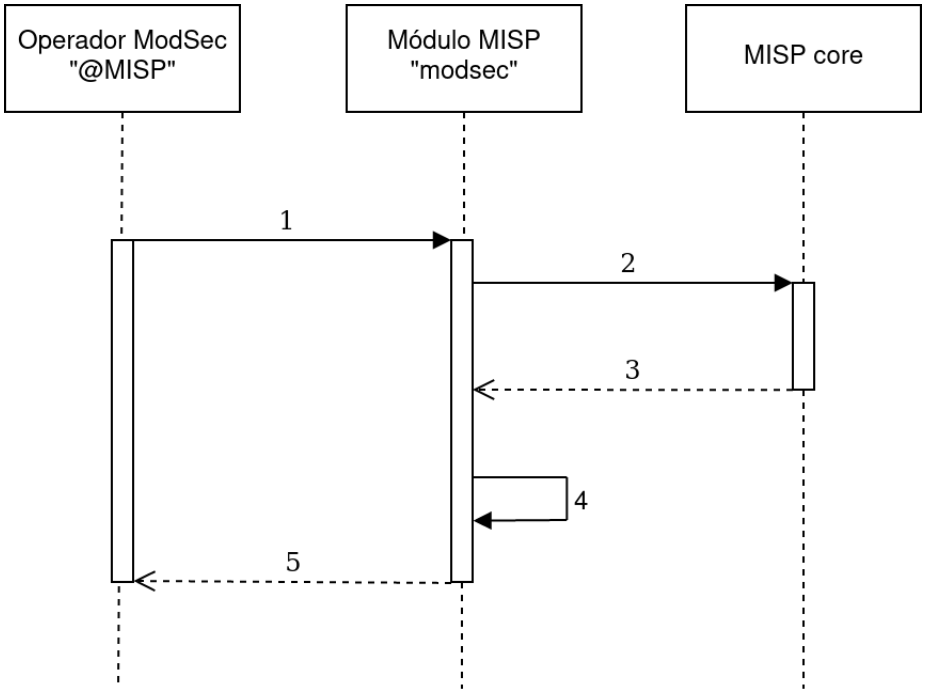

Figura 4.2: Diagrama flujo

1. El operador @MISP realiza una consulta al módulo MISP "modsec" mediante el endpoint `POST /query`, enviándole el identificador del caso de uso (en esta versión "blacklist_ip", "malicious_file", "malicious_url") así como los datos de interés (en esta versión IP, hash MD5 o URL)

2. El módulo MISP realiza una consulta al core de MISP en función del caso de uso y los datos proporcionados mediante la API MISP para obtener los eventos que contengan alguno de los datos de interés proporcionados por ModSecurity

3. MISP responde con los eventos que cumplan las características especificadas en el paso anterior

4. En caso de que la respuesta contenga al menos un evento, se filtran los mismos para eliminar los eventos que no contengan objetos asociados a el/los templates de interés (como se mencionó en 3.5 no se pueden filtrar por template de objeto utilizando la API). Los UUIDs de los templates de interés se encuentran definidos en el módulo MISP y discriminados por el identificador del caso de uso

5. El módulo MISP responde si encontró o no algún evento con las características indicadas

En caso de que se haya encontrado algún evento con las caracteristicas indicadas, se ejecuta la acción especificada en la regla que provocó el flujo.

En las siguientes secciones se describe de qué forma sucede comunicación entre el operador @MISP y el módulo "modsec" para cada caso de uso, así como el procesamiento que debe realizar el módulo "modsec".

4.1.1. Blacklistear conjunto de IPs

El operador @MISP envía una solicitud al módulo MISP con el identificador del caso de uso y la dirección IP asociada a la solicitud entrante, la cual es obtenida desde la variable `REMOTE_ADDR`. Luego el módulo MISP realiza una consulta al core de MISP (mediante el lenguaje de queries) para chequear si existe algún evento asociado a esa IP de origen, en particular chequea que no exista ningún evento con la IP proporcionada en el atributo `ip-src`. Esto se logra mediante el endpoint `/attributes/restSearch` proporcionado por el core de MISP.

4.1.2. URLs maliciosas

El operador @MISP envía una solicitud al módulo MISP con el identificador del caso de uso y la información sobre la URL accedida, la cual es obtenida desde la variable `REQUEST_URI`.

El módulo MISP se comunica con el core de MISP de forma análoga a lo planteado en el caso de uso anterior. En esta comunicación se pueden filtrar atributos por tipos

`url` y `uri`. No existen tipos de atributo que referencien parámetros de una query dentro de una URL. Para filtrar parámetros se utiliza el wildcard "%"[35] que equivale al regex ".*". A modo de ejemplo, para buscar el parámetro `paramMalicioso` se puede filtrar por `%&paramMalicioso=% || %?paramMalicioso=%`. El módulo MISP extrae estos parámetros de la URL para realizar dichas consultas.

4.1.3. Subida/descarga de archivos maliciosos con firma conocida

El operador @MISP envía una solicitud al módulo MISP con el identificador del caso de uso y el hash MD5 del archivo: para el caso de subida se envía el MD5 del archivo adjunto en la solicitud, el archivo es obtenido mediante la variable `FILES_TMP_CONTENT`. En el caso en que se adjunte más de un archivo se realizan tantas solicitudes como archivos haya: este comportamiento no es definido en el operador, sino que ModSecurity realiza una invocación al operador por cada archivo adjunto. Para el caso de bajada el archivo es obtenido mediante la variable `RESPONSE_BODY`. En ambos casos se utiliza primero la transformación "md5" para obtener el hash del archivo en formato binario y luego la transformación "hexEncode" para obtener la representación hexadecimal en ASCII del mismo, esto se debe a que el formato JSON en el que se van a codificar los datos para enviarlos al servidor de módulos MISP no soporta datos binarios.

El funcionamiento del módulo es análogo a lo planteado en casos anteriores (en la comunicación con el core de MISP se puede filtrar los atributos por los tipos de hash).

4.2. Instancia ModSecurity

4.2.1. Operador "@MISP"

El operador `@MISP` es el encargado de comunicarse con el módulo MISP para identificar si la solicitud entrante es identificada como maliciosa o no. Requiere dos parámetros: el primero es un string que identifica el caso de uso mientras que el segundo es un string indicando la ruta absoluta al archivo de configuración para ese caso de uso.
La lista de posibles valores del primer parámetro es obtenida dinámicamente desde el submódulo MISP, en particular para esta prueba de concepto puede adquirir los valores `blacklist_ip`, `malicious_file` y `malicious_url`. El operador arroja un error en caso de que no se proporcione un parámetro o el valor proporcionado no se encuentre entre los valores válidos. Este chequeo es realizado en la etapa de configuración del operador.

El operador espera distinto input en función del caso de uso. En particular para los siguientes IDs de caso de uso espera:

- `blacklist_ip`: una dirección IP
- `malicious_file`: un hash MD5
- `malicious_url`: una URL

Como se mencionó en 2.4.2, para implementar un operador personalizado es necesario implementar un módulo apache.
Cada operador de ModSecurity implementa las siguientes funciones:

- función inicializadora: se encarga de chequear la definición de la regla junto con sus parámetros y transformarlos en caso de ser necesario. Debe ser nombrada como `op_misp_init`
- función ejecutora: se encarga del procesamiento propiamente dicho del operador. Debe ser nombrada como `op_misp_exec`
- funciones registradoras: se encargan de registrar el nuevo operador a ModSecurity para que pueda ser utilizado en cualquier regla. Para esto es necesaria la implementación de las funciones `hook_pre_config` y `register_hooks`

A continuación se detalla cada función.

`op_misp_init`

Función inicializadora del operador. Es invocada una única vez por cada regla que utilice el operador `@MISP`. En ella se chequea la conexión con el módulo MISP (utilizando los datos del archivo de configuración, ver 4.2.3) y el parámetro proporcionado para verificar que exista y que sea válido. Estos controles se realizan en la etapa de configuración, es decir, cuando ModSecurity es iniciado o reiniciado.
Luego de chequear la conexión se verifica que el servidor de módulos MISP soporte el módulo `modsec` y el submódulo especificado en el parámetro de la regla. Esto es logrado obteniendo la lista de los módulos/submódulos desde el servicio `GET /modules`. En caso de que ocurra algún error se aborta la ejecución y se devuelve un mensaje de error descriptivo.

`op_misp_exec`

Esta función es la encargada de generar la query y realizar la consulta al módulo MISP, para luego activar la regla en caso de que se encuentren elementos maliciosos. En caso de que MISP indique que existen eventos maliciosos con las características proporcionadas, esta función activa la regla y genera un log indicando la razón. En caso de que no se pueda establecer una conexión con MISP se genera un error descriptivo de forma similar a la función `op_misp_init`.

`hook_pre_config`

Esta función es invocada con cada reconfiguración de Apache (con cada restart). Es la encargada de registrar el nuevo operador para que pueda ser utilizado por ModSecurity mediante la invocación a la función `modsec_register_operator`, la cual

provee ModSecurity. En ella se define el nombre del operador, qué función se utilizará para inicializarlo y qué función se encargará del procesamiento.

`register_hooks`
Esta función es la encargada de registrar el hook de apache `hook_pre_config`. Esto es necesario para que el hook sea ejecutado con cada reconfiguración de Apache. Esta función es referenciada en el momento de definir el módulo de Apache.

4.2.2. Comunicación HTTP

Como ya fue mencionado, en cada solicitud HTTP entrante es necesario realizar (al menos) una solicitud HTTP al servidor de módulos MISP. Si bien Apache cuenta con la capacidad para realizar solicitudes HTTP (ver mod_proxy_http[36]), no utiliza ninguna librería externa sino que el código para realizar las solicitudes se encuentra autocontenido en el código de Apache. Es por esta razón que se decidió incluir una librería para ese propósito. Se analizaron únicamente librerías no deprecadas escritas enteramente en C que ofrezcan al menos un cliente básico HTTP(s) y que no dependan de otras librerías.

`Curl`
Curl[37] es una de las librerías HTTP más conocidas y soporta un amplio rango de protocolos (HTTP, FTP, TELNET, etc.). Fue descartada por ser una de las librerías más pesadas.

`Serf`
La librería Serf[38] es una librería que provee un cliente HTTP de alta performance construida sobre la librería APR de apache. A primera vista parece ser la librería más adecuada para este caso de uso ya que fue construida sobre una dependencia de Apache (APR) y actualmente es mantenida por la propia organización Apache, pero fue descartada por prácticamente no contar con documentación.

`Neon`
Neon[39] es un cliente HTTP/1.1 y WebDAV. Junto con Serf, es una de las librerías usadas en el proyecto Subversion. Fue elegida por su pequeño tamaño, su performance[40] y su buena documentación.

4.2.3. Configuración

Al operador se le proporcionan parámetros para su configuración, particularmente los parámetros de conexión con el servidor de módulos MISP, tales como el host y el puerto, además de la clave privada para poder interactuar con el core de MISP mediante su API rest. Notar que el operador no interactúa directamente con la API MISP sino a través del servidor de módulos MISP. En el caso que la API KEY estuviera almacenada en el servidor de módulos MISP se debería contar con algún tipo de autenticación extra de la instancia ModSecurity que origina las solicitudes. Al almacenar la key del lado de ModSecurity la autenticación es enforzada por el

servidor MISP, permitiendo manejar permisos de forma granular desde la interfaz web que proveé (o desde la API en su defecto). Esto se debe a que la key viaja desde ModSecurity hasta el servidor de módulos MISP, y de este servidor hacia el servidor MISP, donde es evaluada.
Estos parámetros son definidos en un archivo de configuración en texto plano donde cada linea del archivo sigue el formato *KEY=VALUE* y *KEY* puede adquirir los valores "MISP_MODS_HOST", "MISP_MODS_PORT" y "MISP_API_KEY".
Este archivo de configuración es almacenado en el sistema de archivos sobre el que se ejecuta el servidor de ModSecurity, y la ruta a este archivo es proporcionada en cada regla que utilice el operador MISP. Notar que esto permite definir distintas configuraciones para los distintos casos de uso.

4.3. Módulo MISP

En esta sección se detallan distintos aspectos del módulo MISP desarrollado, en particular se detalla su arquitectura tipo plugin y la interfaz que expone para la comunicación con ModSecurity.

4.3.1. Arquitectura

El módulo sigue una arquitectura de tipo plugin: un componente central en el cual se definen todas las funciones necesarias para que funcione como un módulo MISP (ver 4.3.3) y un componente (plugin) por caso de uso. Nos referiremos a estos últimos componentes como submódulos. Esto permite agregar componentes sin que sea necesario modificar el código existente. Puede verse ilustrado en la figura 4.3, en la cual se ilustra el módulo "modsec" y sus submódulos. La comunicación entre el módulo y sus submódulos ocurre internamente: el módulo importa dinámicamente el código del submódulo necesario en función del caso de uso. Es esta importación dinámica uno de los principales puntos responsables de la extensibilidad del sistema, ya que permite agregar nuevos submódulos sin la necesidad de modificar el código existente.
El nombre del archivo de cada submódulo (sin tener en cuenta la extensión) debe ser igual al identificador del caso de uso para facilitar su listado.
El componente central lista todos los demás componentes, de esta forma puede obtener una lista de los casos de uso soportados. Delega el procesamiento de cada solicitud al componente correspondiente.

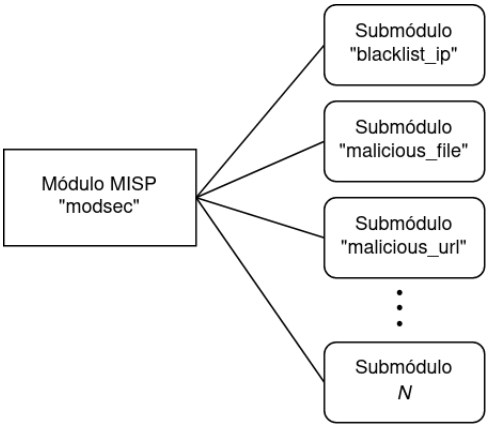

Figura 4.3: Diagrama arquitectura módulo MISP

4.3.2. Comunicación

Como todo módulo MISP, este módulo es contactado mediante una comunicación HTTP (ver 2.4.1).
MISP expone dos endpoints para la comunicación con los módulos: `GET /modules` para obtener información de los mismos y `POST /query` para ejecutar una consulta sobre el módulo especificado en el payload.

En `GET /modules` MISP retorna información sobre los módulos, tales como atributos de input/output así como metadatos del mismo, tales como versión del módulo, descripción, etc. El módulo desarrollado devuelve un diccionario con los atributos soportados, tanto de input como de output. El resultado contiene los siguientes elementos:

- `input`: lista de atributos de input soportados (`use_case_id` y `data`)
- `output`: lista de atributos de output (`malicious`)
- `available_use_cases`: lista de IDs de casos de uso disponibles (en esta versión `blacklist_ip`, `malicious_file` y `malicious_url`)

Una consulta al módulo ModSecurity (mediante `POST /query`) contiene los siguientes atributos:

- `module`: string, debe tener el valor "modsec". Esto le indica a MISP qué módulo debe procesar la solicitud
- `use_case_id`: string, debe tener el ID del caso de uso seleccionado. En esta versión puede adquirir tres posibles valores: `blacklist_ip`, `malicious_file` o `malicious_url`
- `data`: string, es donde se especifican los datos necesarios de la solicitud entrante para poder satisfacer el caso de uso. En esta versión puede contener una IP, un hash MD5 o una URL

Con respecto a la comunicación entre el módulo y el core MISP, los eventos deben ser filtrados mediante el endpoint `/events/restSearch` expuesto por el core. En lo que refiere a la comunicación entre el módulo y sus submódulos, esta ocurre internamente y será detallada en las siguientes secciones.

4.3.3. Estructura

El módulo central debe implementar al menos 3 funciones: `introspection`, `handler` y `version`. A continuación se detallan las mismas.

`introspection`

Esta función devuelve un diccionario con los atributos soportados, tanto de input como de output, así como los IDs de los casos de uso soportados. El diccionario devuelto es el que utilizará el endpoint `GET /modules` para devolver información sobre el módulo.
La función devuelve un diccionario con los siguientes elementos:

- `input`: lista de strings con los siguientes elementos: `hostname`, `use_case_id`, `data`
- `output`: lista de strings con el siguiente elemento: (`malicious`)
- `available_use_cases`: lista de strings con todos los IDs de los casos de uso soportados. Esta lista es generada dinámicamente a partir de los submódulos presentes

`handler`

Es la función encargada de procesar el input para obtener el identificador de caso de uso y luego delegar el procesamiento al submódulo correspondiente, el cual retornará un booleano indicando si se encontraron eventos maliciosos con las características presentadas. Específicamente, esta función invoca a la función `handler` del submódulo correspondiente. Esta función acepta un documento JSON como parámetro, este es el JSON proporcionado en la solicitud.
La función devuelve un diccionario con las siguientes keys anidadas: `results->values->malicious`, donde el valor de `` `malicious` `` es el booleano retornado por el submódulo.

`version`

Devuelve un diccionario con el número de versión y los metadatos asociados. Es usada junto con `introspection` para devolver información sobre el módulo en `GET /modules`. En particular contiene los siguientes atributos:

- `version`: número de versión (string). Ej: '1.1'
- `author`: nombre de el/los autor/es
- `description`: breve descripción del módulo. Debe ser "Module to integrate ModSecurity with MISP"
- `module-type`: debe ser igual a la siguiente lista: `['export']`

4.3.4. Estructura submódulo

Como se comentó en la sección anterior, cada submódulo implementa al menos una función `handler`.
Esta función acepta 2 parámetros: `use_case_id` y `data`, con el mismo contenido especificado en 4.3.2. Es la encargada de generar la consulta y enviarla al core de MISP para luego procesar su respuesta. Retorna un booleano indicando si se encontraron eventos maliciosos con las características presentadas.

4.4. Definición de templates de objeto

Cada submódulo contiene el UUID del template de objeto correspondiente.
A continuación se definen los templates de objeto para cada caso de uso.

4.4.1. Blacklistear lista de IPs

En el siguiente bloque de código se define el template de objeto del caso de uso "blacklistear lista de IPs".

```
{
  "attributes": {
    "ip-src": {
      "description": "Blacklisted IP",
      "misp-attribute": "ip-src",
      "ui-priority": 1
    },
  },
  "description": "IP blacklist",
  "meta-category": "network",
  "name": "blacklist-ip",
  "required": [
    "ip-src"
  ],
  "uuid": "2e03fcc5-0b7e-4781-b6d4-8ed6cd3f7faf",
  "version": 1
}
```

Listing 4.1: Template de objeto: blacklistear lista de IPs

El único atributo utilizado es `ip-src`, y se eligió la categoría `network` ya que el elemento pertenece a ese dominio. El elemento `ui-priority` indica el orden en que los atributos figurarán en la UI, en este caso se encuentra presente ya que de lo contrario el atributo no figurará en la UI al momento de crear un objeto asociado a este template.

El elemento `uuid` fue generado con un generador de UUID versión 4 (a diferencia de otras versiones, en esta versión los UUIDs son generados de forma aleatoria).

4.4.2. URLs maliciosas

En el siguiente bloque de código se define el template de objeto del caso de uso "URLs maliciosas".

```
{
  "attributes": {
    "url": {
      "description": "Malicious URL",
      "misp-attribute": "url",
      "ui-priority": 1
    }
  },
  "description": "Malicious URL",
  "meta-category": "network",
  "name": "malicious-url",
  "required": [
    "url"
  ],
  "uuid": "e9f187c9-9440-4988-9225-3161c49d7453",
  "version": 1
}
```

Listing 4.2: Template de objeto: URLs maliciosas

En este template se utiliza únicamente el atributo `url`. Los otros elementos son similares a los presentados en 4.4.1.

4.4.3. Archivos maliciosos

En el siguiente bloque de código se define el template de objeto del caso de uso "archivos maliciosos".

```
{
  "attributes": {
    "md5": {
      "description": "Malicious file MD5 hash",
      "misp-attribute": "md5",
      "ui-priority": 1
    },
  },
  "description": "Malicious file",
  "meta-category": "file",
  "name": "malicious-file",
  "required": [
    "md5"
  ],
  "uuid": "ec1fd23f-1fc5-4ecd-b642-9482c71b3362",
  "version": 1
}
```

Listing 4.3: Template de objeto: archivos maliciosos

El único atributo utilizado es `md5`. A diferencia de los templates anteriores, este template se encuentra asociado a la categoría `file` y no `network`. Los otros elementos

son similares a los presentados en 4.4.1.

Capítulo 5

Implementación

Se implementó un operador personalizado (`@MISP`) de ModSecurity escrito en C y un módulo MISP (`modsec`) escrito en python. Además se implementaron distintas reglas de ModSecurity para satisfacer los casos de uso.
A continuación se detalla lo implementado para cada uno de los componentes mencionados así como la prueba de concepto.

5.1. Operador @MISP

Como se mencionó en 2.4.2, para implementar un operador de ModSecurity basta con desarrollar un módulo Apache en C. Para esto es necesaria la implementación de una estructura (`op_misp_module`) y dos funciones (`register_hooks` y `hook_pre_config`). A continuación se detalla la implementación de los puntos mencionados:

```
/**
 * Function to be invoked every time Apache is reconfigured
 */
static int hook_pre_config(apr_pool_t *mp, apr_pool_t *mp_log, apr_pool_t *
    mp_temp) {
    void (*fn)(const char *name, void *fn_init, void *fn_exec);

    // Look for the registration function exported by ModSecurity
    fn = APR_RETRIEVE_OPTIONAL_FN(modsec_register_operator);
    if (fn) {
        // Register the `misp` operator under the name "misp"
        fn("misp", (void *)op_misp_init, (void *)op_misp_exec);
    } else {
        ap_log_error(APLOG_MARK, APLOG_ERR | APLOG_NOERRNO, 0, NULL,
            "mod_op_misp: Unable to find modsec_register_operator.");
    }

    return OK;
}

/**
 * Callback to register `hook_pre_config`
 */
static void register_hooks(apr_pool_t *p) {
    ap_hook_pre_config(hook_pre_config, NULL, NULL, APR_HOOK_LAST);
}
```

```
/* Dispatch list for API hooks */
module AP_MODULE_DECLARE_DATA op_misp_module = {
    STANDARD20_MODULE_STUFF,
    NULL,                  /* create per-dir    config structures */
    NULL,                  /* merge  per-dir    config structures */
    NULL,                  /* create per-server config structures */
    NULL,                  /* merge  per-server config structures */
    NULL,                  /* table of config file commands       */
    register_hooks         /* register hooks                      */
};
```

Listing 5.1: Código de @MISP referente al registro del módulo Apache

La estructura `op_misp_module` es utilizada por Apache para verificar que la librería es efectivamente un módulo[41]. En ella se define el nombre del módulo (debe ser único) y la función encargada de registrar los hooks (`register_hooks`). Esencialmente, un hook es un mensaje informándole al servidor que se va a servir o al menos a inspeccionar determinadas solicitudes. Este hook a su vez registra a la función `hook_pre_config`, la cual es invocada cada vez que Apache es reconfigurado.

La función `hook_pre_config` se encarga de registrar el operador @MISP. Esto es necesario para que pueda ser utilizado en las reglas de ModSecurity. Para esto se usa el mecanismo de Apache "funciones opcionales"[42], el cual consiste de dos pasos: primero se le solicita a Apache la función para registrar operadores (mediante `APR_RETRIEVE_OPTIONAL_FN`), la cual es exportada por ModSecurity, y luego se utiliza esa función para registrar el operador. La función obtenida utiliza 3 parámetros: por un lado el nombre de la función (misp) y por el otro la función inicializadora y la función en sí (función ejecutora). Ambas funciones deben seguir una firma especificada por ModSecurity. Esta función cuenta con los parámetros `op_misp_init` y `op_misp_exec`, los cuales indican qué función se utilizará para inicializar el operador y cual se encargará del procesamiento respectivamente.

En las siguientes secciones se detalla la implementación de las funciones utilizadas al registrar el módulo: la función inicializadora `op_misp_init` y la función ejecutora `op_misp_exec`. También se detallan las funciones auxiliares utilizadas por las mismas.

5.1.1. Función inicializadora

Esta es la función encargada de inicializar el operador para un caso de uso dado, sigue la siguiente firma:

```
static int op_misp_init(msre_rule *rule, char **error_msg)
```

Listing 5.2: Función inicializadora op_misp_init

La función cuenta con los siguientes parámetros:

- `rule`: estructura que almacena los datos de la regla, en particular cuenta con el campo `op_param`, el cual contiene un string con el parámetro proporcionado.

- `error_msg`: puntero a mensaje de error

El primer paso de la función es extraer el identificador del caso de uso de la regla así como la ruta al archivo de configuración. En caso de que alguno de los parámetros no sea proporcionado se devuelve el mensaje de error "`Missing parameter for operator 'misp'.`". Con "devolver el mensaje de error" nos referimos a almacenar el mensaje de error en la memoria apuntada por el puntero `error_msg` y abortar la ejecución, es decir, devolver el int 0. Esto impide que ModSecurity sea iniciado.
Luego del control de los parámetros se parsea el archivo de configuración (ver 5.1.3) y se almacenan los datos obtenidos en una estructura para que puedan ser accedidos en la etapa de ejecución, de esta forma no es necesario volver a cargar el archivo de configuración.
A continuación se chequea la conexión con el servidor de módulos MISP mediante una invocación al servicio `GET /modules` utilizando los datos obtenidos del archivo de configuración: en caso de no obtener una respuesta exitosa se devuelve el error "`Error when connecting to MISP server`". De lo contrario se parsea la respuesta utilizando la librería YAJL (ver 5.1.4) para verificar que el módulo `modsec` se encuentre dentro de los módulos disponibles. En caso de que no sea asi, se devuelve el mensaje de error "`'modsec' module not found in MISP server`". De lo contrario se chequean los submódulos del módulo `modsec` listados bajo la key `available_use_cases` para verificar que el caso de uso proporcionado se encuentre dentro de los submódulos de `modsec`. En caso de que el submódulo no sea encontrado se devuelve el mensaje de error `Unrecognized parameter for operator 'misp'.`
Si no ocurrieron errores se devuelve el int 1 y se finaliza la inicialización del operador.

5.1.2. Función ejecutora

Esta función se encarga del procesamiento de la solicitud entrante. Sigue la siguiente firma:

```
op_misp_exec(modsec_rec *msr, msre_rule *rule, msre_var *var, char **
    error_msg)
```

Listing 5.3: Función ejecutora op_misp_exec

Cuenta con cuatro parámetros:

- `msr`: estructura donde se almacenan todos los datos de la transacción HTTP
- `rule`: idem `op_misp_init`
- `var`: la estructura de la variable en la cual se encuentra almacenados los datos a inspeccionar
- `error_msg`: idem `op_misp_init`

En ella se procesan los datos de la solicitud (en caso de ser necesario) y se construye el payload a enviar al módulo MISP a partir de los datos obtenidos del archivo de

configuración, los datos (procesados) de la solicitud y el caso de uso. Luego se realiza una consulta al módulo desarrollado en MISP mediante el endpoint `POST /query` y se extrae el campo `malicious` de la respuesta. En caso de que el valor extraído sea `true` se activa la regla mediante el retorno del int 1 y se almacena el mensaje "`MISP event match for $caso_de_uso_id with var $variable`", almacenándolo de la misma forma en que se almacenan los mensajes de error en la función inicializadora. En caso contrario la función devuelve el int 0.
En el caso en que no se pueda establecer la conexión con el servidor se procede de la misma forma que con la función inicializadora.
Ejemplo de payload generado para el caso de uso blacklistear IP:

```
{
    "module": "modsec",
    "use_case_id": "blacklist_ip",
    "data": "10.5.7.7",
    "misp_api_key": "uxRP5vT7dzTVCBoyraLL4CBvTu1TG5CCCaOavlSO"
}
```

Listing 5.4: Payload generado para el caso de uso blacklistear IP

Además de las funciones mencionadas, fue necesaria la implementación de otras funciones para poder satisfacer las necesidades de la función inicializadora y la ejecutora. En particular fue necesaria la implementación de funciones que se encarguen de resolver los siguientes puntos:

- Solicitudes HTTP
- Procesamiento y extracción de datos de las respuestas del módulo MISP
- Procesamiento del archivo de configuración de @MISP

A continuación se detallan las funciones mencionadas.

5.1.3. Solicitudes HTTP

Toda la comunicación con el módulo MISP ocurre a través de HTTP, es por esto que se implementó una función para realizar solicitudes HTTP mediante la utilización de la librería Neon. Esta función realiza la solicitud HTTP y almacena la respuesta en un string.

Se utilizó un timeout de 20 segundos: si no se obtiene una respuesta de MISP en menos de ese periodo se asume que no se pudo establecer una conexión. Se empleó un timeout pequeño (en comparación con el timeout tradicional de 60 segundos) ya que probablemente sean necesarias varias consultas al servidor de módulos MISP (una por regla), retrasando el procesamiento de la solicitud.

5.1.4. Procesamiento de la respuesta del módulo MISP

Dado que todas las respuestas del servidor de módulos MISP se encuentran en formato JSON, es necesario contar con una librería para el procesamiento de las

mismas. Afortunadamente ModSecurity tiene como dependencia a la librería YAJL (Yet Another JSON Library)[43][44], una pequeña librería para parsear y validar JSON. Se eligió YAJL ya que cumple con el requerimiento de parsear JSON y es dependencia de ModSecurity.

En particular fue necesario parsear dos tipos de respuestas: la respuesta a `GET /modules` y la respuesta a `POST /query`.
`GET /modules` se utiliza tanto para verificar la conexión con el servidor de módulos MISP como para obtener la lista de submódulos disponibles, y `POST /query` es utilizada para chequear si determinados parámetros de una request son maliciosos contra el módulo modsec.
Es por esto que se implementaron distintas funciones para el procesamiento de las respuestas mencionadas así como para la extracción de determinados campos de las mismas (por ejemplo el campo `malicious` de la respuesta a `POST /query`).

5.1.5. Procesamiento del archivo de configuración

Se implementó una función encargada del procesamiento del archivo de configuración, el mismo sigue una estructura del tipo KEY=VALUE. Se encarga de cargar una estructura formada por el host de los módulos misp, el puerto del mismo y la API key de MISP. Esta estructura es accedida posteriormente para poder realizar las solicitudes HTTP al servidor de módulos MISP. Ejemplo de archivo de configuración:

```
MISP_MODS_HOST=10.5.0.10
MISP_MODS_PORT=6666
MISP_API_KEY=uxRP5vT7dzTVCBoyraLL4CBvTu1TG5CCCaOavlSO
```

Listing 5.5: Ejemplo de archivo de configuración

5.2. Módulo MISP "modsec"

Para la implementación del módulo MISP fue necesaria la implementación de un script en python, ya que esta es la única forma de implementar un módulo. Como se mencionó en 4.3.1, se buscó que el mismo fuera extensible mediante el desarrollo de submódulos. Se implementó un archivo `modsec.py`, el cual es el verdadero módulo MISP, y un script distinto (submódulo) por cada caso de uso siguiendo la siguiente estructura:

```
modsec.py
_modsec
  blacklist_ip
    main.py
  malicious_file
    main.py
  malicious_url
    main.py
```

Como se puede apreciar en la estructura, en el primer nivel se encuentra el archivo `modsec.py` y el directorio `_modsec`. Este último lleva un guión bajo como prefijo

para evitar que el servidor de módulos MISP lo reconozca como un módulo. Cada submódulo se encuentra contenido en su propia carpeta, la cual debe ser nombrada con el nombre del caso de uso, y dentro de la misma debe contener obligatoriamente un archivo `main.py`. Se podría haber logrado lo mismo sin el uso de estas carpetas, donde cada submódulo sea representado por un script (por ejemplo `blacklist_ip.py`), pero el enfoque de carpetas permite definir varios archivos por caso de uso, permitiendo modularizar los mismos en caso de ser necesario. Para los casos de uso implementados solo se utilizó el archivo `main.py` debido al pequeño tamaño del código.

En `modsec.py` se definen todas las funciones necesarias para un módulo MISP: `handler`, `introspection` y `version` (ver 2.4.1).

La función `introspection` devuelve los atributos de input y output soportados de la misma forma que todos los módulos MISP. Como se mencionó en 4.3.2, ambos atributos son listas de strings: input se encuentra compuesto de los strings `use_case_id`, `data` y `misp_api_key` y output por `malicious`. Además de esos atributos se devuelven los casos de uso soportados dentro del atributo `available_use_cases`, esto se consigue listando el contenido del directorio `_modsec`, es por eso que el nombre de cada directorio dentro de `_modsec` se debe corresponder con el identificador del caso de uso. Al obtener los nombres de las carpetas dinámicamente es posible agregar un nuevo caso de uso simplemente creando una carpeta y un archivo `main.py`, evitando tener que modificar el código existente y reiniciar el servidor de módulos MISP.

La función `handler` se encarga de parsear el caso de uso proporcionado por el cliente y delegar el procesamiento al submódulo correspondiente basándose en el identificador del caso de uso (el valor de `use_case_id` especificado por @MISP). Es por esto que cada submódulo implementa una función `handler`, la cual es invocada con los datos proporcionados por el cliente desde la función `handler` del módulo modsec.

La función `handler` de cada submódulo se encarga de conectarse con el servidor MISP utilizando la API key proporcionada por el cliente y el host especificado en el archivo `keys.py`. Esta función realiza una consulta al servidor MISP para encontrar todos los eventos con el tipo adecuado (el tipo queda determinado por el caso de uso) y el valor proporcionado por el cliente. Luego se filtran todos los eventos que no contengan el template de objeto necesario (también determinado por el caso de uso) ya que no pueden ser filtrados mediante la API. En caso de que se encuentre algún evento con las características especificadas esta función envía una respuesta al cliente indicando que se encontraron elementos maliciosos, de lo contrario se le responde que no se encontraron. Para indicarlo se usa el atributo de output `malicious`.

Para la comunicación entre el módulo MISP y el servidor MISP se utilizó la librería PyMISP, ya que facilita el consumo de la API y es dependencia del servidor de módulos MISP[45].

El archivo `keys.py` se encuentra en el directorio de módulos MISP, en él se define la URL del servidor MISP, como lo indica la documentación de PyMISP[46]. Como se mencionó en el párrafo anterior, la API key de MISP es proporcionada por @MISP y no por este archivo, de esta forma se puede revocar el acceso a MISP de cualquier

instancia de ModSecurity de forma granular como se mencionó en 4.2.3.

5.3. Reglas de ModSecurity

Las reglas son declaradas de la siguiente forma:

```
SecRule *VARIABLE* "@MISP *ID_CASO_DE_USO* *RUTA_ARCHIVO_CONF*" "*
  TRANSFORMATIONS*, *ACTIONS*"
```

Listing 5.6: Formato general para las reglas que utilizan @MISP

Algunos elementos son incluidos en todas (o casi todas las reglas). En todas las reglas se incluyen las acciones `log`, `auditlog` y `deny`. `log` indica que la regla debe ser logueada en el log de errores de Apache en el caso de que sea activada y `auditlog` marca a la transacción para que sea logueada en el audit log. `deny` provoca que la transacción sea abortada en caso de que la regla sea activada.

La ruta al archivo de configuración en el que se almacenan los datos necesarios para la conexión con el servidor de módulos MISP y la API key de MISP (ver 5.1.5) también se encuentra presente en la definición de cada regla, aunque no necesariamente tiene que ser la misma ruta. Esto permite parametrizar la conexión con MISP a nivel de regla.

A excepción de la regla del caso de uso de archivos maliciosos, en la cual es necesaria una transformación para obtener un hash, las demás reglas emplean la transformación `t:none`. Esta transformación es utilizada para resetear el pipeline de transformaciones y asegurar la obtención de la variable especificada sin transformaciones aplicadas.

En las siguientes secciones se describen las reglas implementadas para cada caso de uso.

5.3.1. Blacklistear conjunto de IPs:

A continuación se describe la regla para este caso de uso.

```
SecRule REMOTE_ADDR "@MISP blacklist-ip /path/to/config.conf" ""id
  :'210000',phase:1,t:none,log,auditlog,deny"
```

Listing 5.7: Regla del caso de uso blacklistear conjunto de IPs

La acción `phase` es utilizada para colocar regla en la primer fase, la fase que corresponde con los headers de la solicitud.

5.3.2. URLs maliciosas:

En el siguiente bloque de código se puede apreciar la regla para este caso de uso.

```
SecRule REQUEST_URI "@MISP malicious-url /path/to/config.conf" "id
  :'210001',phase:1,t:none,log,auditlog,deny"
```

Listing 5.8: Regla del caso de uso URLs maliciosas

En esta regla se emplea la acción `phase` con el mismo valor que en la regla anterior por el mismo motivo.

5.3.3. Subida/descarga de archivos maliciosos con firma conocida:

Subida

A continuación se describe la regla para este caso de uso, en particular para la subida.

```
SecRequestBodyAccess On
SecTmpSaveUploadedFiles On
SecRule FILES_TMP_CONTENT "@MISP malicious-file /path/to/config.conf" "id
  :'210002',phase:2,t:md5,t:hexEncode,log,auditlog,deny"
```

Listing 5.9: Regla del caso de uso subida/descarga de archivos maliciosos con firma conocida: subida

A diferencia de las reglas anteriores, para que estas reglas funcionen correctamente es necesaria la inclusión de algunas directivas.

La directiva `SecRequestBodyAccess` es utilizada para activar el procesamiento de los cuerpos de una request, de lo contrario los archivos adjuntos no podrán ser accedidos.

La variable `FILES_TMP_CONTENT` almacena el contenido de los archivos en una solicitud multipart/form-data. Para poder utilizar esta variable es necesario almacenar los archivos subidos por el usuario, esto puede ser logrado mediante la utilización de 2 directivas: `SecUploadKeepFiles` y `SecTmpSaveUploadedFiles`. Al ser activadas ambas directivas provocan que se almacene el contenido de los archivos en disco. Con `SecTmpSaveUploadedFiles` los archivos son eliminados luego de finalizada la transacción, a diferencia de `SecUploadKeepFiles` que provoca que los archivos queden almacenados permanentemente. Se utilizó `SecTmpSaveUploadedFiles` ya que no es necesario almacenar los archivos para su posterior inspección.
Existen otras directivas útiles para los casos en que es necesario almacenar el cuerpo de la solicitud que no fueron empleadas dada la generalidad del servidor web, por lo que se emplean sus valores por defecto, pero vale la pena mencionarlas:

- `SecRequestBodyLimit`: establece el tamaño máximo del cuerpo de la solicitud que ModSecurity aceptará. En caso de sobrepasarlo se ejecutará la acción especificada en `SecRequestBodyLimitAction`. Default: `reject`
- `SecRequestBodyNoFilesLimit` idem a `SecRequestBodyLimit` pero excluye los archivos adjuntos. Default: `reject`

- `SecRequestBodyInMemoryLimit`: establece el tamaño del buffer de memoria a nivel de solicitud. Default: `128 KB`
- `SecRequestBodyLimitAction`: especifica qué ocurre cuando se sobrepasa el limite de tamaño de la solicitud. Default: `reject`

Con respecto a las transformaciones, se utiliza la transformación `md5` para transformar los archivos adjuntos en la solicitud convirtiéndolos en su hash md5. La acción `phase:2` es utilizada para colocar regla en la segunda fase, la fase que corresponde con el cuerpo de la solicitud.

Descarga

Para la descarga se reutiliza el mismo parámetro utilizado en la subida (`malicious-file`). Es necesaria la implementación de dos reglas encadenadas, esto permite combinar varias reglas utilizando las compuertas lógicas `AND` y `OR`. La primera regla implementada provoca que la segunda regla se ejecute sólo si el recurso accedido se encuentra dentro del directorio objetivo. La segunda regla es análoga a la regla de subida, como se puede apreciar en el siguiente extracto de código.

```
SecResponseBodyAccess On
SecResponseBodyMimeType text/plain text/html null
SecRule REQUEST_FILENAME "@rx (?i)^/upload/.*" \
                         "id:210003,phase:4,deny,nolog,chain"
    SecRule RESPONSE_BODY "@MISP malicious_file /root/misp_conn.conf" \
                          "t:md5,t:hexEncode,log,auditlog"
```

Listing 5.10: Reglas del caso de uso subida/descarga de archivos maliciosos con firma conocida: descarga

La primera regla hace uso del operador `rx`, el cual evalúa una expresión regular contra la variable, en este caso la ruta del recurso accedido. La acción `chain` provoca que la segunda regla se ejecute sólo si la ruta del recurso accedido comienza con el string "/uploads/". La acción "phase:4" es utilizada para colocar la regla en la cuarta fase, la fase que corresponde con el cuerpo de respuesta de la solicitud.
De forma similar al caso de subida, es necesaria la inclusión de algunas directivas. En particular es necesaria la inclusión de la directiva `SecResponseBodyAccess` con el valor `On` para que el cuerpo de la respuesta HTTP pueda ser analizado. También es necesaria la inclusión de `SecResponseBodyMimeType` con el valor `text/plain text/html null`: esta directiva indica qué tipos MIME[47] serán considerados para el análisis. El valor `null` es utilizado para que se analicen los cuerpos que tengan un tipo MIME desconocido.

5.4. Prueba de concepto

Para la prueba de concepto se implementó una infraestructura con el propósito de simular un escenario real, conteniendo todos los componentes presentados en 4.1. En el servidor MISP contenido en ella se agregaron eventos catalogados como maliciosos

y asociados a los templates de objeto correspondientes a los casos de uso, para luego evaluar la respuesta del servidor web ante distintas solicitudes HTTP. Esa respuesta fue comparada con la respuesta de las mismas solicitudes en el caso en que no existen eventos maliciosos en MISP.

Para la generación de eventos se optó por utilizar la interfaz web que provee el servidor MISP debido a que ese es el método en el que seguramente se terminen generado los mismos en producción.
A continuación se detalla la infraestructura utilizada así como las pruebas realizadas para cada caso de uso.

5.4.1. Infraestructura

El ambiente de prueba consiste en una serie de hosts conectados mediante una subred (`10.5.0.0/16`) sobre Docker. Específicamente se implementó una red con 3 componentes: un servidor web, un servidor MISP y una base de datos.

El servidor web consiste de una imagen de Debian Buster sobre la cual corre un servidor Apache version 2.4.48 bajo la IP `10.5.0.80`. Apache escucha en el puerto 80 y 443.

El servidor MISP consiste en la imagen MISP Docker[48], una imagen basada en Ubuntu y recomendada en la página oficial de MISP[49]. Esta imagen se encuentra compuesta por un servidor MISP junto con el servidor de módulos MISP, y ambos fueron alojados en el mismo host bajo la IP `10.5.0.10`. El servidor MISP escucha en el puerto 80 mientras que el servidor de módulos MISP lo hace en el 6666.

La base de datos consiste en una imagen de MySQL Server versión 5.7, su IP es asignada dinámicamente ya que no es necesario referenciarla manualmente, MISP se comunica con ella a partir de una IP mapeada por Docker.

Todas las solicitudes fueron originadas desde el host que levantó la infraestructura de Docker, el cual tiene la IP `10.5.0.1` asignada.

5.4.2. Blacklistear conjunto de IPs

Se realizó una solicitud al servidor web, en particular un GET a `http://10.5.0.80:80/` y se comprobó que el mismo respondiera con la página de prueba incluida por defecto en la instalación de Apache. Luego se generó un evento conteniendo un objeto asociado al template del caso de uso "Blacklistear conjunto de IPs" (template con uuid "2e03fcc5-0b7e-4781-b6d4-8ed6cd3f7faf"). Dentro del objeto se agregó el atributo `ip-src`, usando como valor la IP privada de la interfaz por la cual se realizó la primer solicitud (`10.5.0.1`). A continuación se realizó otra solicitud idéntica a la primera y se comprobó que el server respondiera con el status HTTP 403 Forbidden (el status por defecto empleado por ModSecurity cuando se ejecuta la acción `deny`).

5.4.3. Subida/descarga de archivos maliciosos con firma conocida

Para ambos casos se utilizó un archivo conteniendo una shell de PHP, específicamente la shell C99Shell[50] con nombre de archivo "c99shell.php", un shell comúnmente utilizado para ganar acceso a servidores. El mismo se almacenó en el directorio objetivo "/uploads" (el mismo especificado en la regla asociada a este caso de uso).

Se realizó una solicitud al servidor web de tipo multipart/form-data, particularmente a la URL `http://10.5.0.80:80/` y se comprobó que el servidor respondiera con la página de prueba. Luego se realizó una segunda solicitud para descargar el archivo malicioso, específicamente
`GET http://10.5.0.80:80/uploads/c99shell.php` y se comprobó que el servidor respondiera con el contenido del archivo. De esta forma se comprobó el correcto funcionamiento en caso de que no exista un evento malicioso asociado al archivo.

A continuación se generó un evento conteniendo un objeto asociado al template del caso de uso (template con uuid "ec1fd23f-1fc5-4ecd-b642-9482c71b3362"). Dentro de ese objeto se encapsuló el atributo "md5" con el valor del hash MD5 del archivo ("49f39e957ce08224aee4907d5bc7fc0a"). Luego de generado se repitieron las 2 solicitudes HTTP previas y se verificó que el servidor respondiera con el status HTTP 403 para ambas.

Para verificar el correcto funcionamiento de las reglas encadenadas se copió el archivo malicioso de forma que quede alojado en la ruta
`http://10.5.0.80:80/c99shell.php` y se logró descargarlo satisfactoriamente.

5.4.4. URLs maliciosas

Para testear este caso de uso se asumió que el recurso `/shell.php` es malicioso. De forma similar a los casos anteriores, se realizó una solicitud GET a la URL `http://10.5.0.80:80/shell.php` y se comprobó que el servidor web respondiera con el código HTTP 404 dado que no existe un recurso para esa URL/método. Luego se generó un evento en MISP de forma análoga a los casos anteriores utilizando el template de objeto con uuid "e9f187c9-9440-4988-9225-3161c49d7453". En él se cargó el atributo "url" con el valor "/shell.php" y a continuación se volvió a realizar la misma request, comprobándose que el servidor respondiera con el status HTTP 403.

Capítulo 6

Conclusiones y trabajo futuro

El principal aporte de este trabajo es la posibilidad de poder contar con un WAF que pueda detectar satisfactoriamente solicitudes maliciosas a partir de información obtenida dinámicamente y en tiempo real de un repositorio de información sobre amenazas, el cual por su naturaleza se encuentra en constante actualización. Esto permite centralizar la definición de amenazas, simplificando la tarea de mantener el WAF (o red de WAFs) actualizado ante nuevos ataques sin la necesidad de modificar la configuración de los mismos, como ocurre con el CRS.

Si bien los casos de uso atacados son relativamente simples comparándolos con otros ataques más sofisticados, sirven a modo de prueba de concepto y de ejemplo para la implementación de casos de uso más complejos en un futuro. Dada la facilidad de extensión del software desarrollado, agregar un nuevo caso de uso implica simplemente agregar una regla al WAF y desarrollar un submódulo en Python, por lo que no es necesario que el desarrollador deba ocuparse con tareas más complejas como lo son el desarrollo de módulos Apache.

En un futuro la solución implementada podría ser mejorada implementando un método para extender el archivo de configuración. Este podría ser extendido para soportar parámetros arbitrarios de la forma X=Y, donde cada key con su correspondiente valor sean incluidos en el payload hacia MISP. Esto permitiría configurar aspectos del procesamiento del submódulo MISP correspondiente sin la necesidad de alterar el código del módulo Apache. Con esto se podría soportar nuevos casos de uso que requieran de configuraciones extra.

También se podría trabajar en la integración de ambas tecnologías pero en el sentido inverso, es decir, alimentar MISP con eventos generados a partir de requests entrantes de ModSecurity. Esto permitiría que dada una red de WAFs asociados a MISP, si un WAF detecta una nueva amenaza automáticamente todos los WAFs de la red queden protegidos ante la misma. Para esto se podría utilizar el proyecto "Honeypot" mencionado en 2.5.

Bibliografía

[1] *Aumento en ciberataques.* URL: https://sectigostore.com/blog/42-cyber-attack-statistics-by-year-a-look-at-the-last-decade.

[2] *OWASP: WAF.* URL: https://owasp.org/www-community/Web_Application_Firewall.

[3] *Modelo OSI.* URL: https://www.os3.nl/_media/info/5_osi_model.pdf.

[4] *ModSecurity.* URL: https://web.archive.org/web/20210110104956/https://www.modsecurity.org/.

[5] *Mod Security Anomaly Score.* URL: https://www.modsecurity.org/CRS/Documentation/anomaly.html.

[6] *Advanced Topic of the Week: Traditional vs. Anomaly Scoring Detection Modes.* URL: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/advanced-topic-of-the-week-traditional-vs-anomaly-scoring-detection-modes.

[7] *Anomaly scoring is a better way to detect a real attack.* URL: https://web.archive.org/web/20210127171926/https://blogs.akamai.com/2013/12/anomaly-scoring-is-a-better-way-to-detect-a-real-attack.html.

[8] *Core Rule Set.* URL: https://owasp.org/www-project-modsecurity-core-rule-set.

[9] *Mod Security Severity Score.* URL: https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-(v2.x)#severity.

[10] *IoC.* URL: https://tools.ietf.org/html/draft-paine-smart-indicators-of-compromise-01.

[11] *Threat Intelligence Sharing Platforms: An Exploratory Study of Software Vendors and Research Perspectives.* URL: https://www.wi2017.ch/images/wi2017-0188.pdf.

[12] *Observables e Indicadores.* URL: https://stixproject.github.io/documentation/concepts/composition.

[13] *kill chains.* URL: https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf.

[14] *MISP.* URL: https://www.misp-project.org.

[15] *MISP attributes*. URL: https://www.misp-project.org/datamodels.

[16] *MISP objects*. URL: https://github.com/MISP/misp-objects.

[17] *MISP objects*. URL: https://www.misp-project.org/objects.html.

[18] *Managing MISP feeds*. URL: https://www.circl.lu/doc/misp/managing-feeds.

[19] *MISP API*. URL: https://www.circl.lu/doc/misp/automation.

[20] *PyMISP*. URL: https://pymisp.readthedocs.io/en/latest/modules.html.

[21] *MISP Modules*. URL: https://github.com/MISP/misp-modules.

[22] *ModSecurity Handbook (2nd edition): Logging*. URL: https://www.feistyduck.com/library/modsecurity-handbook-2ed-free/online/ch04-logging.html.

[23] *ModSecurity: Logging and Debugging*. URL: https://www.nginx.com/blog/modsecurity-logging-and-debugging.

[24] *Analyzing Mod Security Logs*. URL: https://resources.infosecinstitute.com/analyzing-mod-security-logs.

[25] *Lua*. URL: https://www.lua.org/manual/5.4/readme.html.

[26] Christian Folini e Ivan Ristić. «ModSecurity Handbook». En: 2.ª ed. Build 279. London: Feisty Duck, 2020. Cap. 14, págs. 239-255.

[27] *OWASP: Honeypot Project*. URL: https://github.com/OWASP/Honeypot-Project.

[28] *ELK stack*. URL: https://www.elastic.co/what-is/elk-stack.

[29] *OWASP: Honeypot Project (architecture diagram)*. URL: https://raw.githubusercontent.com/OWASP/Honeypot-Project/master/honeytraps/honeytrap_arch.jpg.

[30] *SecRemoteRules*. URL: https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-%28v2.x%29#SecRemoteRules.

[31] Christian Folini e Ivan Ristić. «ModSecurity Handbook». En: 2.ª ed. Build 279. London: Feisty Duck, 2020. Cap. 15, pág. 295.

[32] *APR*. URL: https://apr.apache.org.

[33] *ssdeep*. URL: https://ssdeep-project.github.io/ssdeep/index.html.

[34] *tlsh*. URL: https://github.com/trendmicro/tlsh.

[35] *MISP queries: wildcards*. URL: https://www.misp-project.org/2018/09/06/MISP.2.4.95.released.html.

[36] *mod_proxy_http*. URL: https://fossies.org/linux/httpd/modules/proxy/mod_proxy_http.c.

[37] *Curl*. URL: https://curl.se/libcurl.

[38] *Serf*. URL: https://serf.apache.org.

[39] *Neon*. URL: https://notroj.github.io/neon.

[40] *Neon performance*. URL: https://curl.se/libcurl/neon.html.

[41] *Apache: guía sobre módulos*. URL: https://httpd.apache.org/docs/2.4/developer/modguide.html.

[42] *Apache: funciones opcionales*. URL: https://ci.apache.org/projects/httpd/trunk/doxygen/group__APR__Util__Opt.html.

[43] *YAJL*. URL: https://lloyd.github.io/yajl/.

[44] *YAJL dependencia de ModSec*. URL: https://www.feistyduck.com/library/modsecurity-handbook-2ed-free/online/ch02-installation.html.

[45] *Dependencias PyMISP*. URL: https://github.com/MISP/misp-modules/blob/main/Pipfile.

[46] *PyMISP docs*. URL: https://www.circl.lu/doc/misp/pymisp/.

[47] *Tipos MIME*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types.

[48] *MISP Docker*. URL: https://github.com/misp/misp-docker.

[49] *Download MISP*. URL: https://www.misp-project.org/download.

[50] *C99Shell*. URL: https://github.com/KaizenLouie/C99Shell-PHP7/blob/master/c99shell.php.txt.

[51] *ModSecurity Handbook (2nd edition)*. URL: https://www.feistyduck.com/library/modsecurity-handbook-2ed-free/online/.

[52] *ModSecurity Wiki*. URL: https://github.com/SpiderLabs/ModSecurity/wiki.

[53] *MISP features*. URL: https://www.misp.software/features.html.

[54] A. Iklody A. Dulaunoy. *MISP object template (internet draft)*. 2019. URL: https://tools.ietf.org/html/draft-dulaunoy-misp-object-template-format-03.

[55] *MISP book*. URL: https://misp.gitbooks.io/misp-book.

Capítulo 7

Anexo

7.1. Template de objeto MISP para DDoS

El template de objeto presentado a continuación es el template oficial utilizado para describir un ataque DDoS. Como lo indica el valor del elemento `requiredOneOf`, para que un objeto asociado a este template sea válido es necesario que tenga definido el atributo `ip-dst`, `ip-src` o `domain-dst`.

```
{
  "attributes": {
    "domain-dst": {
      "categories": [
        "Network activity",
        "External analysis"
      ],
      "description": "Destination domain (victim)",
      "misp-attribute": "domain",
      "ui-priority": 1
    },
    "dst-port": {
      "categories": [
        "Network activity",
        "External analysis"
      ],
      "description": "Destination port of the attack",
      "misp-attribute": "port",
      "ui-priority": 0
    },
    "first-seen": {
      "description": "Beginning of the attack",
      "disable_correlation": true,
      "misp-attribute": "datetime",
      "ui-priority": 0
    },
    "ip-dst": {
      "categories": [
        "Network activity",
        "External analysis"
      ],
      "description": "Destination IP (victim)",
      "misp-attribute": "ip-dst",
      "ui-priority": 1
    },
    "ip-src": {
```

```
      "categories": [
        "Network activity",
        "External analysis"
      ],
      "description": "IP address originating the attack",
      "misp-attribute": "ip-src",
      "ui-priority": 1
    },
    "last-seen": {
      "description": "End of the attack",
      "disable_correlation": true,
      "misp-attribute": "datetime",
      "ui-priority": 0
    },
    "protocol": {
      "description": "Protocol used for the attack",
      "misp-attribute": "text",
      "ui-priority": 0,
      "values_list": [
        "TCP",
        "UDP",
        "ICMP",
        "IP"
      ]
    },
    "src-port": {
      "categories": [
        "Network activity",
        "External analysis"
      ],
      "description": "Port originating the attack",
      "misp-attribute": "port",
      "ui-priority": 0
    },
    "text": {
      "description": "Description of the DDoS",
      "disable_correlation": true,
      "misp-attribute": "text",
      "ui-priority": 0
    },
    "total-bps": {
      "description": "Bits per second",
      "misp-attribute": "counter",
      "ui-priority": 0
    },
    "total-pps": {
      "description": "Packets per second",
      "misp-attribute": "counter",
      "ui-priority": 0
    }
  },
  "description": "DDoS object describes a current DDoS activity from a
    specific or/and to a specific target. Type of DDoS can be attached to
    the object as a taxonomy",
  "meta-category": "network",
  "name": "ddos",
```

```
  "requiredOneOf": [
    "ip-dst",
    "ip-src",
    "domain-dst"
  ],
  "uuid": "e2f124d6-f57c-4f93-99e6-8450545fa05d",
  "version": 6
}
```

Listing 7.1: Objecto MISP para DDoS