Diseño y desarrollo de un sistema colaborativo para la prevención de ataques coordinados

Joaquín García, Sergio Castillo, Guillermo Navarro, Joan Borrell
Dept. de Informàtica,
Universitat Autònoma de Barcelona,
{jgarcia,scastillo,gnavarro,jborrell}@ccd.uab.es

Resumen En este artículo presentamos el diseño de un sistema colaborativo para la compartición de alertas entre nodos cooperativos a través de una infraestructura multicast segura. El propósito de este sistema es prevenir la utilización de los recursos de nuestra red para la realización de ataques coordinados contra terceras partes. A través de un esquema descentralizado basado en paso de mensajes, los distintos nodos del sistema colaborarán para detectar y prevenir su participación en la realización de un ataque coordinado. Presentamos también en este artículo el desarrollo de una primera implementación de la arquitectura descrita. Dicha implementación, desarrollada para sistemas GNU/Linux, va a demostrar la practicabilidad de nuestra propuesta.

1. Introducción

A pesar de los avances actuales en cuanto a seguridad en redes, tales como sistemas cortafuegos, mecanismos de autenticación y sistemas para la detección de intrusos, los sistemas conectados a Internet nunca han sido tan vulnerables. La proliferación de dispositivos con acceso a Internet, el incremento en la movilidad de dichos dispositivos y su utilización por parte de aplicaciones basadas en red hace que las infraestructuras tradicionales de seguridad en redes deban enfrentarse a una nueva generación de ataques informáticos. Generalmente, estos ataques suelen iniciarse con la intrusión en una red vulnerable mediante el compromiso de alguno de sus equipos, tras lo cual el atacante podrá realizar toda una serie de acciones contra el propio sistema o incluso contra otras redes. Como se apunta en [3], una vez que estos recursos han sido comprometidos, pueden actuar como lanzadera para la ejecución de nuevos ataques (generalmente, de mayor complejidad) contra terceras partes a no ser que los responsables de dichos recursos se hayan encargado de desarmarlos correctamente.

La utilización de técnicas distribuidas y coordinadas en estos ataques son cada vez más usuales entre la comunidad de atacantes, ya que aumentan las posibilidades de ejecución de tareas más complejas, tales como la realización de denegaciones de servicio distribuidas, exploraciones de puertos coordinadas, etc. La detección y prevención de este tipo de ataques no siempre es posible mediante la monitorización y recogida de información aislada desde los distintos equipos de la red. En la mayoría de los casos será necesaria una visión global de los eventos y acciones que se producen en el sistema. Por ello, la prevención y detección ante estos ataques requerirá una recogida de información específica que se encuentra repartida por los distintos equipos de la red a vigilar, y la correlación de tal información. Dentro de esta recogida de información deberemos poder obtener todos aquellos eventos y acciones que preceden a la realización del ataque, tales como establecimiento de conexiones sospechosas, ejecución de procesos anómalos, adición de información en el sistema de ficheros, cambios repentinos en el tráfico de la red, etc.

De acuerdo con [6], podemos definir el término *ataque* como una combinación de acciones ejecutadas por un adversario malicioso con el objetivo de violar la política de

seguridad del equipo o red que son blanco del ataque. Por otro lado, podemos definir el proceso de *detección* de un ataque como la secuencia de acciones elementales que deberán ser ejecutadas con el objetivo de identificar y responder contra dicho ataque. Los *sistemas de detección de intrusos* (*Intrusion Detection Systems* o *IDS*) son uno de los componentes más importantes para la realización de tal proceso. Como [9] apunta, un IDS debe cumplir una serie de requerimientos tales como *precisión* (no debe confundir acciones legítimas con acciones deshonestas), *rendimiento* (el sistema debe ser capaz de poder realizar el proceso de detección a tiempo real), *completitud* (el sistema no debe dejar escapar ataques), *tolerancia a fallos* (debe garantizar su funcionamiento aún en el caso de ser atacado él mismo) y *escalabilidad* (debe ser capaz de realizar el proceso de detección con el mayor número de eventos posible sin que se produzca pérdida de información).

En este artículo presentamos el diseño y desarrollo de un sistema de detección de intrusos que proporciona una solución descentralizada para prevenir la utilización de recursos de red en ataques coordinados contra terceras partes. El sistema se basa en un conjunto de entidades cooperativas (llamadas *celdas de prevención*) que son albergadas en cada uno de los recursos de red que queremos desarmar. Estas entidades colaborarán entre sí para detectar si el equipo donde se encuentran instaladas participa de forma activa en la realización de un ataque coordinado. En caso de detectar tal situación, las celdas de prevención actuarán de la forma apropiada sobre cada uno de sus recursos asociados para, finalmente, eludir su participación en el ataque detectado. Así, la principal diferencia entre nuestra propuesta frente a sistemas de detección similares es que cada nodo que alberga una celda de prevención está a la espera de ser el origen o formar parte de un ataque coordinado, no la víctima del mismo.

La mayoría de los sistemas actuales para la detección y prevención de ataques coordinados se basan en la utilización de técnicas centralizadas o jerarquizadas, presentando problemas tradicionales de *cuellos de botella* y de tolerancia a fallos. Por ello, nuestro sistema ha sido diseñado de acuerdo con las últimas propuestas presentadas por la comunidad de investigación en el campo de la detección de intrusiones que tratan de solucionar tales inconvenientes. Adicionalmente, el desarrollo de nuestra propuesta para sistemas GNU/Linux se ha basado en herramientas ya existentes, lo que nos ha permitido una rápida y eficiente implementación de la arquitectura propuesta en este artículo.

El resto del documento está organizado de la siguiente manera. La sección 2 presenta algunos trabajos relacionados con la detección y prevención de ataques distribuidos y coordinados, cuyas aportaciones y diseños han sido utilizados como punto de partida. El diseño de nuestro sistema es presentado en la sección 3, y una primera implementación en la sección 4. La utilización de nuestro sistema en un escenario real es descrito en la sección 5. Las conclusiones y el trabajo futuro se podrán encontrar en la última sección.

2. Antecedentes

Actualmente existe un gran número de publicaciones relacionadas con el diseño de sistemas de detección y prevención contra ataques coordinados y distribuidos. A pesar de las diferencias existentes entre los distintos trabajos y mecanismos propuestos para conseguir cumplir los requerimientos apuntados por [9], podemos identificar los siguientes componentes en cualquiera de las arquitecturas de IDS existentes:

- *Sensores*, encargados de recoger información del sistema o de la red que están vigilando. Es posible identificar dos tipos básicos de sensores según dónde se encuentren instalados. Por un lado, tenemos aquellos sensores basados en la recogida de información a nivel de equipo. Esta primera categoría, que se conoce como *host based IDS* o *HIDS*, está a la expectativa de que se produzcan eventos a nivel de sistema operativo tales como llamadas al sistema sospechosas, ejecución de procesos no permitidos, etc. El segundo tipo de sensores, conocido como *network based IDS* o *NIDS*, realizan una recogida de información a nivel de red como, por ejemplo, información referente a los datos contenidos en los datagramas IP que pasan por la red que vigilan.
- *Analizadores*, que procesan y ponen en correspondencia (o correlacionan) la información que ha sido recogida por los sensores asociados. A través de este proceso, se encargarán de inferir posibles ataques. Podemos identificar dos clases básicas de analizadores dependiendo del tipo de detección que realicen. En primer lugar, aquellos analizadores que basan su proceso de detección en la utilización de una información a priori de secuencias y actividades de ataque ya conocidas. Este tipo de detección se conoce como detección basada en usos indebidos (*misuse based intrusión detection*). La segunda categoría, conocida como detección basada en anomalías (*anomaly based intrusion detection*), trata de identificar actividad maliciosa comparando la información recogida por los sensores contra la representación de actividades consideradas como normales.
- *Unidades de respuesta*, encargadas de iniciar acciones para neutralizar la intrusión o los ataques una vez detectados. Estas acciones pueden ser automáticas, actuando contra el atacante mientras la intrusión o el ataque está aún en marcha, o bien pueden requerir la intervención de los administradores de la red, limitándose, por ejemplo, al lanzamiento de alarmas o notificaciones para avisar al responsable del equipo o de la red.
- *Gestores*, responsables de ejecutar funciones de más alto nivel tales como configuración de analizadores y sensores, notificación de alertas y contramedidas, consolidación y almacenamiento de información, generación de informes, etc. La implementación de un gestor de base de datos, por ejemplo, permitirá al sistema de detección almacenar de forma correcta toda la información generada por el sistema para su correcto tratamiento durante diferentes periodos de tiempo.

En [17] podemos encontrar un buen número de referencias y revisiones sobre como combinar los elementos anteriores para la detección de ataques distribuidos. La mayor parte de estos trabajos han sido concebidos como sistemas centralizados o jerárquicos, presentando toda una serie de problemas relacionados con la saturación del servicio por parte de analizadores centrales o del nivel superior de la jerarquía. Como se apunta en [2], sistemas como DIDS [27] y NADIR [15], que basan la correlación de la información en elementos dedicados, presentan fuertes restricciones en cuanto a escalabilidad y eficiencia. Éstos se encargan de almacenar toda la información recogida por el sistema en un punto central para proceder a la ejecución del proceso de detección. Por otro lado, la utilización de esquemas jerárquicos en proyectos como Emerald [24], GrIDS [28], AAFID [2] y NetSTAT [30] continúan utilizando nodos dedicados que actúan como puntos centrales de recogida de información. Aunque estos esquemas mejoran la escalabilidad respecto a esquemas centralizados, siguen presentando deficiencias similares y son vulnerables a problemas de saturación de servicio por parte de los nodos superiores de la jerarquía.

En contra de estos esquemas tradicionales, algunas propuestas alternativas, como Micael [8], IDA [1], Sparta [19] y MAIDS [14] tratan de utilizar el paradigma de los agentes móviles para realizar el proceso de detección distribuido. Puesto que las distintas evidencias del ataque se encuentran repartidas sobre los diferentes equipos del origen y del destino del ataque, la idea de utilizar un proceso de detección distribuido en forma de agentes móviles tiene como principales ventajas una sobrecarga mínima del sistema, así como la posibilidad de consumir recursos únicamente en el lugar donde los agentes se van a ejecutar.

Aunque la tecnología de agentes móviles parece tener un buen número de prometedoras características para la implementación de una arquitectura descentralizada para la detección de ataques coordinados, su utilización en la mayoría de los casos estudiados parece acabar siendo innecesaria y contraproducente. Además, la utilización de los agentes en la mayoría de los diseños anteriores se limita al transporte de la información. Esta funcionalidad, que puede ser desempeñada de una forma más eficiente mediante un esquema de paso de mensajes, no justifica por sí sola la necesidad de utilizar un paradigma como el de los agentes móviles. Por contra, su utilización introduce una serie de riesgos de seguridad adicionales y una penalización en la ejecución global del sistema.

Algunos diseños basados en paso de mensajes, como por ejemplo CSM [32], Quicksand [18] e Indra [16], tratan de eliminar la necesidad de utilizar elementos dedicados por medio de la introducción de entidades de trabajo distribuidas por los diferentes equipos involucrados. Tales entidades tratan de realizar tareas equivalentes. Para ello, compartirán la información de sus respectivos procesos de detección mediante la utilización de una infraestructura de comunicación común que permitirá la realización de un algoritmo de correlación descentralizado. La principal ventaja de este nuevo diseño es la inexistencia de puntos de fallo aislados o cuellos de botella en el sistema.

3. Sistema de celdas de prevención

En esta sección presentamos el diseño de un sistema cuyo principal propósito es la detección y prevención de ataques coordinados. Por medio de la utilización de un conjunto de entidades cooperativas, el sistema será capaz de evitar la participación de los recursos donde dichas entidades están siendo albergadas para la realización de un ataque coordinado.

El diseño de nuestro sistema tiene dos objetivos principales. El primero de ellos es obtener una arquitectura modular compuesta por un conjunto de entidades que cooperen. Estas entidades colaborarán entre sí para detectar si el equipo donde están instaladas participa de forma activa en la elaboración de un ataque coordinado contra la red donde están conectados o contra una red de terceras partes. Una vez el ataque, o una de sus acciones, haya sido detectado, estas entidades deberán ser capaces de prevenir la utilización de los recursos asociados para evitar su participación en el ataque detectado.

El segundo objetivo es conseguir una relación completa e independiente entre los diferentes componentes que conforman cada una de estas entidades cooperativas. De esta forma, será posible distribuir tales componentes de acuerdo con las necesidades de cada recurso que queramos desarmar. El resto de esta sección ha sido organizada de la siguiente manera. Primero, presentamos las características básicas de la arquitectura de comunicaciones presente en el diseño y el modelo utilizado para tal diseño. A continuación, mostramos los elementos que conforman los diferentes nodos de tal arquitectura.

Finalmente, presentamos algunos lenguajes y mecanismos de correlación que han sido estudiados para la realización del diseño de nuestra propuesta.

3.1. Arquitectura de comunicaciones *multicast*

Para alcanzar el primer objetivo señalado más arriba, proponemos la utilización de una arquitectura *multicast* para la comunicación entre las distintas entidades de nuestro sistema. A través de esta arquitectura de comunicaciones, cada una de las distintas entidades (*celdas de prevención*) intercambiará con el resto un conjunto de mensajes cooperativos para participar en la realización de un proceso de detección descentralizado (tal y como muestra la figura 1). Para la realización de esta comunicación multicast, proponemos la utilización de un modelo *publicador-suscriptor* (*publish-subscribe model*). A través de este modelo, los distintos componentes de cada celda de prevención serán capaces de producir y consumir mensajes del bus multicast compartido.

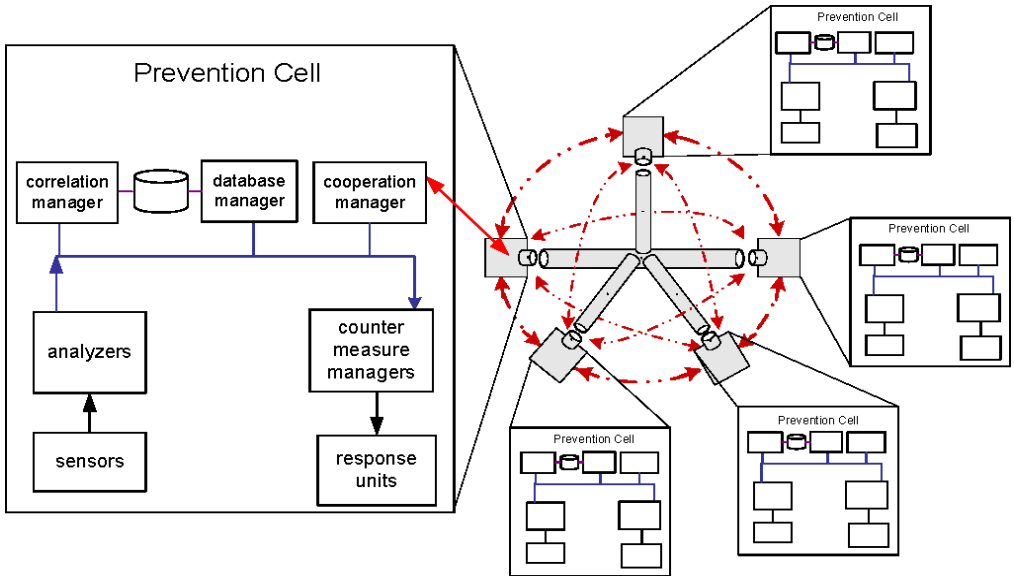

Figura 1. Arquitectura de paso de mensajes basada en celdas de prevención

De acuerdo con [13], en un sistema *publicador-suscriptor* los distintos componentes producen mensajes y anuncian (o publican) tales mensajes en un bus compartido. Otros componentes podrán escuchar (o suscribirse a) los mensajes producidos. Una vez recibidos, la información contenida en ellos será consumida por estos componentes. Dichos componentes pueden ser objetos, procesos, servidores, aplicaciones, herramientas o cualquier otro tipo de instancias ejecutadas en el sistema. Los mensajes, o eventos, intercambiados entre estos componentes, pueden ser simples cadenas de texto o complejas estructuras de datos. La característica clave de este modelo es la posibilidad que cualquier componente pueda suscribirse a un canal y empezar a recibir información, sin necesidad de conocer el nombre, o ni tan siquiera la existencia, de los componentes que producen dicha información. Así pues, algunas ventajas inmediatas para nuestra propuesta mediante la utilización de un modelo *publicador-suscriptor* son la relativa facilidad de añadir o eliminar

componentes al sistema, así como la introducción de nuevos tipos de mensajes o la modificación del conjunto de elementos a producir y consumir información.

3.2. Celdas de prevención

Teniendo en cuenta las ventajas de un modelo *publicador-suscriptor* que se han discutido anteriormente, este modelo es también de gran utilidad para alcanzar la independencia entre componentes que anunciábamos más arriba como nuestro segundo objetivo. Así pues, proponemos también la utilización de un modelo *publicador-suscriptor* para la relación entre los componentes internos de cada celda de prevención. Todos estos componentes han sido diseñados de acuerdo con los elementos básicos de cualquier IDS que enumerábamos en la sección 2 (sensores, analizadores, gestores y unidades de respuesta). Los mensajes intercambiados entre estos componentes serán básicamente tres: *eventos* (entre sensores y analizadores), *alertas* (entre analizadores y gestores), y *acciones* (entre gestores y unidades de respuesta). Estos componentes, y los distintos mensajes intercambiados entre ellos, se muestran en la figura 2 y son descritos a continuación.

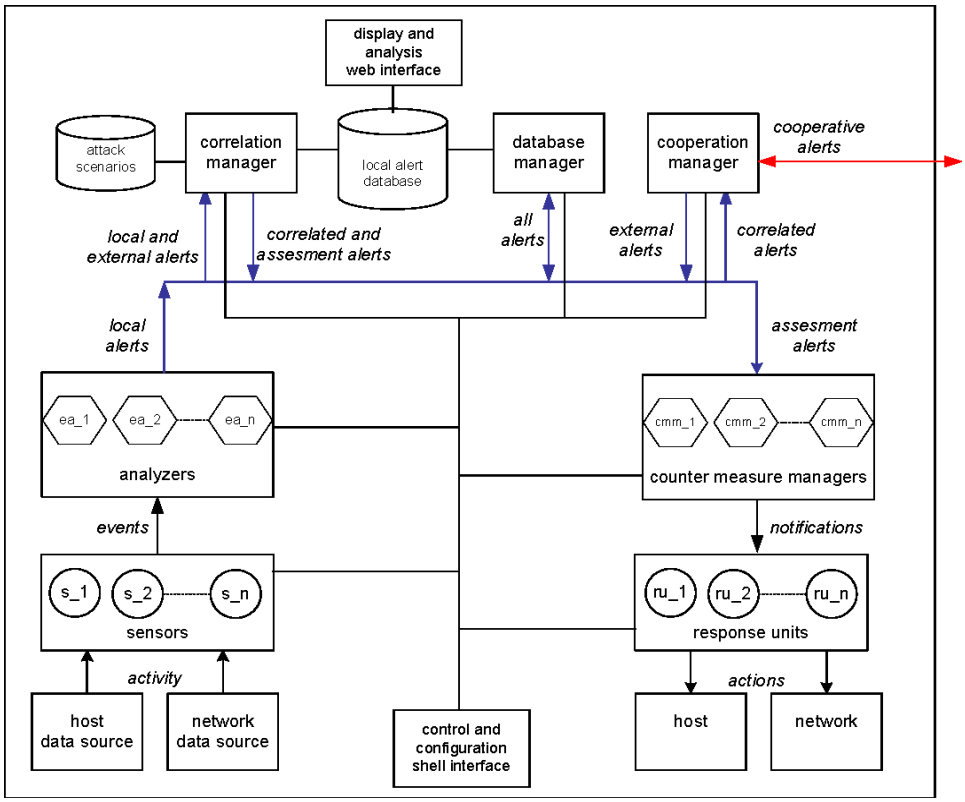

Figura 2. Esquema de una celda de prevención

- *Sensores*, encargados de obtener información sospechosa, tanto a nivel de red como a nivel del propio equipo donde se encuentran instalados, para publicarla en el canal específico (donde algunos analizadores asociados estarán a la escucha). Proponemos la utilización de sensores basados en red (en busca de tráfico ilegal o peligroso como, por ejemplo, desbordamientos de buffer, datagramas con suplantación de IP,

inundación de tráfico, etc) y de sensores basados en equipo (en busca de acciones sospechosas o peligrosas originadas en el propio equipo y que pueden suponer un riesgo como, por ejemplo, utilización abusiva de la CPU, ejecución de comandos potencialmente hostiles, etc.).

- *Analizadores*, responsables de recibir los eventos publicados por los sensores del sistema para realizar con ellos un proceso de análisis y de correlación de bajo nivel. Así, estos componentes consumirán eventos y producirán alertas locales en el interior de cada celda de prevención. Estas alertas serán publicadas en el canal correspondiente (en este caso, en el canal de alertas locales). Proponemos el uso tanto de analizadores con detección basada en usos indebidos como de analizadores con detección basada en anomalías.
- *Gestor de correlación*, que estará a la escucha de alertas locales y externas a través de los canales correspondientes, para consumir dicha información y contrastarla contra sus escenarios de ataque asociados. Este componente realizará un proceso de correlación de alto nivel. Estará, por tanto, involucrado en la parte del proceso de correlación relativa a la celda de prevención donde se encuentra albergado. Puesto que la correlación de un ataque coordinado es un conjunto de alertas internas y externas reportadas por elementos internos y externos del sistema, todas estas alertas serán analizadas en un proceso de correlación de alto nivel ejecutado por el gestor de correlación. Este elemento será también el responsable de publicar alertas de correlación y alertas de contramedida.
- *Gestor de base de datos*, que estará a la escucha de cualquier canal de la celda de prevención donde se encuentra instalado. De esta forma, cada celda de prevención contará con una base de datos local donde podremos encontrar todas las alertas que se han generado en el interior de la celda (incluyendo alertas cooperativas que hayan llegado del exterior). Como podemos ver en la figura 2, esta base de datos local tendrá, en cada celda de prevención, una interfaz web que servirá como herramienta administrativa para la visualización y análisis de alertas.
- *Gestor de cooperación*, que estará a la escucha de alertas cooperativas provenientes del exterior de la celda de prevención donde se encuentra instalado, y publicará alertas externas en su interior. A la vez, consumirá las alertas de correlación producidas en el interior de la celda de prevención donde se encuentra, y las publicará en el exterior en forma de alertas cooperativas.
- *Gestores de contramedidas*, encargados de recibir alertas de contramedida publicadas por el gestor de correlación de la celda de prevención que les alberga. Estos gestores son los responsables de consumir contramedidas y transformarlas en las acciones correspondientes. Una vez la alerta de contramedida es transformada, las acciones asociadas serán publicadas en el canal asociado a las unidades de respuesta de la celda de prevención.
- *Unidades de respuesta*, que tomarán las acciones producidas por los gestores de contramedidas para llevarlas a cabo. Cada acción habrá sido generada para prevenir uno de los distintos pasos del ataque coordinado que ha sido detectado, y será ejecutada contra el recurso donde la celda de prevención se encuentra alojada. Al igual que los sensores, proponemos el uso de unidades de respuesta basadas es red (para, por ejemplo, bloquear conexiones, cerrar puertos TCP/UDP, etc.) y unidades de respuesta basadas en equipo (para, por ejemplo, finalizar la ejecución de procesos, realizar un cambio de permisos, bloquear cuentas de usuario, etc).

3.3. Correlación de alertas

Para la realización del proceso de correlación de las alertas generadas por los diferentes analizadores y gestores del sistema al completo, cada celda de prevención contendrá un gestor de correlación encargado de realizar el proceso de correlación a alto nivel. Para poder realizar correctamente su tarea, el gestor de correlación necesitará tres elementos principales: un lenguaje para poder modelar y describir escenarios de ataque, algunos mecanismos para la correlación de alertas contra los escenarios modelados, y un lenguaje para poder describir las alertas que deben ser correladas.

Existe un gran número de propuestas de lenguajes para modelar escenarios de ataque. Generalmente, cada IDS posee su propio lenguaje, el cual está fuertemente relacionado con las técnicas y mecanismos que dichos sistemas utilizarán para realizar su proceso de detección. Algunos ejemplos de estas combinaciones de lenguajes y mecanismos de correlación son, por ejemplo, P-BEST (*Production-Based Expert System Toolset*), un lenguaje de detección basado en reglas para el sistema Emerald [24], el lenguaje STATL, un lenguaje basado en transición de estados utilizado por el sistema NetSTAT [30], o el lenguaje ASL (*Attack Specification Language*), un lenguaje basado en descripción de patrones utilizado por Quicksand [17]. Otros lenguajes, como por ejemplo LAMBDA [6], utilizado por la arquitectura cooperativa propuesta en [5] (CRIM), y el lenguaje del sistema CARDS [22], son ejemplos de lenguajes de detección algebraicos. Estos lenguajes no tan solo son válidos como lenguajes de detección sino que son de utilidad para describir muchos otros aspectos de los ataques al permitir una alta abstracción y simplificar la especificación de firmas de ataque. Finalmente, el uso de formalismos para el modelado de sistemas dinámicos, como por ejemplo *chronicles*, han sido también propuestos para la ejecución de la correlación de alertas en el entorno M2D2 [21].

De todas estas combinaciones de lenguajes de especificación y mecanismos de correlación, nuestra propuesta basará el proceso de correlación de alto nivel en el uso de CRIM, y modelará los escenarios de ataque asociados utilizando LAMBDA. LAMBDA es un lenguaje para la descripción de ataques basado en lógica. Los distintos pasos en los escenarios descritos por LAMBDA representarán las acciones realizadas por un posible atacante (tanto si son observables como si no). Como se apunta en [4], el mismo formalismo utilizado por LAMBDA para modelar los distintos pasos de un ataque coordinado podrá ser utilizado para indicar las contramedidas necesarias. De esta manera, LAMBA permitirá tanto la especificación del concepto de correlación como del proceso de anti-correlación (necesario para especificar qué contramedidas deberá realizar el sistema).

Por otro lado, CRIM es un mecanismo flexible para la correlación de alertas a través del tratamiento de las consecuencias y prerequisitos de los ataques. Este módulo de correlación implementa las funciones necesarias para el agrupamiento de las distintas alertas que pertenecen a una misma instancia de ataque. Proporciona también funciones para el mezclado y creación de alertas correladas para representar la información contenida en las alertas pertenecientes al mismo grupo. Así, estas funciones para la agrupación, mezclado y creación de alertas de correlación serán utilizadas para correlar todas las alertas producidas tanto en el interior como en el exterior de las celdas de prevención con el objetivo de reconocer intentos de ataques coordinados y prevenirlos. Por lo tanto, la utilización no se limitará tan solo a la detección de ataques, sino que la posibilidad de utilizar el proceso de anti-correlación ejecutado por CRIM permitirá lanzar las alertas de contramedidas correspondientes para detener dicho ataque. Estas alertas serán escuchadas por los gestores de contramedidas que se encargarán de transformarlas en las acciones

correspondientes, que a su vez, serán escuchadas por las unidades de respuesta para detener la ejecución del ataque coordinado.

Una vez decidido el lenguaje para el modelado de los escenarios de ataque y el mecanismo para la correlación de las alertas, el gestor de correlación propuesto necesitará algún lenguaje o formato para especificar la información contenida en las alertas. Aunque la mayoría de los productos propietarios basan el formato de sus alertas en soluciones propias, la creación de un formato común de alertas ha sido propuesto recientemente por varios miembros de la comunidad de la detección de intrusos. El *Common Intrusion Specification Language* (*CISL*) [12], por ejemplo, fue propuesto unos años atrás para permitir que los componentes del *Common Intrusion Detection Framework* (*CIDF*) pudiesen intercambiar la información de una forma semánticamente bien definida. Otros lenguajes, con el mismo propósito, son el *Intrusion Detection Message Exchange Format* (*IDMEF*) [7], propuestos por el *Detection Exchange Format Working Group* (*IDWG*) del *IETF*, y el *Security Device Event Exchange* (*SDEE*) [11], propuesto por el *Intrusion Detection Systems Consortium* (*IDSC*).

De entre estos tres lenguajes, hemos escogido el formato IDMEF para la comunicación entre los componentes de nivel superior (analizadores y gestores). En primer lugar porque este lenguaje es la base para el operador de similitud utilizado en el mecanismo de agregación y fusión de CRIM. Este formato permite también una excelente especificación de alertas generadas a partir tanto de sensores basados en red como de sensores basados en equipo. A parte, es un formato extensible y ofrece la posibilidad de incorporar información adicional en el interior de las alertas. Por otro lado, existe un gran número de herramientas e implementaciones de IDS actuales basados en el formato IDMEF, lo que hace más sencillo la integración de tales aportaciones dentro de nuestro sistema. Así, la combinación de todos estos componentes para el intercambio de mensajes en nuestro sistema será compatible con el marco de trabajo para la detección de intrusiones propuesto por el IDWG.

En el momento de escribir este artículo, nos encontramos finalizando la implementación de un primer prototipo de aplicación para el sistema *publicador-suscriptor* propuesto y los distintos componentes que se han visto en esta sección. La implementación de este prototipo es comentada en la siguiente sección.

4. Desarrollo actual

Esta sección muestra el desarrollo de una plataforma que implementa la arquitectura colaborativa presentada en este artículo. Dicha plataforma (ver figura 3) está siendo implementada para sistemas GNU/Linux en lenguaje C y C++. La naturaleza de este sistema operativo en cuanto a lo que a red se refiere, código abierto y documentación, ha facilitado el análisis de requerimientos y el desarrollo.

La implementación de la plataforma ha sido probada con distintas versiones de la serie 2.4.x de Linux y con las versiones 2.9.x y 3.x del compilador `gcc` de GNU. Todos los componentes de la plataforma, excepto el motor de correlación, han sido programados en lenguaje ANSI C. El motor de correlación, basado en los módulos de fusión y correlación de CRIM y el parser de LAMBDA desarrollados por Fabien Autrel y Thierry Sans del ENST Bretagne, están siendo integrados en nuestro sistema en C++.

La principal diferencia entre nuestra propuesta frente a sistemas de detección similares es que cada nodo que alberga una celda de prevención está a la espera de ser el

origen de un ataque coordinado (o alguna de las acciones relacionadas con dicho ataque), no la víctima del mismo. Este echo implica algunas consideraciones en el análisis de requerimientos de los sensores y las unidades de respuesta. En primer lugar, el número de estos elementos debe ser lo suficientemente representativo para detectar y reaccionar contra los distintos pasos de los escenarios de ataque que el sistema conoce. En segundo lugar, tanto analizadores como gestores de contramedidas necesitarán una comunicación suficientemente rápida con sensores y unidades de respuesta. Es decir, esta velocidad ha de ser suficiente para poder recoger todos los eventos y proporcionar todas las respuestas posibles.

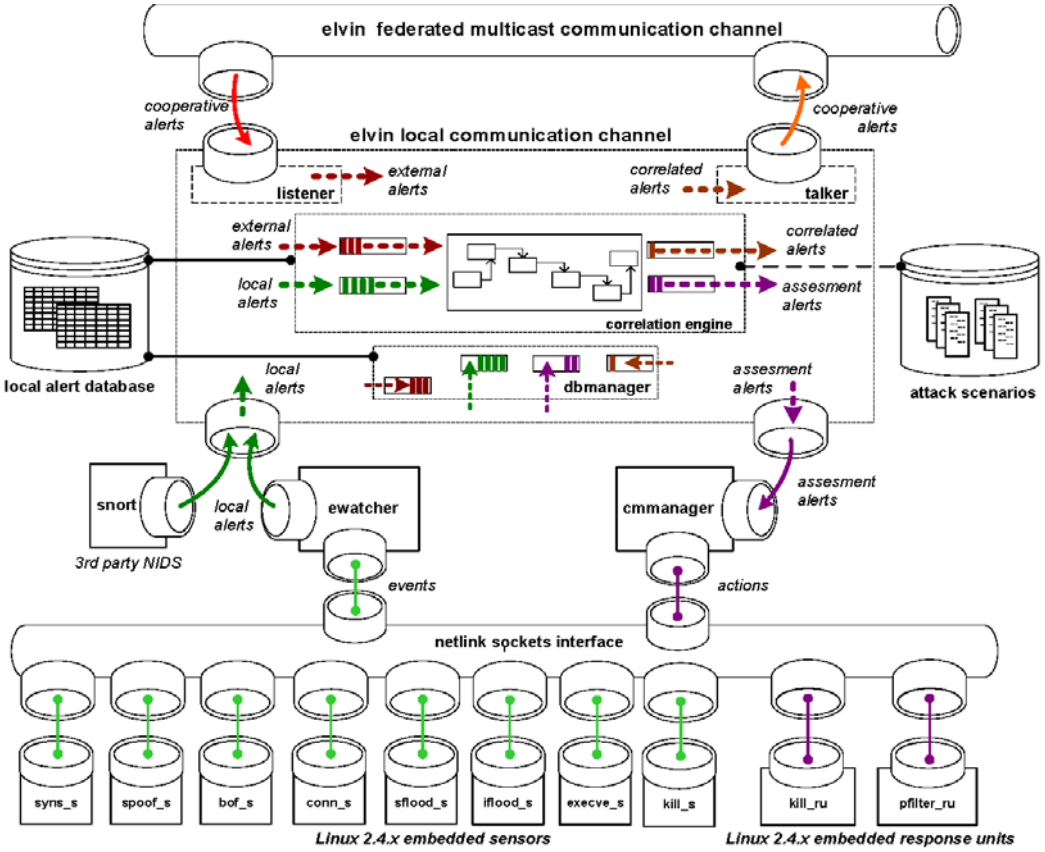

Figura 3. Esquema del desarrollo actual

Sensores y unidades de respuesta Con el objetivo de conseguir los requerimientos anteriores, el desarrollo de nuestra plataforma se inició con el diseño e implementación de un conjunto de sensores y unidades de respuesta empotrados en el propio kernel del sistema operativo GNU/Linux en forma de módulos. Aún así, el uso de sensores y unidades de respuesta de terceras partes es posible en nuestro sistema (tal y como muestra la figura 3 con el uso del NIDS *Snort*). La utilización de sensores y unidades de respuesta en forma de módulos de kernel ofrece a nuestro sistema toda una serie de ventajas. En primer lugar, la posición de privilegio de los módulos del kernel dentro del sistema operativo proporciona un acceso directo a cualquier información necesaria de una forma eficiente y de total confianza. Por otro lado, la carga producida por el intercambio de información entre el

espacio de kernel y el espacio de usuario se reduce, transfiriendo únicamente aquella información necesaria en el momento en que un evento se ha producido. Como resultado de estas ventajas, la capacidad de proceso a la hora de analizar patrones o comportamientos (por ejemplo, datagramas IP o comandos ejecutados) puede maximizarse con facilidad.

El comportamiento y el estado tanto de sensores como de unidades de respuesta puede ser modificado dinámicamente dependiendo de los parámetros que gobiernan sus algoritmos. A través de alertas IDMEF, por ejemplo, el gestor de contramedidas puede modificar el comportamiento de alguna de las unidades de respuesta añadiendo nuevas reglas de filtrado o añadiendo nuevos procesos a eliminar. El objetivo de este comportamiento es conseguir que la propia plataforma pueda adaptarse por sí misma en función del estado interno cada celda de prevención o de la clase de ataque coordinado que se pretenda prevenir. Adicionalmente, para facilitar el acceso a estos elementos (para tareas de chequeo o de *debugging*, por ejemplo), tanto sensores como unidades de respuesta crean una entrada en el sistema de ficheros *procfs*. Este sistema de ficheros (*proc file system*, *procfs*) es un sistema de ficheros especial en el kernel Linux que permite acceder desde el espacio de usuario a estructuras de información dentro del espacio de kernel. En nuestro caso, tanto sensores como unidades de respuesta ofrecen la posibilidad de ser activados o desactivados a través de este sistema de ficheros virtual.

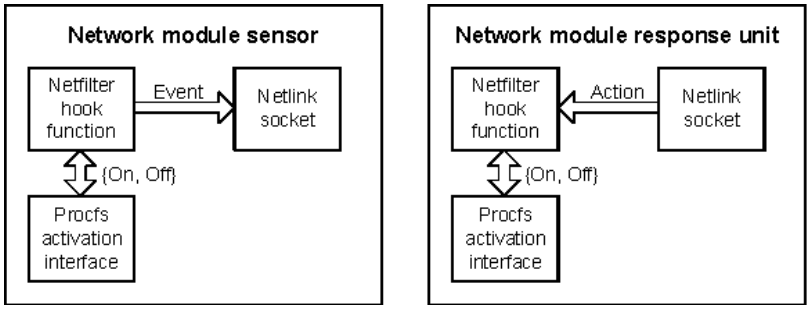

Figura 4. Sensores y unidades de respuesta basadas en red

La implementación de los sensores de red y de las unidades de respuesta de red (ver figura 4) se basa en la utilización del subsistema netfilter [31], un marco de trabajo para la manipulación de paquetes en el kernel Linux. Netfilter permite tanto el filtrado de paquetes como la traducción de direcciones de red y otras manipulaciones más complejas. Actualmente se está trabajando en el desarrollo de los sensores y unidades de respuesta descritos en la tabla 1: un sensor para la detección de exploraciones de puerto silenciosas (*syns_s*), un sensor para la detección de IP *spoofing* (*spoof_s*), un sensor para la detección de *buffer overflows* en paquetes de red (*bof_s*), un sensor para la detección de conexiones TCP que puede ser utilizado para inferir cadenas de conexiones TCP (*conn_s*), dos sensores para detectar ataques de denegación de servicio (DoS) basadas en SYN e ICMP *flooding* (*sflood_s* y *iflood_s*) y, finalmente, una unidad de respuesta capaz de realizar filtrado de paquetes (*pfilter_ru*).

La implementación de los sensores de equipo propuestos (ver tabla 1) se basa en la captura de algunas llamadas al sistema con el propósito de obtener información de interés en la búsqueda de actividades ilícitas o sospechosas. Por otro lado, la implementación de las unidades de respuesta utiliza la misma idea proporcionando los mecanismos necesarios para prevenir las acciones necesarias asociadas con los distintos pasos del ataque a

prevenir. En el momento de escribir este artículo, se está trabajando en la implementación de un sensor para monitorizar la ejecución de comandos ilícitos o sospechosos (*execve_s*), un sensor para detectar qué procesos van a ser eliminados (*kill_s*) y una unidad de respuesta capaz de eliminar y proteger procesos del sistema (*kill_ru*).

Sensor	**Descripción**
`syns_s`	Detección de exploraciones de puerto silenciosas monitorizando el protocolo de establecimiento de sesión TCP.
`Spoof_s`	Detección de IP spoofing en busca de paquetes locales cuya dirección IP de origen no esté asociada a ninguna interfaz de red.
`bof_s`	Detección de Buffer overflows basada en ejecución abstracta del campo de datos del paquete [29].
`conn_s`	Detección de establecimientos de sesión monitorizando el protocolo de establecimiento de sesión de TCP. Servirá para inferir cadenas de conexiones.
`sflood_s`	Detección de ataques DoS basados en SYN flooding.
`iflood_s`	Detección de ataques DoS basados en ICMP flooding.
`execve_s`	Monitorización de programas y de comandos de sistema a través de la captura de la llamada al sistema *execve syscall*.
`kill_s`	Detección de cancelación inusual de procesos a través de la captura de la llamada al sistema `KILL`, interceptando las señales `SIGTERM` y `SIGKILL`.
Unidad de respuesta	**Descripción**
`kill_ru`	Cancelación de procesos a través de la llamada al sistema `KILL`. Además, también proporciona una protección de los componentes del sistema (*listener*, *talker*, *correlation engine*, etc.).
`pfilter_ru`	Filtrado de paquetes mediante la utilización de reglas definidas por el componente *cmmanager* y basadas en criterios tradicionales (IP de origen, protocolo, banderas TCP, etc.)

Cuadro 1. Descripción de sensores y unidades de respuesta propuestos

Comunicación de eventos y acciones Como se muestra en la figura 3, los sensores proporcionan eventos al analizador *event watcher* (*ewatcher*) y el gestor de contramedidas (*cmmanager*) provee las acciones a las unidades de respuesta. La complejidad de estos componentes y la limitación que supone trabajar en el espacio de kernel requiere su diseño como procesos demonio (*daemon*) en espacio de usuario. Así, para hacer posible este esquema, es necesario la utilización de mecanismo de comunicación entre espacio de kernel y espacio de usuario.

Entre las diferentes alternativas posibles para comunicar procesos entre el espacio de kernel y el espacio de usuario, hemos optado por la utilización de *netlink sockets* [10] para la comunicación entre los sensores y analizadores, y entre unidades de respuesta y gestores de contramedidas. *Netlink sockets* es un mecanismo específico para el kernel Linux que proporciona un enlace de comunicación bidireccional, no orientado a conexión y asíncrono. Aunque el uso de *netlink sockets* está más enfocado a la implementación de servicios IP, este mecanismo puede ser también utilizado como interfaz estándar para

establecer un enlace de comunicación entre módulos de kernel y procesos en el espacio de usuario. Además, *netlink sockets* se basa en las primitivas para el tratamiento de sockets tradicionales, proporcionándonos transparencia en lo que a mecanismos de *buffering* se refiere.

Analizadores y gestores Tanto la implementación del analizador *ewatcher* y del gestor de contramedidas *cmmanager*, así como la implementación del resto de componentes mostrados en la figura 3 (*listener*, *talker*, *dbmanager* y *correlation engine*), se basa en un mecanismo de *plug-ins*, lo que facilita su desarrollo y mantenimiento. Así, a través de *netlink sockets*, tanto analizadores como gestores de contramedidas consumen o producen información. Pero, para generar la información, o para manejarla, distintos *plug-ins* pueden ser activados o desactivados.

Algunos de estos *plug-ins* son lanzados como threads. El analizador *ewatcher*, por ejemplo, lanza distintos *plug-ins* para manipular los eventos recibidos haciendo uso de un mecanismo *multi-threading*. De esta forma, es posible paralelizar la recogida de los diferentes eventos producidos por el conjunto de sensores del sistema. Otros *plug-ins* como, por ejemplo, el responsable de enviar las acciones a las unidades de respuesta, o el responsable de manejar alertas externas y transformarlas a alertas locales, no necesitan el uso de este mecanismo *multi-threading* para realizar sus trabajo.

Aunque no todos los *plug-ins* son necesarios para el correcto funcionamiento de la plataforma, hay algunos cuya presencia es imprescindible para el funcionamiento del sistema. Es el caso, por ejemplo, del *plug-in* responsable de la generación, análisis y comunicación de las alertas IDMEF. Este *plug-in* se basa en la utilización de la librería *libidmef* [20], una librería en lenguaje ANSI C que, a su vez, se basa en la librería *Libxml* para la generación y el análisis de mensajes IDMEF. La utilización de *libidmef*, además de proporcionar un desarrollo libre, sencillo y eficaz, también hace posible la integración de analizadores y gestores desarrollados por terceras partes en nuestro sistema. El uso, por ejemplo, del NIDS mostrado en la figura 3, basado en un *plug-in* desarrollado con *libidmef*, da una prueba de ello.

Comunicación de alertas La comunicación entre analizadores y gestores, tanto en el interior de cada celda de prevención como con el resto de celdas de prevención de nuestra arquitectura, se realiza gracias a la utilización del software Elvin [26]. Elvin ofrece un entorno sencillo, flexible y seguro para el desarrollo de aplicaciones *publicadorsuscriptor*. Para ello, tanto los analizadores como los gestores de la plataforma han sido desarrollados haciendo uso de *libelvin* y *e4xx*, dos librerías para desarrollar aplicaciones en C y C++, cumpliendo el protocolo de cliente de Elvin. Por otro lado, cada equipo que albergue en su interior una celda de prevención ejecutará un servidor de Elvin para encaminar todas las alertas publicadas en su interior.

Finalmente, para poder compartir las alertas producidas por las distintas celdas de prevención mediante una *comunicación multicast segura*, nuestra plataforma utiliza el protocolo de federación y de comunicación *multicast* proporcionado por Elvin, así como otras interesantes características de seguridad ofrecidas por este sistema, como por ejemplo las opciones criptográficas y de tolerancia a fallos. De esta forma, utilizando SSL a nivel de transporte, se garantiza la confidencialidad, integridad y autenticidad de las alertas cooperativas comunicadas entre las diferentes celdas de prevención.

5. Escenario de ejemplo

En esta sección mostramos la prevención del ataque de Mitnick mediante el uso del sistema de celdas de prevención presentado en este artículo. Aunque se trata de un ataque antiguo, se trata de un excelente ejemplo para mostrar como los componentes de nuestra arquitectura manejan la detección y prevención de un ataque coordinado. Como [25] apunta, este ataque trata de explotar una relación de confianza entre dos equipos para conseguir un acceso remoto ilegal en el equipo de destino haciendo uso de la coordinación de tres técnicas. En primer lugar, mediante la utilización de un ataque de denegación de servicio basado en SYN flooding, el atacante dejará fuera de servicio el equipo de confianza para que no pueda transmitir y pueda evitar así el tercer paso del ataque. En segundo lugar, una predicción de números de secuencia TCP se utilizará para obtener el siguiente número de secuencia TCP que el equipo víctima utilizará en la siguiente conexión que se realice. Finalmente, un establecimiento de sesión remota, haciendo uso de una suplantación de la dirección IP de origen del equipo de confianza (que se encuentra fuera de servicio) y la utilización del número de secuencia obtenido con la predicción anterior, tratará de concluir el ataque.

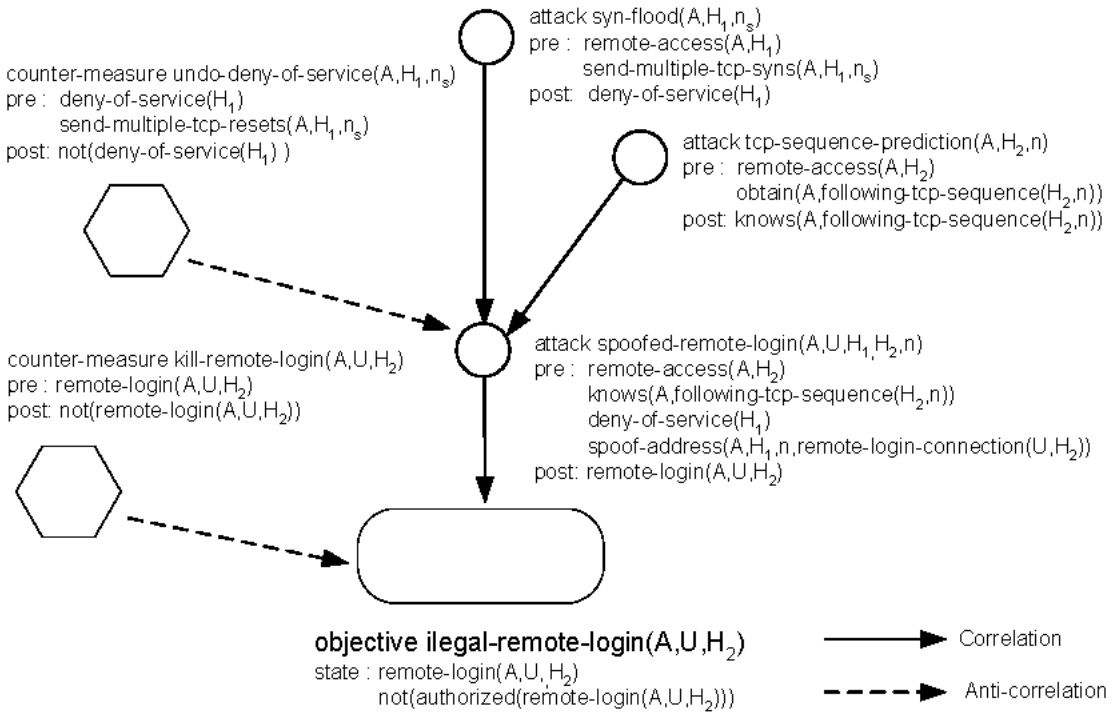

Figura 5. Grafo de correlación y anti-correlación para el ataque de Mitnick

Un posible grafo de correlación y anticorrelación [4] para este ataque se muestra en la figura 5. En el primer paso de este modelo, *A* (el agente que ejecuta el ataque) realiza una inundación de tráfico SYN TCP contra el equipo H_1. En el segundo paso, *A* realizará una predicción de números de secuencia TCP contra el equipo H_2 parao obtener el siguiente número de secuencia TCP que este equipo utilizará. Finalmente, mediante este número de secuencia, *A* iniciará la sesión remota contra H_2 suplantando la identidad del equipo H_1. Ya

que H_1 se encuentra fuera de servicio, será incapaz de cancelar la conexión en el instante en que H_2 le envié la confirmación del inicio de conexión. Si este segundo paso se realiza con éxito, A habrá establecido una sesión remota ilegal como usuario administrador en el equipo H_2.

El modelo de la figura 5 propone dos contramedidas para tratar de prevenir este ataque coordinado. La primera contramedida tratará de neutralizar la denegación de servicio contra el equipo H_1 tan pronto como el equipo que está realizando tal ataque lo detecte. La segunda contramedida, suponiendo que la primera contramedida no haya sido ejecutada con éxito, y por tanto, que el ataque haya continuado con normalidad, tratará de neutralizar el inicio de la sesión remota con suplantación de IP contra el equipo H_2. Para ello, el equipo eliminará el proceso responsable de la conexión tan pronto como el acceso remoto ilegal se vaya a producir.

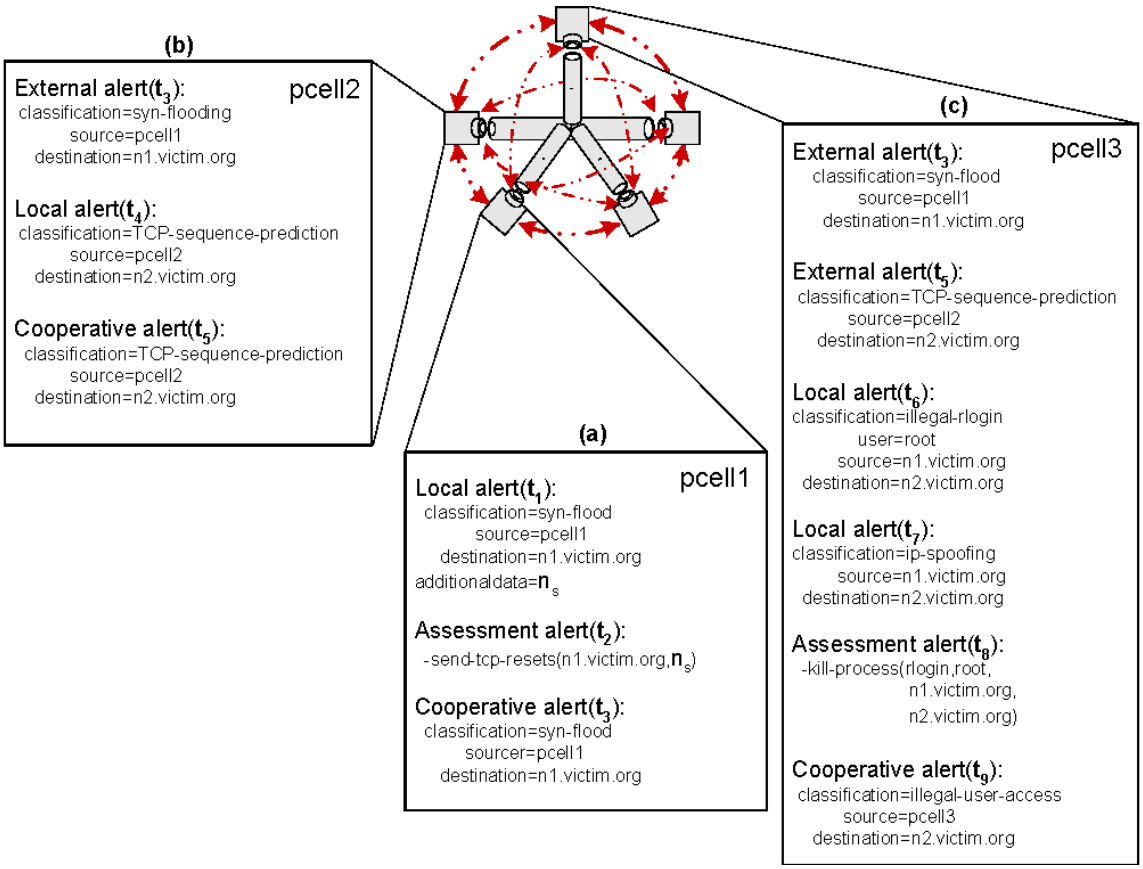

Figura 6. Secuencia de alertas lanzadas en el interior de cada celda de prevención

Para mostrar como los componentes de nuestra arquitectura manejarán la prevención del ataque coordinado descrito en la figura 5, consideraremos el escenario mostrado en la figura 6. Asumiremos que un atacante trata de realizar un acceso remoto ilegal contra uno de los equipos de la red del dominio `victim.org`. Para ello, el atacante utilizará los recursos de otra red corporativa que se encuentra protegida con nuestro sistema de celdas de prevención. Las diferentes partes del ataque serán detectadas por las celdas de

prevención que se encuentran alojadas en los equipos *pcell1*, *pcell2*, y *pcell3* (ver figura 6). Para cada celda de prevención se muestra en la figura 6 tan sólo la información más relevante de la alerta IDMEF publicada y consumida por los componentes de la celda. Esta información ha sido simplificada lo máximo posible por motivos de claridad. También, por motivos de claridad, asumiremos que el grafo de correlación y anti-correlación para el ataque de Mitnick se encuentra instalado únicamente en la base de escenarios de estas tres celdas de prevención. Cada alerta se denotará con el identificador t_i, que corresponde al campo *DetectionTime* de la alerta IDMEF.

El primer indicio del ataque es detectado por los sensores de la celda de prevención alojada en *pcell1*. Estos sensores detectan el ataque de denegación basado en SYN flooding y generan la alerta local t_1. Esta alerta será recibida por el motor de correlación de la celda, que a su vez generará la alerta de contramedida t_2 informando que el ataque de DoS debe ser neutralizado inmediatamente. Esta alerta de contramedida será observada por el gestor de contramedidas de la propia celda, que lanzará la acción correspondiente contra la unidad de respuesta que tratará de bloquear la denegación de servicio. Al mismo tiempo, a través del gestor de cooperación, la celda de prevención lanzará la alerta de cooperación t_3 hacia el resto de celdas del sistema. Esta alerta será recibida por los gestores de cooperación de las celdas de prevención del sistema y será publicada en forma de alerta externa en el interior de las mismas.

En este momento, la celda de prevención albergada en el equipo *pcell1* habrá tratado de prevenir el ataque de denegación de servicio contra el equipo `n1.victim.org`. Este caso corresponde con la primera acción del escenario coordinado. En caso de que la contramedida consiga neutralizar la denegación de servicio, el ataque será frustrado. Aún así, supondremos para este ejemplo la contramedida no ha conseguido su objetivo, bien por que la acción no ha podido contrarrestar el ataque de la forma adecuada, o bien por que el atacante ha realizado de nuevo una denegación de servicio contra el nodo `n1.victim.org` desde un equipo sin celda de prevención. Por tanto, el sistema continuará a la espera de que los siguiente pasos del ataque se produzcan tal y como muestra el escenario de la figura 5.

El siguiente paso del ataque, la predicción de los números de secuencia TCP contra el equipo `n2.victim.org`, es detectada por los sensores del equipo que alberga la celda de prevención *pcell2*, que se encargarán de publicarlo con la alerta local t_4 . El gestor de correlación de *pcell2* consumirá la alerta y producirá la correspondiente alerta cooperativa t_5. Esta alerta será enviada a las otras celdas de prevención del sistema, para ponerlas a la espera del siguiente paso que se ha de producir para concluir el ataque.

Finalmente, la detección del ataque coordinado será completada cuando el atacante trate de realizar el inicio de sesión remota en `n2.victim.org` con la suplantación de la IP de `n1.victim.org` desde el equipo que alberga la celda de prevención *pcell3*. En el momento que los analizadores de *pcell3* detecten tal situación, publicarán las alertas t_6 y t_7. Estas alertas, junto con las alertas externas t_3 y t_5, serán puestas en correspondencia por el motor de correlación de *pcell3*. El resultado será la detección del escenario mostrado en la figura 5. El motor de correlación producirá en ese momento la alerta de contramedida t_8 para cancelar la ejecución del proceso utilizado para realizar el inicio de sesión remoto con suplantación de IP. Al mismo tiempo, se encargará de publicar la alerta cooperativa t_9 para informar al resto del sistema lo que ha sucedido.

En esta sección hemos visto como el sistema de prevención que se ha presentado en este artículo puede detectar y prevenir un ataque coordinado. El ataque de Mitnick ha servido para mostrar la forma de actuar de nuestro sistema ante la realización de un sencillo

ataque coordinado de tres pasos. Actualmente estamos trabajando en el modelado de otros ataques coordinados más complejos que ayuden a evaluar nuestro sistema.

6. Conclusiones y trabajo futuro

En este artículo hemos presentado el diseño de un sistema *publicador-suscriptor* para la detección y prevención de ataques coordinados desde recursos de red. Este sistema utiliza una comunicación multicast entre distintas entidades para evitar la participación de los recursos asociados en la realización de un ataque coordinado contra redes de terceras partes o incluso contra la propia red. Nuestra propuesta puede ser fácilmente integrada en cualquier red corporativa pudiendo ser utilizada como un marco de trabajo común para la prevención de ataques coordinados en este tipo de entornos.

Hemos visto también en este artículo la implementación de una plataforma para sistemas GNU/Linux que implementa la mayor parte de los componentes propuestos en la arquitectura. Aunque los componentes de detección y reacción de esta plataforma (sensores y unidades de respuesta implementados como módulos de kernel) están disponibles únicamente para la serie 2.4 del kernel Linux, planeamos una actualización de dichos componentes para la serie 2.6 de Linux en un futuro cercano. Hemos visto también en este artículo como nuestro sistema puede detectar y prevenir un ataque de Mitnick, explotando la distribución y coordinación de los componentes de nuestro sistema.

Como trabajo futuro, pensamos estudiar la posibilidad de incorporar nuevas contribuciones en cuanto a correlación de alertas para nuestro sistema. Algunos ejemplos a estudiar son el modelo de datos formal propuesto en M2D2 [21], el uso de prerrequisitos y consecuencias en ataques propuesto en CARDS [22] y la construcción de escenarios de ataque a través de la integración complementaria de los métodos de correlación propuestos en [23]. Pretendemos realizar también un estudio en mayor profundidad del formato de alertas IDMEF para solucionar problemas de cálculos innecesarios en el interior de las distintas celdas de prevención. Finalmente, sería interesante el estudio e incorporación de mecanismos para tolerancia a intrusiones en nuestro sistema. Estos mecanismos ayudarían a hacer que nuestro sistema ofreciera mayor confianza en el momento que los distintos equipos del sistema vayan siendo atacados.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología, a través de sus proyectos TIC-2003-02041 y TIC2001-5108-E, y el Departamento DURSI de la Generalitat de Catalunya a través del proyecto 2001SGR 00219.

Nos gustaría agradecer a todos los miembros del departamento RSM del ENST Bretagne por el soporte proporcionado a Joaquín García durante su estancia en el ENST Bretagne de Rennes, especialmente a Frédéric Cuppens, Nora Cuppens, Fabien Autrel y Yacine Bouzida por sus comentarios en versiones previas de este artículo.

Referencias

1. M. Asaka, A. Taguchi, and S. Goto. The implementation of IDA: An intrusion detection agent system. In *11th Annual FIRST Conference on Computer Security Incident Handling and Response (FIRST'99)*, 1999.
2. J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, Eugene H. Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *ACSAC 1998*, pages 13–24, 1998.
3. D. Bruschi, L. Cavallaro, and E. Rosti. Less harm, less worry or how to improve network security by bounding system offensiveness. In *Proceedings of the 16th Annual Computer Security Application Conference, ACSAC 2000*, pages 188–195, December 2000.
4. F. Cuppens, S. Gombault, and T. Sans. Selecting appropriate counter-measures in an intrusion detection framework. In *Proceedings of 17th IEEE Computer Security Foundations Workshop*, Asilomar, Pacific Grove, CA, June 2004.
5. F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 2002.
6. F. Cuppens and R. Ortalo. LAMBDA: A language to model a database for detection of attacks. In *Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, Toulouse, France, 2000.
7. D. Curry, H. Debar, and B. Feinstein. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. Internet draft, January 2004.
8. J. D. de Queiroz, L. F. R. da Costa Carmo, and L. Pirmez. Micael: An autonomous mobile agent system to protect new generation networked applications. In *2nd Annual Workshop on Recent Advances in Intrusion Detection*, Purdue, IN, USA, September 1999.
9. H. Debar, M. Dacier, and A. Wespi. *Towards a Taxonomy of Intrusion Detection Systems*. Computer Networks, 1999.
10. G. Dhandapani and A. Sundaresan. Netlink sockets overview. Technical report, The University of Kansas, September 1999.
11. S. Markle et al. The security device event exchange. http://www.icsalabs.com/html/communities/ids/sd-ee/, 2004.
12. R. Feiertag, C. Kahn, P. Porras, D. Schnackenberg, S. Staniford-Chen, and B. Tung. A common intrusion specification language, June 1999.
13. D. Garlan, S. Khersonsky, and J. S. Kim. Model checking publish-subscribe systems. In *Proceedings of the 10th International SPIN Workshop*, Portland, Oregon, USA, May, 2003.
14. G. Helmer, J.Wong, M. Slagell, V. Honavar, L. Miller, R. Lutz, and Y.Wang. Software fault tree and colored petri net based specification, design and implementation of agent- based intrusion detection systems., 2002. Submitted to IEEE Transaction of Software Engineering.
15. J. Hochberg, K. Jackson, C. Stallins, J. F. McClary, D. DuBois, and J. Ford. NADIR: An automated system for detecting network intrusion and misuse. In *Computer and Security*, volume 12(3), pages 235–248. May 1993.
16. R. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proceedings of IEEE WETICE 2003*, Austria, June 2003.
17. C. Kruegel. *Network Alertness - Towards an adaptive, collaborating Intrusion Detection System*. PhD thesis, Technical University of Vienna, June 2002.
18. C. Kruegel and T. Toth. Distributed pattern detection for intrusion detection. In *Network and Distributed System Security Symposium Conference Proceedings: 2002*, 1775 Wiehle Ave., Suite 102, Reston, Virginia 20190, U.S.A., 2002. Internet Society.
19. C. Kruegel and T. Toth. Flexible, mobile agent based intrusion detection for dynamic networks. In *European Wireless*, Italy, February 2002.
20. J. McAlerney and A. Migus. IDMEF XML library version 0.7.1. http://www.silicondefense.com/idwg/lib-idmef/, July 2003.
21. B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: a formal data model for intrusion alarm correlation. In *Proceedings of the 5th Recent Advances in Intrusion Detection (RAID2002)*, Zurich, Switzerland, October 2002.
22. P. Ning, S. Jajodia, and X. S. Wang. Abstraction-based intrusion detection in distributed environments. In *ACM Transactions on Information and System Security*, volume 4, pages 407–452. November 2001.
23. P. Ning, D. Xu, C. G. Healey, and R. A. St. Amant. Building attack scenarios through integration of complementary alert correlation methods. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04)*, pages 97–111, February, 2004.
24. P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, pages 353–365, October 1997.
25. S. Northcutt. *Network Intrusion Detection: An analyst's Hand Book*. New Riders Publishing, third edition edition, September 2002.
26. B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of the third annual technical conference of AUUG 1997*, pages 243–255, Brisbane, September 1997.
27. S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. DIDS (distributed intrusion detection system) - motivation, architecture and an early prototype. In *Proceedings 14th National Security Conference*, pages 167–176, October, 1991.
28. S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – a graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.
29. T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. In *Proceedings of the 5th Recent Advances in Intrusion Detection (RAID2002)*, Zurich, Switzerland, October 2002.
30. G. Vigna and R. A. Kemmerer. NetSTAT: A network-based intrusion detection system *Journal of Computer Security*, 7(1):37–71, 1999.
31. H.Welte, J. Kadlecsik, M. Josefsson, P. McHardy, and et. al. The netfilter project: firewalling, nat and packet mangling for linux 2.4. http://www.netfilter.org/, 2004.
32. G. B. White, E. A. Fisch, and U. W. Pooch. Cooperating security managers: A peer-based intrusion detection system. *IEEE Network*, 7:20–23, January / February 1999.