

ARQUITECTURA Y COMUNICACIONES EN UN SISTEMA DE DETECCIÓN DE INTRUSOS

Autores : Óscar Andrés López, Misael Leonardo Prieto Parra

Coautora : Beatriz Acosta

Universidad de Los Andes
Bogotá, Colombia
{os-lopez, m-prieto, bacosta}@uniandes.edu.co

Abstracto. La arquitectura de los sistemas de detección de Intrusos (IDSs) presente en productos comerciales, está diseñada de tal manera que su interoperabilidad, eficiencia y escalabilidad son limitadas, además la información en cuanto a detalles técnicos de su implementación es reducida y en general el uso de los estándares establecidos en la materia es poco utilizado. En este artículo, se describirán dos estándares, el del “Intrusion Detection Working Group” (IDWG) y el del “Common Intrusion Deteccion Framework” (CIDF), como posibles soluciones para la arquitectura y en especial la infraestructura de comunicaciones y control necesarias para sentar las bases de un IDS.

1 Introducción

Se muestra el problema de especificar una arquitectura y un esquema de comunicaciones desde dos puntos de vista diferentes. Para lograrlo se hace una sinopsis de la arquitectura estándar propuesta por el “Common Intrusion Deteccion Framework” (CIDF), y el planteamiento novedoso del “Intrusion Detection Working Group” (IDWG), el cual es necesario mencionar, dado que está enfocando sus esfuerzos hacia un estándar de comunicaciones de datos y acceso a éstos usando “Extensible Markup Language” (XML).

En el apéndice, se muestran varias arquitecturas de IDSs ya existentes, aclarando que esto se hace con propósitos ilustrativos y no pretenden dar una descripción exhaustiva.

2 Introducción a la Detección de Intrusos

Los sistemas y redes cotidianamente están sujetos a ataques electrónicos, éstos son tan frecuentes que es necesario imponer una gran cantidad de requerimientos de seguridad para la protección de los datos y de los equipos. La vulnerabilidad de los

equipos se chequea mediante herramientas del sistema a nivel de red, que permiten la corrección de problemas y errores de configuración potencialmente responsables de ocasionar fallas en la seguridad de un sistema de información.

Para el manejo de detección de intrusos se colecta información desde una variedad de fuentes del sistema, analizando esta información de diferentes maneras. [19] Algunas de éstas, consisten en comparar esta información con grandes bases de datos de firmas de ataques,[3] otras consisten en mirar problemas relacionados con usuarios autorizados que sobrepasan sus actos permitidos [5] y por último mediante el análisis estadístico sobre la información, buscando patrones que indiquen actividad anormal y que no se tuvo en cuenta en los análisis anteriores (Ej.: accesos que ocurren en horarios extraños, o un inusual número de *logins* fallidos, etc.).

Los ataques a un sistema pueden ser externos a éste o provenientes del interior, por ejemplo, usuarios autorizados o intrusos que se hacen pasar como usuarios legítimamente autenticados por el sistema.[2] Las herramientas del sistema y los sistemas de detección de intrusos en conjunto permiten a una organización protegerse a sí misma de pérdidas asociadas con estos problemas de seguridad.

La detección de intrusos, que es usada cada vez con mayor frecuencia, es el complemento lógico a una red con *firewalls*. Los *firewalls* actúan como una barrera entre la red interna y la externa, filtrando el tráfico que llega de acuerdo con unas reglas y una política de seguridad. Esto sería suficiente de no ser por varios aspectos, entre estos están: [16][19]

- No todos los accesos a Internet o la red externa ocurren a través del *firewall*.
- No todas las amenazas son originadas en la zona externa del *firewall*.
- Los *firewalls* son objeto de ataques.

Al usar un IDS se extienden las capacidades en la administración de seguridad, incluyendo auditoría, monitoreo, reconocimiento de ataques y respuestas, además provee capas adicionales de seguridad al sistema a proteger: puede reconocer ataques, y potencialmente responder a ellos, mitigando los daños. Adicionalmente, cuando los dispositivos fallan debido a ataques que cambian la configuración, ataques conocidos, o errores del usuario, los IDSs pueden reconocer el problema iniciando un mecanismo de respuesta, como por ejemplo, notificar a la persona indicada por un medio seguro. [1]

Para clarificar qué hace y qué no hace un IDS, a continuación se enuncian algunas de sus funciones y limitaciones.

Funciones generales de un Sistema de Detección de Intrusos: [19]

- Monitoreo y análisis de la actividad de los usuarios y del sistema.
- Auditoría de configuraciones del sistema y vulnerabilidades (*firewalls*, *routers*, servidores, archivos críticos, etc.)
- Evaluación de integridad de archivos de datos y sistemas críticos.

- Reconocimiento de patrones reflejando ataques conocidos.
- Análisis estadístico para patrones de actividad anormal.
- Manejo de *audit-trail* a nivel del sistema operativo, con reconocimiento de actividad que refleje violaciones a una política de seguridad.

Funciones generales que un Sistema de Detección de Intrusos *no* realiza: [19]

- No puede compensar mecanismos débiles de identificación y autenticación.
- No puede conducir investigación de ataques sin intervención humana.
- No puede intuir el contenido de la política de seguridad organizacional.
- No puede compensar las debilidades presentadas por manejo de protocolo de redes.
- No puede compensar los problemas en la calidad o integridad de la información que los sistemas proveen.
- No puede analizar todo el tráfico sobre una red congestionada o utilizada a su capacidad máxima.

3 Clasificación de las Características de un IDS

A continuación se enumeran algunos de los conceptos y características operacionales básicas de los IDSs, que es necesario conocer para lograr una mejor comprensión del tema. Esta sección está basada en el documento [14]; para analizar estas características se puede recurrir al estudio de IDSs reales, algunos ejemplos de éstos se encuentran en el apéndice A.

Detección por anomalías. En la detección por anomalías no se buscan señales de intrusiones conocidas, sino eventos anormales en el tráfico de una red en cuestión, se asume la actitud de que algo anormal probablemente sea sospechoso. La construcción de un detector de este tipo comienza formándose una opinión de lo que es "normal" para el sujeto observado (bien sea un sistema computacional, usuario en particular, etc.) y después decidir en qué porcentaje de esta actividad se va a nombrar anormal y cómo tomar esta decisión en particular. Este principio de detección marca comportamiento que es poco probable que se origine en procesos normales pero que no necesariamente está relacionado con casos de intrusiones.

Detección por firmas de ataques. En detección de firmas la decisión es tomada a partir de las bases de conocimiento de un modelo de procesos de intrusión. Se puede definir en cualquiera y en todas las instancias que constituyen comportamiento autorizado o no autorizado, y comparar con el comportamiento observado.

Tiempo de detección. Dos grupos principales pueden ser identificados: Aquellos que detectan una intrusión en tiempo real o cercano al tiempo real (*in-line*) y los que procesan datos de auditoría con algún tiempo de demora (*off-line*) es decir, tiempo no real. Algunos sistemas que hagan detección en tiempo real *in-line*, también pueden realizar el *off-line*, sobre los datos históricos de auditoría. A este tipo de sistemas que combinan los dos tiempos de detección se les denomina híbridos.

Granularidad de procesamiento de datos. Esta es una categoría que contrasta sistemas que procesan datos continuamente con aquellos que procesan datos por lotes (*batches*) en un intervalo de tiempo regular. La otra opción es procesar datos continuamente con un retraso considerable, o procesar datos en pequeños lotes en tiempo real. Nótese la relación que existe entre la granularidad y el tiempo de detección. (tiempo real / tiempo no real, continuo / lotes).

Fuente de datos de auditoría. Las dos mayores fuentes de auditoría de datos en los sistemas supervisados son datos de red (típicamente datos leídos directamente de una red *multicast*) y datos basados en el *host*, teniendo en cuenta *logs* de seguridad; en este campo se incluyen *logs* de sistemas operativos, *logs* de aplicación, *logs* de equipos de red, etc.

Respuesta. Pasiva: Los sistemas de respuesta pasiva responden por notificación a la autoridad que corresponde, y no trata de mitigar por sí mismos el daño realizado. Activa: Los que ejercen control sobre el sistema atacado. En respuesta activa existen dos clasificaciones, estas son: las que ejercen control sobre el sistema atacado y las que ejercen control sobre el sistema atacante.

Procesamiento de datos. Esta clasificación indica el lugar en el cual se procesan los datos. Si los datos de auditoría se analizan en un lugar central se habla de procesamiento centralizado o al contrario, si los datos son recolectados y procesados en muchas fuentes diferentes, se habla de procesamiento distribuido.

Recolección de datos. Se clasifica en distribuido y centralizado dependiendo de la localización de las fuentes.

Seguridad. La habilidad para sobrevivir a ataques hostiles contra el IDS. Se clasifica básicamente en la escala de Alta / Baja.

Grado de interoperabilidad. El grado en el cual el sistema puede operar en conjunto con otro sistema de detección de intrusos, difiere del concepto "número de diferentes plataformas en que corre el IDS". Se maneja a nivel de comunicación entre los componentes de un IDS con los componentes pertenecientes a otro IDS.

4 Estándares en Comunicación y Arquitectura de IDSs

4.1 CIDE

4.1.1 Introducción a CIDE

El “Common Intrusion Detection Framework” (CIDE) fue desarrollado por el CIDE Working Group, el cual comenzó labores en Enero de 1997, conducido por Teresa Lunt de la “Advanced Research Projects Agency” (ARPA), con el fin de cumplir el objetivo de convertir el CIDE en un conjunto de especificaciones, las cuales permitieran tener :

- Sistemas de Detección de Intrusos interoperables y con el máximo de información compartida.
- Componentes de detección de intrusión fácilmente reutilizables en contextos diferentes a los que fueron diseñados.

En el grupo de trabajo se incluyó personal de ARPA, representantes gubernamentales, comerciales y de organizaciones académicas. [7] En su especificación, el CIDE comprende las siguientes partes:

- Un conjunto de convenciones en cuanto arquitectura para modelar las diferentes partes de un IDS.
- Una especificación de mensajería llamada “Generalized Intrusion Detection Objects” (GIDOs) que permite descripción de eventos, dirección de acciones, consulta de eventos y descripción de componentes.
- Unos mecanismos para transmitir GIDOs entre componentes funcionales del CIDE.
- Protocolos para interoperación de los componentes CIDE.
- Una *Application Programming Interface* (API) para reutilización de componentes del CIDE. [6] [7]

4.1.2 Arquitectura

Esta sección introduce la arquitectura asumida por el CIDE, la cual se debe tener como marco general. Es una propuesta de soporte básico para organizar un sistema de detección de intrusos, el cual no necesariamente debe estar organizado de esta manera.

Descomposición Funcional CIDE

CIDE define interfaces entre sus diferentes tipos de componentes. Las cuatro clases de componentes reconocidos son:

- Generadores de Eventos
- Analizadores
- Bases de Datos
- Unidades de Respuesta

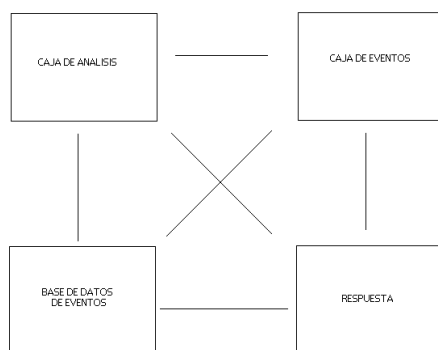


Fig. 1. Arquitectura CIDF. Basada en [8]

La figura 1 relaciona los componentes de un IDS en forma general, cabe anotar que la decisión de estas relaciones depende del ambiente y de la implementación en particular que se desee lograr.

Estos componentes son piezas de un IDS, los cuales se comunican a través de GIDOs y son representados mediante un formato estándar. [7] Los componentes son entidades lógicas y pueden representar algo que produce o consume GIDOs. Un componente debería ser implementado como un proceso sencillo sobre un computador (*thread*), o debería ser una colección de muchos procesos sobre un número de computadores. [8] Es conveniente pensar en GIDOs como objetos, pero esto no predispone una implementación en un lenguaje orientado a objetos. Los componentes que deseen recibir cierta clase de GIDOs describen qué quieren, por su parte los componentes que producen GIDOs describen qué es lo que ellos producen.[7]

4.1.2.1 Generadores de Eventos

El papel de un generador de eventos es obtener eventos en un ambiente computacional externo al IDS y proporcionarlos en forma de GIDOs al resto del sistema. [7] Tales eventos podrían originarse a partir de archivos de *audit-trail*, transmisión binaria de datos, desde otras redes o provenir de otro IDS. [6]

Esto implica que los componentes son reutilizable dado que utilizan GIDOs, pero convertir lo filtrado a GIDOs debe ser una tarea que los desarrolladores de un IDS deben emprender. Es útil generar una especificación acerca de cómo se deben

generar, configurar y utilizar. Los generadores de eventos proveen eventos tan pronto como éstos ocurren, tratando de tener la menor demora en transporte. El almacenamiento de eventos es llevado a cabo en la base de datos para eventos. [8]

4.1.2.2 Analizadores de Eventos

Son los componentes que obtienen GIDOs desde otros componentes, los analizan y retornan nuevos eventos (los cuales se espera estén representando alguna especie de filtro o resumen de la entrada). [7] Es aquí donde se encuentra el corazón de los algoritmos de detección. Una buena parte de la investigación actual al construir un IDS es emprendida en este componente. [6]

4.1.2.3 Bases de Datos - Eventos

Estos componentes representan la funcionalidad de almacenamiento en un IDS, [6] existen para darle persistencia a los GIDOs que sean necesarios. Las interfaces permiten que otros componentes accedan a la base de datos y al consultar la base de datos para GIDOs, éstos sean devueltos. No se asume que este componente sea necesariamente una base de datos, puede ser tan simple como un archivo. [7]

4.1.2.4 Unidades de Respuesta

Estas unidades consumen GIDOs que los dirigen para tomar alguna acción en favor de otro componente CIDF, y llevan a cabo dicha acción. Esto incluye cosas tales como terminar procesos, reinicializar conexiones, alterar permisos de archivos, etc.[8]

4.1.2.5 Componente de Identidad

Es necesario para los componentes tener identificadores, los cuales son usados para distinguir un componente de los otros.

4.1.3 Capas y Servicios

Los componentes CIDF se comunican mediante una arquitectura de tres capas:



Fig. 2. Capas de comunicación en CIDF [7]

4.1.3.1 Capa de GIDOS

Independiente del lenguaje de programación, protocolos de red, etc., CIDF define formatos de datos comunes para la detección de intrusos. Estos datos vienen en paquetes llamados GIDOS. La organización de los datos, su semántica para un componente, y la manera de codificarlos en *bytes* están todos definidos en este nivel. La justificación de esta decisión es que permite separar los datos según como están organizados y qué es lo que significan.[7] La capa de GIDOS existe porque para que los IDSs interoperen de manera significativa, deben tener un entendimiento común de la semántica de los datos. Así mismo, el formato de GIDO es un formato extensible con semánticas definidas para la expresión de eventos de interés de los IDSs. [8] Los GIDOS y su organización se verán más adelante.

4.1.3.2 Capa de Mensajes

Existe porque las múltiples opciones de transporte disponibles presentan deficiencias, de acuerdo con los objetivos que se buscan conseguir. La capa de mensajes garantiza la disponibilidad de enviar mensajes de autenticación encriptados, a través de *firewalls*, etc. La capa de mensajes no acarrea información acerca de la semántica del mensaje, sólo se preocupa de llevar un mensaje desde una fuente a un destino. Correspondientemente, la capa de GIDOS no contiene ninguna información acerca de procedencia, ni destino; esta sólo tiene que ver con el significado de la información portada. [8].

4.1.3.3 Capa de Transporte

La introducción de esta capa está dada en la fase de integración, cada desarrollador negocia y se pone de acuerdo acerca del canal común de transporte. La razón para tener una capa de transporte bajo la de mensajes es que nuestro único requerimiento

es que los componentes entiendan los mensajes. Esto es independiente de la manera en la cual sean transmitidos. Diferentes aplicaciones requerirán diferentes mecanismos de transporte. Todos los componentes tienen como requerimiento soportar unos mecanismos de protocolo por defecto, es UDP confiable. Esto es necesario solamente si los participantes en una comunicación no han acordado previamente un mecanismo de transporte usando medios externos (por Ej., configuraciones locales o mediante el servicio de directorio de CIDF).

A continuación se presenta un esquema de la interoperación entre componentes, la cual es independiente del lenguaje usado para su implementación.

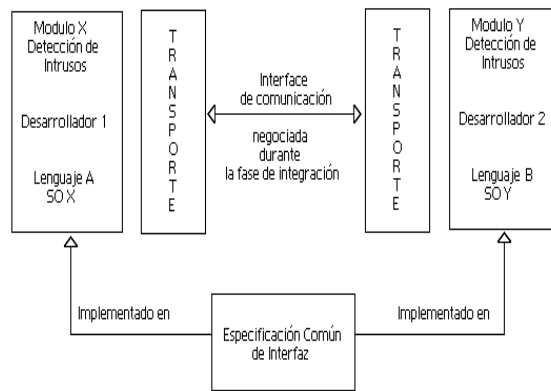


Fig. 3. Interoperación entre componentes [7]

4.1.4 Lenguaje [10]

CIDF describe un lenguaje que puede ser usado para expresar cada una de las acciones detectadas por los componentes del IDS, para esto proporciona una codificación que traduce los mensajes de manera eficiente.

4.1.4.1 Requerimientos

4.1.4.1.1 Impacto de la Arquitectura CIDF

Bajo el modelo CIDF, los componentes manejan entradas y salidas que bien pueden ser el producto de un componente o el resultado. En otras palabras, la salida de un componente puede ser la entrada del otro. En vista que los componentes expresan múltiple información y que ésta debe estar siendo intercambiada, el hecho de que estén interconectados y relacionados hace que sea benéfico encontrar una manera común para expresar los diversos tipos de información. Para esto se puede tener en cuenta que hay actividades en común que se pueden tomar como básicas, sin embargo estos parecidos no pueden oscurecer el hecho que ellos son diferentes en formas

significativas. Sin entrar en detalles específicos de eventos y sus análisis, se nota que los registros de eventos simplemente representan eventos hechos por un componente sobre un sistema. Muchos de estos registros pueden involucrar computación ordinaria o rutinas de mantenimiento. Como resultado, estos registros muy probablemente son producidos en gran volumen.

Por otro lado, los resultados de análisis y las recomendaciones de respuestas están fuertemente correlacionadas con ataques sospechosos. Ellos pueden afectar el razonamiento de un IDS acerca de un ataque, o su proceso de toma de decisiones sobre cómo reaccionar; probablemente tendrán requerimientos de seguridad más fuertes.

4.1.4.1.2 Lista de Metas del Lenguaje

Cualquier lenguaje usado para intercambiar información acerca de ataques debería tener las siguientes cualidades:

Expresivo. Los componentes deberían ser capaces de expresar un amplio rango de intrusiones y malos usos, fenómenos relacionados y prescripciones.

Único en Expresión. No debería ser capaz de expresar las mismas sentencias en un número de maneras distintas.

Preciso. El significado de una expresión en el lenguaje debería estar bien definido.

Por Capas. El sentido específico debería ser expresado como casos de sentido general, así que diferentes receptores con diferentes requerimientos puedan discernir tanto como ellos necesiten de un mensaje.

Autodefinido. Los consumidores que reciben un reporte deberían ser capaces de interpretar mensajes al nivel que ellos necesitan, sin recurrir a negociación externa.

Eficiente. Los mensajes deberían consumir tan pocos recursos del sistema como sea posible.

Extensible. Si un grupo de productores y consumidores deciden sobre información adicional, deberían ser capaces de expresarla.

Simple. Los productores deberían ser capaces de codificar información de una forma rápida, los consumidores deberían ser capaces de extraer la información que ellos necesitan sin necesitar de procesamiento excesivo.

Portable. El lenguaje debería soportar una variedad de plataformas y mecanismos de transporte y soportar significados que no deberían estar limitados a la información que es intercambiada.

Fácil de implementar. Si el lenguaje es muy difícil de implementar no se debe usar.

4.1.4.2 Convirtiendo las Metas en Requerimientos

Para hacerlas útiles, las metas se deben convertir en requerimientos: características comprobables que el lenguaje debe poseer.

Expresivo. Para satisfacer esto, el lenguaje debería tener un amplio vocabulario y sintaxis sofisticada suficiente para cubrir un amplio rango de expresiones.

Único en Expresión. Probablemente no es posible tener un lenguaje expresivo que admita exactamente una formulación para cada sentencia. Si un emisor y un receptor pueden ponerse de acuerdo sobre los objetos de interés, pero no sobre la forma que ellos expresan información acerca de estos objetos, aún así podrían estar habilitados para entenderse el uno al otro. Si no pueden, entonces el lenguaje es demasiado arbitrario.

Preciso. Dos receptores del mismo mensaje no deben tener conclusiones contradictorias.

Por Capas Debería tenerse un mecanismo en el lenguaje por el cual conceptos específicos son definidos en términos de lo más general. Es decir que lo que el lenguaje expresa lo hace mediante la forma más general.

Autodefinido. Ello debería ser auto evidente en un mensaje, cómo cada dato debería ser interpretado.

Eficiente. El costo marginal de usar este lenguaje debería ser no mayor que un factor de dos.

Extensible. Aquí debería contarse con un mecanismo por el cual un productor puede usar su propio vocabulario, e indicar este hecho a los receptores, de tal manera que los receptores puedan recuperar el significado del nuevo vocabulario, o decidir cómo interpretar el resto de un mensaje en el cual ocurre.

Simple. Los llamados componentes "tontos", tal como los *routers*, deberían ser capaces de enviar y recibir mensajes simples sin entender el lenguaje como un todo, simplemente llenando los espacios vacíos y / o extrayendo de ellos la información requerida.

Portable. La codificación para el lenguaje no debería depender de el orden (*endian-ness*) del *host* sobre el cual el mensaje este codificado, o sobre los detalles de su red.

Fácil de implementar. El requerimiento final es que se desarrolle. Es el requerimiento más difícil de establecer, su establecimiento se asegura en la etapa de pruebas.

4.1.4.3 La Aproximación Propuesta

Se propone el uso de S-expresiones, estos son grupos recursivos de marcadores (*tags*) y datos; típicamente, la agrupación es hecha con paréntesis. La ventaja de usar S-expresiones es que ellos proveen una asociación explícita entre términos, sin limitar qué expresan esos términos y sus grupos.

Para llevar a cabo la auto-definición, se añade a la S-expresión un *tag* inicial, que proporciona alguna clave semántica a la interpretación del resto de la S-expresión. Por esta razón, estos *tags* serán llamados identificadores semánticos (SIDs). [10]

4.1.4.4 S-Expresiones y CISL

El lenguaje propuesto para expresar los diferentes eventos al interior de un IDS es el “Common Intrusion Specification Language” (CISL), que usa S-Expresiones.

Se prosigue ahora a describir cómo los SIDs y las S-expresiones son usadas para expresar información acerca de intrusiones y anomalías. Se inicia planteando un ejemplo sencillo [13]:

```
(And
  (OpenApplicationSession
    (When
      (Time 14:57:36 24 Feb 1998)
    )
    (Initiator
      (HostName 'isis.edu.co')
    )
    (Account
      (UserName 'jonny')
      (RealName 'Jonny Dumb')
      (HostName 'tetris.com')
      (ReferAs 0x12345678)
    )
    (Receiver
      (StandardTCPPort 223)
    )
  )
  (Delete
    (World Unix)
    (When
      (Time 14:58:12 24 Feb 1998)
    )
    (Initiator
      (ReferTo 0x12345678)
    )
    (FileSource
      (HostName 'tetris.com')
      (FullFileName '/etc/passwd')
    )
  )
)
```

En este ejemplo, tomado de la documentación de CIDF y adaptado para este marco teórico se puede notar que no es un mensaje que se pueda entender de manera directa, pero tampoco es un mensaje tan complicado que la persona que no conozca las comunicaciones en CIDF no pueda entender parte de su significado.

En este mensaje, a grandes rasgos se dice que se ha iniciado una sesión por parte del usuario con alias Jonny (Nombre real Jonny Dumb) en el *host* tetris.com desde el *host* isis. Y luego de 1 minuto se esta viendo en el tráfico de la red una cadena sospechosa como lo es `‘/etc/passwd’` y se indica que se esta borrando un archivo. La idea con este ejemplo es tener en claro que mediante un SID se pueden tener datos simples como argumentos y que en otros SIDs se puede tener una s-exp como argumento.

4.1.4.5 Tipos de S-Expresiones [10]

- Verbos SID, tales como Delete y OpenApplicationSession
- SIDs de Rol, tales como Initiator y FileDestination
- Adverbios SID, como Outcome y When
- Atributos SID, como Owner
- Átomos SID, como UserName y Time
- SIDs referentes, como ReferTo y ReferAs
- SIDs conjunción, como And

4.1.4.6 Codificación Extendida – Sintaxis de S-Expresiones

Antes de codificar una sentencia, el ordenamiento de SIDs con la sentencia debe ser canonizado. Bajo algún SID, las S-expresiones deberían estar ordenados en orden lexicográfico por el código del SID. Ejemplo:

```
(Delete
  (Outcome ...)
  (Initiator ...)
  (FileSource ...)
)
```

Los códigos SID para Delete, Outcome, Initiator, y FileSource son 0x08000013, 0x08005001, 0x08001001, y 0x08001021, respectivamente. Así mismo, la sentencia debería ser reordenada de la siguiente manera:

```
(Delete
  (Initiator ...)
  (FileSource ...)
  (Outcome ...)
)
```

Después de que el ordenamiento ha sido canonizado, la expresión CISL debe ser decodificada.

La siguiente es una gramática estilo BNF (Backus-Naur Form) para S-expresiones en CISL, se debe tener en cuenta que la gramática no produce sentencias válidas, sin embargo se debe dar el caso que las sentencias válidas tengan sólo un derivación de acuerdo a las siguientes reglas :

```
<SExpression> ::= '(' <SID> <Data> ')'  
    E[SExpression] = length_encode(sid_encode(SID) E[Data])  
                    sid_encode(SID) E[Data]  
  
<Data> ::= <SimpleAtom>  
    E[Data] = simple_encode(SimpleAtom)  
  
<Data> ::= <ArrayAtom>  
    E[Data] = array_encode(ArrayAtom)  
  
<Data> ::= <SExpressionList>  
    E[Data] = E[SExpressionList]  
  
<SExpressionList> ::= <SExpression>  
    E[SExpressionList] = E[SExpression]  
  
<SExpressionList> ::= <SExpression> <SExpressionList>  
    E[SExpressionList] = E[SExpression] E[SExpressionList]
```

Para ver más detalles acerca de los tipos de SIDs, referirse al documento de comunicación de CIDF. [10]

4.1.4.7 Paradigmas CISL

La forma como se codifican varias clases de eventos y análisis de resultados, son conocidos en CISL como paradigmas, los cuales son estructuras básicas a las cuales pertenecen una amplia clase de GIDOs. El uso de estos paradigmas no es parte de la especificación de CISL, pero se espera que la comprensión entre los componentes participantes se incremente si se usan. Ejemplo de un paradigma: Paradigmas de ataque (*outcome, attack class, mechanism*)

4.1.4.8 Funciones de Codificación

4.1.4.8.1 Códigos SID y Longitud de Código

En general, los códigos SID miden cuatro octetos. A cada octeto le corresponde darle un sentido distinto al mensaje, a el primer octeto se le conoce como *bitfield*, el cual contiene información acerca del SID y su tipo de dato. La siguiente es una descripción tomada de la documentación del CIDF [10] , que explica el contenido del primer octeto:

Bit	Interpretation
---	-----
0 (MSB) -- 2	[Reserved, must be 0 for this revision]
3	0 = defined in this specification 1 = vendor-defined SID [if == 1, SID code is followed immediately by 4-octet SID developer ID]
4 -- 5	[Interpreted as an integer from 0 to 2, inclusive:] 0 = simple data type (i.e., not an array) 1 = arrayed data type 2 = S-expression data type (e.g., verb SID)
6 -- 7 (LSB)	[Interpreted as an integer from 0 to 3, inclusive:] 0 = element data type is 1 octet long 1 = element data type is 2 octets long 2 = element data type is 4 octets long 3 = element data type is 8 octets long

Si un SID no es predefinido por CIDEF, entonces el *code* tomado como un SID debe ser de 32 bits y único. Estas ID's deben ser distribuidas por una autoridad apropiada. (Esto se realiza para que no se den repeticiones y para que los SIDs sean significativos y en especial, únicos).

La codificación de un *datum* es contado en octetos. Esta longitud es codificada en una secuencia de cuatro *bytes*. En la notación CIDEF se maneja codificación hexadecimal.

4.1.4.9 GIDO *Addendum*

Esta sección describe el mecanismo de GIDO *addendum*, el cual es usado por un componente CIDEF para adicionar información a un GIDO existente. “Los propósitos son (1) proveer una traducción para la información en el GIDO, (2) proveer alguna información adicional relacionada con el evento descrito en el GIDO, o (3) describir qué fue realizado en respuesta al GIDO.” [10]

El objetivo del GIDO *addendum* es proporcionar estas capacidades sin alterar el GIDO original. Esto habilita a los componentes a preservar la firma atada al GIDO original, lo cual permite la autenticación de origen. Ello también permite a los componentes ver cómo el GIDO ha sido modificado luego de pasar por otros dispositivos CIDEF.

Los GIDO *addendums* son usados para modificar un GIDO previo. Cuando se aplica el GIDO *addendum* a un GIDO, se crea un nuevo GIDO reflejando los cambios especificados en el *addendum*. Si hay múltiples *addendums*, entonces el primer *addendum* es aplicado al GIDO para crear un GIDO modificado. El segundo es entonces aplicado al GIDO resultante para crear un nuevo GIDO modificado, y así sucesivamente.

4.1.4.10 Encabezado de un GIDO

En esta sección se describe a grandes rasgos del encabezado de un GIDO. Cada GIDO formado con las especificaciones del CIDF tiene la siguiente forma:

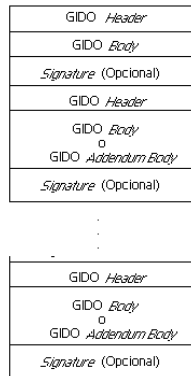


Fig. 4. Estructura de un GIDO con sus correspondientes GIDO *addendums*

4.1.4.11 Los Campos del Encabezado

Esta es una breve descripción de cada uno de los campos que se ubican en el encabezado de un GIDO:

1. *Version ID* (2 octetos)
2. *Class ID* (2 octetos).
3. *Length* (4 octetos, big-endian)
4. *Timestamp* (4 octetos, big-endian)
5. *Thread ID* (4 octetos)
6. *Originator ID* (16 octetos).
7. *Flags* (1 octet).

El contenido del GIDO, plano o comprimido, sigue inmediatamente al encabezado. Si el *bit* de *signature* en uno de los *flags* está en 1, significa que tiene una firma digital. Esta estructura tiene los siguientes campos:

1. Algoritmo de la firma (2 octetos).
2. Longitud de la firma (2 octetos).
3. Longitud específica del algoritmo de firma (2 octetos).
4. Parámetros específicos del algoritmo.
5. Datos de la firma.

4.1.5 API [12]

En este apartado se enuncian algunas de las operaciones disponibles sobre los GIDOs, cómo se codifican, decodifican y transportan, según especificaciones del CIDEF. Los GIDOs pueden ser pensados en dos maneras diferentes: una forma lógica, como una S-expresión, la cual usualmente contiene sub-S-expresiones, y contiene información acerca de una acción importante para el IDS y en una forma codificada, que permite manejar los GIDOs en forma binaria, mediante un API que el CIDEF propone, explicando un mecanismo para construir GIDOs al nivel del programador de la aplicación, codificarlos y posteriormente transmitirlos a otro componente, y decodificarlos a su forma lógica.

Las principales consideraciones que se deben tener en la construcción de un GIDO, se dan en el manejo de estructuras de datos (árboles) y en la implementación necesaria para codificarlos y decodificarlos, para esto se describen todos los llamados o primitivas y su sintaxis propuestas por el CIDEF, entre éstas encontramos:

Constructoras

`cisl_error cisl_init_tree(cisl_tree)` : Este llamado localiza el espacio necesario e inicializa una estructura contenedora de GIDOs.

`cisl_error cisl_attach_sid(cisl_tree, sid)` : Este llamado ubica un SID en la raíz de la estructura contenedora del GIDO.

`cisl_error cisl_attach_data(cisl_tree, type, data)`: Este llamado adiciona los datos a la estructura contenedora.

`cisl_error cisl_attach_child(parent, child)` : Ubica un hijo en la estructura contenedora.

Codificadoras y Decodificadoras

`cisl_error cisl_encode_tree (cisl_tree, cisl_data)` : Codifica un árbol a una secuencia de bytes (`cisl_data`).

`cisl_error cisl_decode_tree (cisl_data, cisl_tree)` : Decodifica una secuencia de bytes y la convierte a un árbol (`cisl_tree`).

Misceláneas

`cisl_error cisl_print_tree (cisl_tree)` : Se encarga de la visualización del árbol por pantalla

`cisl_search_tree(cisl_tree, path, index, return_path)` : Busca en el árbol dado el dato indicado por *index* dado y lo retorna.

`cisl_error cisl_free_tree(cisl_tree)` : Recursivamente convierte el árbol a ceros y libera la memoria que usa.

`cisl_error cisl_free_data(block)` : Pone en ceros la estructura de datos y libera sus apuntadores asociados.

Capa de Mensajes

A nivel de capa de mensajes se manejan variables globales definidas que especifican parámetros de establecimiento de comunicaciones, al igual que la configuración por medio del establecimiento de parámetros para el dispositivo en un archivo de configuración.

Inicialización

`int cidfml_init(filename)` : Este llamado lee en el archivo de configuración del CIDF, en caso de no encontrarlo por defecto se maneja el archivo `"/cidf-ml.ini"`. Las aplicaciones deben llamar `cidfml_init` antes de llamar cualquier otra función de la capa de mensajes CIDF.

`int cidfml_bind(groups[], groupcount)` : Por medio de éste se suscribe a un grupo CIDF o a una lista de grupos (especificado en el archivo de configuración).

`int cidfml_restrict(andel, classid, classcount)` : Restricción de los mensajes que coincidan con una clase de ID especificada.

Comunicación

`int cidfml_recvfrom (handle, data, max_data, timeout, group)` : Recibe un mensaje CIDF. Maneja espera de recibo de mensaje a un nivel individual y de grupos.

`int cidfml_sendto (group[], groupcount, data, length)` : Envía mensaje CIDF comprimido de longitud *length* de datos al grupo (o lista de grupos) especificado.

Terminación

`int cidfml_close(handle)` : Des-inscribe del grupo o lista de grupos CIDF.

`void cidfml_exit(void)` : Libera los recursos CIDF de la capa de mensajes.

Addendum API

`cidfad_error cidfad_apply(body, addenda, result)` : Añade los GIDO *addendum*, explicados en la sección de lenguaje.

También existen métodos del API de firmas digitales.

Cada uno de estos llamados tienen procedimientos generales para el manejo de error, con los cuales se pueden reconstruir exactamente lo que sucedió. En la especificación dada se manejan códigos de error definidos con los cuales se propone manejar la programación .

4.1.6 Comunicaciones

La comunicación dentro del CIDF debe permitir la conexión de manera segura de los componentes del CIDF con otros componentes, lo cual incluye:

- Localizar los componentes apropiados
- Verificar, autenticar y autorizar cada parte de la comunicación.

Los componentes del CIDF, una vez que han hecho contacto y verificado la autenticidad y autoridad del otro, se pueden comunicar de manera segura y eficiente.

El Servicio de Matchmaker

El servicio de *matchmaking* del CIDF, o *Matchmaker*, proporciona un mecanismo estándar y unificado para que los componentes del CIDF se den a conocer con otros componentes y para que localicen "compañeros" de comunicación con los cuales puedan compartir información; el uso de una sola infraestructura para este propósito debería promover en gran medida la interoperación entre componentes y facilitar el desarrollo de un IDS multicomponente.

4.1.6.1 Descripción Funcional del *Matchmaker*

El *matchmaker* debe soportar diferentes niveles de servicio de directorio de acuerdo con lo que solicite un componente cualquiera. Los clientes podrán establecer comunicaciones unos con otros bien sea mediante búsqueda basada en características (opcional) o búsqueda convencional localizando información sobre identidad de los componentes.

Metas Funcionales

- Escalable
- Fácil de administrar.
- Seguro.
- Disponible.
- Debe Sobrevivir a ataques.
- Eficiente.

Objetivos Funcionales

Las siguientes operaciones deberían poder hacerse en un solo paso por parte del administrador:

- Adicionar o eliminar componentes de una categoría.
- Cambiar la categoría jerárquica.
- Actualizar la dirección IP de un componente.

- Los componentes no autorizados en una categoría no deberían poder clamar membresía a ésta.
- Componentes que no estén propiamente autorizados no deberían poder acceder a una categoría o un componente si la búsqueda directa de componentes está habilitada.

4.1.6.2 Arquitectura del Matchmaker

La pieza central de la arquitectura del *matchmaker* es el *matchmaker broker* (intermediario), que proporciona servicios de búsqueda a algunos de los componentes del CIDE, sus "clientes". La arquitectura del *matchmaker* permite que el *broker* se encuentre separado del cliente que sirve, bien sea como un proceso separado en el mismo *host*, o como un proceso en otro *host*. En este caso, un sólo broker podrá servir a más de un cliente.

Los otros componentes del *matchmaker* siempre residen en el lado del cliente. El primero de éstos, es un módulo de autenticación y autorización. El segundo, es un módulo de comunicaciones, que tiene una implementación nula si el broker está localizado en el mismo sitio que el cliente. El tercer componente, es un caché de persistencia para la información recientemente buscada en el directorio.

En las siguientes figuras, las etiquetas en mayúsculas identifican componentes en la arquitectura del *matchmaking*, las etiquetas en minúscula indican interfaces dentro de un componente.

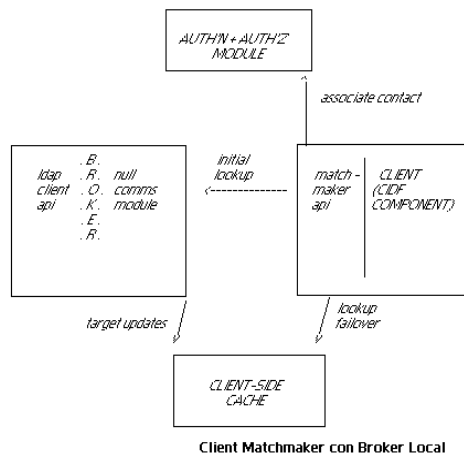


Fig. 5. Cliente del Matchmaker con un broker local [11]

La figura 5 muestra un cliente con un *broker* local; las peticiones de búsqueda del API del *matchmaker* invocan al módulo de comunicaciones, que a su vez llama directamente al *broker*.

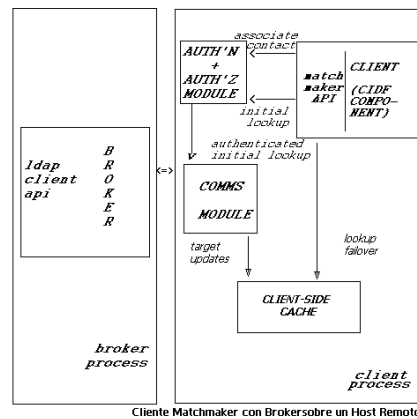


Fig. 6. Cliente del *Matchmaker* con un *broker* en un *host* remoto [11]

La figura 6 muestra lo que sucede cuando el cliente y el broker están en diferentes *hosts*, aquí las peticiones de búsqueda pasan a través del módulo de autenticación / autorización que es quien invoca el módulo de comunicaciones. El módulo de comunicaciones contacta el broker a través de la red.

Para soportar búsqueda basada en características el *matchmaker* agrupa los componentes en categorías y define un orden parcial en las categorías, que permite que se formen relaciones jerárquicas entre ellas.

Para permitir el uso de categorías el *matchmaker* usa un directorio de servicios de propósito general, accedido vía LDAP (Lightweight Directory Access Protocol) [11]

Abstracciones claves

- Es posible tener componentes livianos (*lightweight*) es decir, con poca memoria, escasas capacidades de conexión a una red y sin unidad de disco duro.
- Convenciones de autenticación:
 - La identidad del *host* : El *host* donde está el broker debe tener su propio certificado asociado a una llave pública y firmado por una autoridad certificadora.
 - Identidad de los componentes.

- Identificadores de componentes estables : Un identificador estable es un identificador de instancia de un componente o un identificador de un programa componente.
- Identificador de componentes inestables : Estos no deben ser listados en el directorio.
- Identificador del *broker* : Comprende toda la información necesaria para buscar una instancia de un determinado componente en el directorio.
- Autenticación de componentes: La autenticación debería realizarse en los siguientes puntos:
 - Si el *broker* y el cliente corren en diferentes *hosts*, el cliente debe autenticarse ante el *broker* al iniciar contacto.
 - Al iniciar contacto entre el *broker* y el servidor de directorios debe autenticar al servidor y debe estar preparado para autenticarse ante el servidor si así se le solicita.
 - Al iniciar contacto con un componente remoto, un cliente - componente local debe autenticar el componente remoto y debe estar preparado para autenticarse ante el componente remoto si así se le solicita.

4.1.6.3 Sumario de los Componentes de la Arquitectura del *Matchmaker*

La arquitectura del *matchmaker* fue diseñada para cumplir con los siguientes objetivos:

- Permitir que la costosa responsabilidad de proporcionar funcionalidad de directorio, necesitada principalmente en el momento de inicializar un componente, sea liberada del componente promoviendo modularidad y permitiendo que componentes livianos participen en el *matchmaking*.
- Permitir que la búsqueda de componentes proceda con o sin el soporte de directorio, esto proporcionando APIs separadas para búsqueda de componentes por "características", es decir, utilizando categorías o por medio de direcciones físicas.
- No degradar la operación de los componentes que lo usan, es decir: el *broker* no es un *gateway*, no maneja GIDOs ni se interpone de manera alguna, entre los componentes que ha apareado.
- Proporcionar tolerancia a fallas en el momento de que el directorio no funcione. La información en el lado de los clientes buscada recientemente usando un *broker*, permanece disponible para ser usada por los componentes, aun si las operaciones de la búsqueda no se encuentran disponibles en el momento.

4.1.6.3.1 Módulo de Comunicaciones

La naturaleza y localización del *broker* para un determinado componente estará escondida bajo APIs de configuración neutral. Para los casos en que el *broker* no está en el mismo proceso que el cliente, implica que se debe definir un protocolo por el

cual el código *stub* en el cliente contactará y utilizará al broker, y se recuperará de posibles errores en el proceso.

La labor del módulo de comunicaciones es implementar este protocolo. Cuando el broker y el cliente están en la misma máquina se proporcionará una implementación nula del módulo.

Sin importar la localización del *broker* este debe ser conectado a través de las interfaces importadas para este módulo, sin embargo estas interfaces no son visibles para el cliente si no que son usadas por otro módulo del *matchmaker*.

Si el *broker* se ejecuta en un *host* diferente al del cliente, los dos deben autenticarse mutuamente y deben ser capaces de negociar verificaciones de integridad y confidencialidad en sus comunicaciones.

4.1.6.3.2 Broker de *Matchmaking*

El broker juega dos roles en el *matchmaking*:

- Manejar peticiones provenientes de componentes remotos que se dirigen a su cliente.
- Procesar peticiones que salen desde su cliente, dirigidos hacia otros componentes.

4.1.6.3.3 Operaciones del Cliente en el Broker

Una vez inicializado, un cliente usa su *broker* para buscar otros componentes. Una búsqueda iniciada por el cliente del *broker* se denomina “*lookup* de salida”. Una iniciada por algún componente remoto se denomina “*lookup* de llegada”. Para cualquier forma de “*lookup* de salida”, el broker debe actualizar su caché de llamadas para contener información acerca del cliente actual (este tiene uso potencial para efectuar *matchmaking* local).

El *matchmaking* depende de los roles establecidos en la comunicación, el rol iniciador y el rol aceptante. Cualquier cliente que esté realizando una búsqueda de salida, primero asume el rol iniciador y busca contacto con cualquier asociado que ya esté activo. Adicionalmente, este cliente tiene la opción de asumir el rol de aceptante, pidiendo esperar *lookups* de entrada futuros. Esta acción se llama *lookup* persistente.

El procedimiento normal para establecer una asociación sería entonces que un aspirante a asociado hace un *lookup* persistente, el cual es satisfecho después en un *lookup* posterior (bien sea persistente o no) por un segundo asociado.

Nótese que cuando un directorio está en uso, el *host* del cliente ya debió haber sido añadido a la categoría implicada por la petición de *lookup*. Estas operaciones forman parte de la “certificación de *host*”.

4.1.6.3.4 *Matchmaking* local

Cuando se usa por lo menos un directorio, el broker necesitará llevar cuenta de las clases GIDO en uso por cualquiera de sus clientes, de forma tal que el *broker* se pueda registrar como productor o consumidor de las categorías adecuadas.

4.1.6.3.5 Módulo de autenticación/autorización

Este módulo deberá ser usado por los componentes, también deberá ser usado por los clientes para efectuar autenticación mutua con un *broker* ejecutándose en un *host* remoto.

4.1.6.3.6 Caché en el lado del cliente

Un cliente ubicará información en caché acerca de un cierto número de sus asociaciones recientemente establecidas. Este caché será persistente (en disco). El objetivo de este caché es supervivencia, no desempeño. El caché será usado en los siguientes contextos:

- Si la búsqueda convencional en directorios falla debido a la ausencia de disponibilidad del servidor, se intentará una búsqueda en caché.
- Aún cuando la búsqueda en el directorio tenga éxito, los resultados de la búsqueda en el caché serán usados para aumentar los resultados de la búsqueda en el directorio.

Una descripción del formato de los datos comunicados entre componentes durante el *matchmaking*, protocolo para búsqueda de componentes no basado en características, protocolo para búsqueda de componentes basado en características y protocolo para la capa de mensajes se puede encontrar en el documento [11], la descripción de LDAP se encuentra en mayor detalle en los RFC de LDAP. [17][18]

4.2 IDWG

4.2.1 Introducción

Al igual que en el CIDF, el “Intrusion Detection Working Group” (IDWG) propone su propio esquema de mensajería y comunicaciones, con el propósito de buscar un estándar en el campo de los IDSs.

La propuesta del IDWG se basa en un esquema de mensajería concebido para utilizar el “Extensible Markup Language” (XML). A grandes rasgos, lo que pretende es implementar un modelo de datos orientado a objetos descrito utilizando el

“Universal Modeling Language” (UML), asociando a cada clase un “Document Type Definition” (DTD), usado para describir documentos en XML.

El quehacer del IDWG debe considerarse como un “trabajo en progreso”, es decir, está sujeto a cambios y modificaciones sin previo aviso. Los documentos que describen este esquema son *Internet-Drafts*, lo que significa que se ajustan a las previsiones de la sección 10 del RFC 2026 [36], y tienen una validez máxima de seis meses.

4.2.2 Mensajería

La idea detrás de usar XML para representar un modelo de datos, es poder aplicar diseño Orientado a Objetos (OO) a la construcción de un mensaje dentro de un IDS. XML es un metalenguaje, es decir, un lenguaje para describir lenguajes, en donde cada documento XML tiene asociado un DTD que define cómo debe construirse correctamente. Hoy por hoy, XML es un estándar de construcción de documentos, y un mensaje de un IDS definido en XML tiene la ventaja de poder ser interpretado (*parsed*) utilizando alguna de las numerosas herramientas de libre distribución disponibles para tal propósito. Gracias al paradigma OO dentro del que se diseñaron los mensajes, éstos pueden extenderse y/o ampliarse con facilidad según las necesidades particulares de cada IDS, aplicando conceptos de herencia y agregación.

Los documentos del IDWG proporcionan una extensa descripción de las clases que conforman su modelo de datos, con sus respectivos atributos, la representación de los datos en éstas y las relaciones que deben existir entre ellas. El conjunto de estas clases forman el “Intrusion Detection Message Exchange Format” (IDMEF), la propuesta para un esquema de mensajería del IDWG.

Debe notarse que un documento en XML está escrito básicamente en texto plano. Aunque cuenta con previsiones para incluir información binaria, ésta no se representa de una manera eficiente, y debe evitarse siempre que sea posible. Es decir, dentro de una clase del IDMEF, los tipos de datos int, float, etc., se representan como cadenas de caracteres.

El esquema general del modelo de datos del IDMEF es el siguiente:

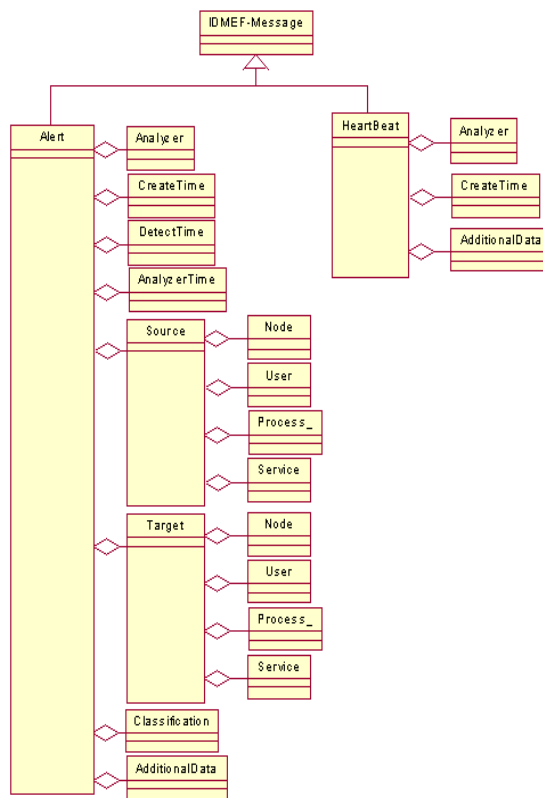


Fig. 7. Modelo de datos del IDMEF [23]

Una descripción detallada de cada clase, con sus atributos y una justificación de sus relaciones con otras clases se puede hallar en [23].

Aunque se hace un gran énfasis en la adecuada sintaxis que debe cumplir un mensaje del IDMEF, no se hacen mayores previsiones acerca de su semántica, incluso podría decirse que la relación entre semántica y sintaxis no está bien definida, puesto que uno o más documentos XML diferentes entre sí podrían tener la misma semántica. En el documento de requerimientos [21] del IDWG, numeral 7.18, se pide que “La semántica de un mensaje IDMEF DEBE estar bien definida”, se da una justificación de este requerimiento, pero no se menciona cómo debe llevarse a cabo.

4.2.3 Comunicaciones

Idealmente, las comunicaciones dentro de un IDS usando el esquema de mensajería propuesto por el IDWG deben llevarse a cabo usando una serie de protocolos

similares a HTTP/1.1, con algunas salvedades que pretenden darle mayor flexibilidad a las comunicaciones. Por ejemplo, en HTTP es el cliente de la comunicación quien inicia la conexión en un esquema de pedido-respuesta, lo que impide correr el protocolo “en reversa”. Este problema, y la inhabilidad de proporcionar a un *proxy* (en caso de que exista uno) información sobre la conexión si el protocolo está siendo ejecutado sobre una sesión “Transport Layer Security” (TLS), hacen necesario definir tres mecanismos de comunicaciones, cada uno para satisfacer una necesidad particular.

El primero de dichos mecanismos es un protocolo, el “Intrusion Alert Protocol” (IAP) [22]. Se trata de un protocolo a nivel de aplicación para intercambiar datos de alertas de intrusiones (entendidas como la sub-clase *alert* de un mensaje IDMEF) entre los elementos de detección de intrusos, en particular, entre sensores/analizadores y *managers*, usando redes IP. Debido a la naturaleza sensible de la información de las alertas, se toman numerosas precauciones (establecer un canal seguro, encriptación usando llaves públicas y privadas, uso de certificados, autenticación de identidad, etc.) para garantizar el transporte y la seguridad del mensaje. IAP usa TCP como mecanismo de transporte subyacente.

A grandes rasgos, este protocolo se encuentra dividido en dos fases. En la primera fase (*setup*), actúa como un protocolo de petición-respuesta, en donde las peticiones son enviadas por la parte (*peer*) que inició la conexión TCP. Esto establece los roles de los *peers*. Las dos partes negocian si la conexión será usada para parejas petición-respuesta donde el analizador es quien envía los mensajes, o viceversa. La segunda fase (*data*) es también una sesión de petición-respuesta, en la cual la aparte que hace las peticiones podría invertirse, basada en las negociaciones de la primera fase. Si la conexión es usada para llevar la información iniciada por el analizador, como las alertas, las peticiones serán del analizador. De lo contrario, serán del manager.

El segundo mecanismo de comunicaciones definido por el IDWG es el “Intrusion Detection Exchange Protocol” (IDXP) [26]. Se trata de un protocolo de nivel de aplicación, con el propósito más general de intercambiar datos entre las entidades de detección de intrusiones. IDXP soporta autenticación mutua, integridad y confidencialidad sobre un protocolo orientado a conexión. Permite el intercambio de mensajes IDMEF, texto sin estructura y datos binarios. IDXP está especificado en parte como un *profile* del “Blocks Extensible Exchange Protocol” (BEEP)[37], el cual es un marco para protocolos de aplicación genéricos para interacciones asíncronas y orientadas a conexión. Es necesario aclarar que BEEP también es un trabajo en progreso, sujeto a modificaciones en cualquier momento, sin embargo, la ventaja de utilizarlo es que características como la autenticación y la confidencialidad son provistas por otros *profiles* de BEEP ya existentes. Para más detalles sobre BEEP y su uso de *profiles*, consultar [37].

El último mecanismo de comunicaciones es un *profile* de BEEP conocido como TUNNEL [25], que le permite a un *peer* de BEEP actuar como un *proxy* de capa de aplicaciones, permitiendo que usuarios autorizados accedan a servicios a través de un *firewall*. El *profile* TUNNEL provee de un mecanismo para que *peers* de BEEP

cooperen para formar un túnel a nivel de aplicación. Los *peers* intercambian elementos “túnel” que especifican una ruta de origen, donde el elemento más externo es removido y usado para decidir el siguiente salto. El elemento más interno, el “túnel” vacío indica el destino final que es, en efecto, el último. Aquí, el término “*proxy*” se usa para referirse a cualquiera de los elementos BEEP distintos del iniciador y del destinatario final.

4.2.4 Arquitectura

El IDWG no define una arquitectura en su propuesta para un estándar dentro de los IDSs. Este grupo se interesa principalmente en definir un esquema de comunicaciones y mensajería, que en principio se pueda adaptar a cualquier tipo de arquitectura, según las necesidades particulares de cada caso. Sin embargo, hace algunas suposiciones arquitecturales, que se enumeran más adelante.

Al igual que en el CIDF, se distinguen varios módulos, no necesariamente separados: el analizador, el sensor, la fuente de datos y el *manager*.

El analizador es el componente de Detección de Intrusos que analiza los datos recolectados por el sensor, buscando señales de actividad no autorizada o indeseada, o de eventos que puedan ser de interés para el administrador de seguridad. Es quien envía mensajes IDMEF.

El sensor es el componente que recolecta datos de la fuente de datos, y que envía eventos al analizador.

La fuente de datos es la información que un sistema de detección de intrusiones usa para detectar actividad indeseada o no autorizada. Incluye paquetes de red, *logs* de auditoría del sistema operativo, *logs* de aplicaciones, etc.

Finalmente, el *manager* es el componente o proceso desde el cual se administran los diversos componentes del IDS. Algunas de sus funciones incluyen: configurar los sensores y los analizadores, administrar la notificación de eventos, consolidar los datos y generar reportes.

A continuación se muestra un esquema de la posible interconexión de los módulos, sus relaciones y el tipo de mensajes que podrían generar. No todos los IDSs tienen todos los módulos, algunos podrían tener más de uno de un determinado tipo, u otros podrían estar integrados en el mismo componente. Se trata simplemente de un diagrama ilustrativo, no se debe tomar como algo establecido.

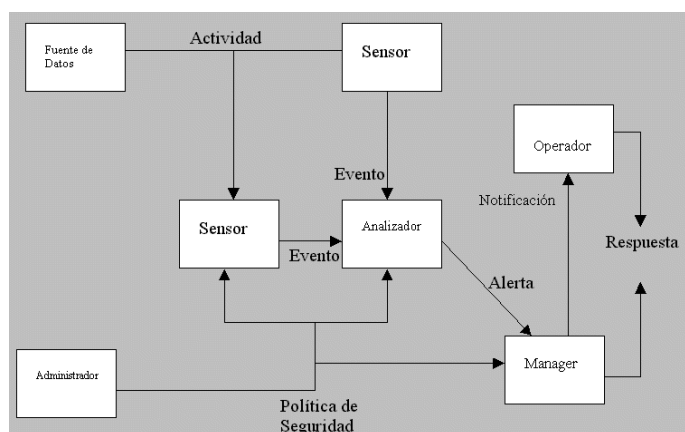


Fig. 8. Ejemplo de Arquitectura IDWG [21]

En la arquitectura IDWG se hacen las siguientes suposiciones: existen como mínimo un módulo analizador y un *manager* o módulo de control, que son componentes separados y que se comunican a través de una red TCP/IP. No se contempla ninguna otra forma de comunicación entre estas entidades. El analizador determina de alguna manera que un evento sospechoso ha sido detectado por un sensor, y envía una alerta al *manager*. El formato de la alerta y el método de comunicarla es lo que se pretende estandarizar, dejando abiertas muchas posibles implementaciones. Por ejemplo, no importa si el sensor y el analizador están integrados o separados, si el analizador y el *manager* están aislados o incorporados dentro de un conjunto de componentes distribuidos, si el *manager* notifica a un humano, toma acciones automáticamente o sólo analiza alertas y las correlaciona, si un componente actúa como un analizador respecto a un componente al tiempo que actúa como *manager* de otro ... etc.

5 Conclusiones

Mediante de este artículo se buscó mostrar la importancia de un IDS en la organización actual que maneje sistemas de información, tratando de tomar una postura imparcial ante el tema.

En vista de la poca información que se consigue en cuanto a estándares en IDS, es importante tener en cuenta las labores desarrolladas por los grupos del CIDEF y del IDWG, también es relevante considerar las consecuencias al estudiar cada una de estas propuestas cuya tendencia es ser lo más generales posibles, pero que definitivamente en algunos aspectos la propuesta del CIDEF aventaja a la presentada por el IDWG y viceversa.

Es por esto que si se necesitara elegir una de estas dos propuestas para implementar un IDS real, cada una de las características de un IDS que se incluyen en este análisis

juega un papel importante a la hora de tomar de una decisión. Adicionalmente se pueden considerar propuestas propias que se adapten mejor a un ambiente en particular, o iniciativas diferentes a las señaladas (AAFID, EMERALD, SNORT, etc.).

Las principales diferencias entre las dos propuestas que compiten por ser el estándar de comunicaciones en IDSs comprenden, por una parte, el uso de un lenguaje bien conocido por parte del grupo IDWG (XML), que según las recientes tendencias en el manejo de datos es casi seguro que se convierta en un estándar de mensajería en IDSs. aunque esta afirmación insinúa que la propuesta del IDWG puede llegar a imponerse sobre la del CIDF, el esquema basado en GIDOs cuenta con una arquitectura bien especificada en cuanto a comunicaciones se refiere, y maneja un lenguaje con una semántica y sintaxis bien definidas que busca ser lo más específico posible a la hora de transmitir información que tenga un significado importante para la detección de intrusos, cosa que no sucede al tratar con XML, dado que éste es un lenguaje que busca expresar contenido en forma general; de igual forma, se cuenta con primitivas para el manejo de componentes en la arquitectura CIDF, que proporcionan la funcionalidad necesaria para el intercambio de mensajes, algo que no se encuentra bien definido en la propuesta de IDWG.

En cuanto a la arquitectura de IDSs se puede notar que las dos propuestas manejan componentes con tareas definidas que buscan cumplir con la detección de intrusos, tanto el CIDF como el IDWG trabajan a un nivel general, el cual permite una implementación que no sea determinada por el modelo escogido.

6 Trabajo Futuro

El estudio de los IDSs propicia un trabajo continuo, debido a la naturaleza cambiante de esta tecnología. Este, y todos los documentos escritos por el grupo de investigación sobre IDSs en la Universidad de los Andes, eventualmente sentarán las bases teóricas para el desarrollo e implementación de un IDS.

El paso lógico a seguir en un futuro cercano es precisamente llevar a la práctica toda la teoría que se ha expuesto en esta investigación, aplicada al desarrollo de un IDS en un ambiente “real”. Esta es la única forma de trascender la teoría: enfrentarse a los retos de implementarla, afrontando las dificultades propias de un caso de uso en particular.

Adicionalmente, sería importante profundizar más en los detalles técnicos de los IDSs ya existentes, en particular, de EMERALD, puesto que emplea uno de los estándares tratados en este artículo: CIDF con manejo de componentes distribuidos.

El tema de la arquitectura y comunicaciones dentro de los IDSs permanece completamente abierto a innovaciones y a nuevas propuestas, hecho que hace necesario estar dispuestos a aceptar con mente abierta nuevas posibilidades. El día de

mañana, la teoría expuesta en este documento podría quedar obsoleta. Quizás el mayor compromiso al investigar en este tema, es el estar siempre actualizados y dispuestos a aceptar el cambio como algo cotidiano.

No se debe dejar aparte el desarrollo de documentos de investigación y la mejora de los mismos con el fin de sentar bases de conocimiento mucho más profundas que sirvan para la implementación de componentes más avanzados y que exijan cada vez una mayor especialización en un componente IDS y que por ende promueva la mejora del mismo.

Referencias

1. Blanco, Marcela. Respuestas avanzadas en un IDS. Universidad de Los Andes, 2001.
2. Holguín, Andrés. Módulo de autenticación LDAP para sistemas de información e IDS. Universidad de Los Andes, 2001.
3. Moreno, Andrés. Procesamiento de Información en un Sistema de Detección de Intrusos. Implementación en un Ambiente TCP/IP. Universidad de Los Andes, 2001
4. Prieto, Edgar Leonardo. Sensores basados en Firewalls. Universidad de Los Andes, 2001
5. Vega, Juan Pablo. Módulo de Análisis off-line. Universidad de Los Andes, 2001
6. Amoroso, Edward G. An Introduction to Internet Surveillance, Correlation, Traps, Trace Back, and Response, Paperback/Published 1999 218 pages 1 edition (February 1999) Intrusion Net Books; ISBN: 0966670078
7. <http://seclab.cs.ucdavis.edu/cidf/members.html>. CIDF V0.5. Consultada en abril 9 del 2001
8. <http://www.gidos.org> - Common Intrusion Detection Framework - Architecture. Consultada en abril 15 del 2001
9. <http://www.cerias.purdue.edu/homes/aafid/docs/tr9805.pdf>. Consultada en mayo 19 del 2001
10. <http://www.gidos.org> - A Common Intrusion Specification Language, CIDF working group document. Consultada en abril 9 del 2001
11. <http://www.gidos.org> - Communication in the Common Intrusion Detection Framework, CIDF working group document. Consultada en abril 11 del 2001
12. <http://www.gidos.org> - Common Intrusion Detection Framework APIs, CIDF working group document. Consultada en abril 15 del 2001
13. <http://theory.lcs.mit.edu/~rivest/sexp.html>. Consultada en abril 15 del 2001
14. Axelsson, Stefan. Intrusion Detection Systems : A Survey and Taxonomy.. Consultada en abril 15 del 2001
15. An Architecture for Intrusion Detection using Autonomous Agents
16. Escamilla, Terry. Intrusion Detection: Network Security Beyond the Firewall, Paperback/Published 1998 416 pages (October 1998) John Wiley & Sons; ISBN: 0471290009
17. <http://www.faqs.org> (RFC 1777). Consultada en febrero 22 del 2001
18. <http://www.faqs.org> (RFC 2253). Consultada en febrero 22 del 2001
19. <http://www.icsa.net/html/communities/ids/White%20paper/Intrusion1.pdf> Consultada en mayo 20 del 2001
20. http://www.messageq.com/security/meinel_2.html. Consultada en abril 15 del 2001
21. <http://www.ietf.org/html.charters/idwg-charter.html> - Intrusion Detection Message Exchange Requirements. Consultada en abril 20 del 2001

22. <http://www.ietf.org/html.charters/idwg-charter.html> - IAP: Intrusion Alert Protocol. Consultada en abril 20 del 2001
23. <http://www.ietf.org/html.charters/idwg-charter.html> - Intrusion Detection Message Exchange Format Extensible Markup Language (XML) Document Type Definition. Consultada en abril 20 del 2001
24. <http://www.ietf.org/html.charters/idwg-charter.html> - Intrusion Detection Message Exchange Format Comparison of SMI and XML Implementations. Consultada en abril 20 del 2001
25. <http://www.ietf.org/html.charters/idwg-charter.html> - The TUNNEL Profile Registration. Consultada en abril 20 del 2001
26. <http://www.ietf.org/html.charters/idwg-charter.html> - The Intrusion Detection Exchange Protocol (IDXP). Consultada en abril 20 del 2001
27. http://www.sans.org/newlook/resources/IDFAQ/ID_FAQ.htm. Consultada en abril 26 del 2001
28. <http://www-rnks.informatik.tu-cottbus.de/~sobirey/ids.html>. Consultada en mayo 21 del 2001
29. <http://www.entropia.de/ecms/Archiv/Security/detection.pdf>. Consultada en mayo 22 del 2001
30. <http://www.snort.org>. Consultada en abril 22 del 2001
31. <http://www.nfr.com>. Consultada en abril 22 del 2001
32. <http://www.sdl.sri.com/projects/emerald/>. Consultada en abril 22 del 2001
33. <http://seclab.cs.ucdavis.edu/papers/DIDS.ncsc91.pdf>. Consultada en abril 22 del 2001
34. <http://www.gnu.org/copyleft/gpl.txt> - GNU General Public License, Richard Stallman, 1989
35. Jeffrey M. Bradshaw. An introduction to software agents. In Jeffrey M. Bradshaw, editor, Software Agents, chapter 1, pages 3-46. AAAIPress/The MIT Press, 1997. May 1993
36. <http://www.faqs.org> (RFC 2026). Consultada en abril 15 del 2001
37. Rose, M.T., "The Blocks Extensible Exchange Protocol Core".

Apéndice: Arquitectura y Comunicaciones en Algunos IDSs

Es posible encontrar en el mercado numerosos IDSs comerciales. La gran mayoría revelan pocos o ningunos detalles sobre su arquitectura y funcionamiento interno: obviamente esta es una información muy sensible. Sin embargo, algunos IDSs no-comerciales, de distribución gratuita o concebidos en ambientes académicos revelan pormenores de su funcionamiento y arquitectura; a continuación se enumeran algunos de los ejemplos de arquitecturas de IDSs reales encontrados, aclarando que estos ejemplos se citan con propósitos informativos, no se pretende en ningún momento dar una descripción exhaustiva de éstos, ni de sus ventajas y fallas. La razón de citar aquí estas arquitecturas, es que son similares a la arquitectura propuesta en este documento, y eventualmente se podrían usar como material de referencia.

Snort

Este es un sistema “liviano” de detección de intrusos en redes. Es multiplataforma y puede ser implantado para monitorear pequeñas redes TCP/IP. Un detalle muy interesante para el estudio de los IDSs, es que está disponible bajo la licencia pública general GNU [34], puede usarse gratuitamente en cualquier ambiente, y la totalidad de su código es de libre distribución.

La arquitectura de Snort comprende tres subsistemas primarios [30]: El decodificador de paquetes, el motor de detección y el subsistema de *logging* y alertas. Utiliza la librería de *sniffing* de paquetes promiscua “libcap” (de distribución gratuita), lo que proporciona capacidades de filtrado y *sniffing* de paquetes fácilmente portables.

El decodificador de paquetes: El motor de decodificación está organizado alrededor de las capas de la pila del protocolo presentes en las definiciones soportadas de los protocolos de Enlace de Datos y TCP/IP. Cada subrutina en el decodificador impone orden sobre los datos del paquete, sobreponiendo estructuras de datos sobre el tráfico de la red. Snort posee capacidades de decodificación para protocolos Ethernet, SLIP y PPP.

El motor de Detección: Snort mantiene sus reglas de detección en una lista encadenada bidimensional de “Encabezados de Cadena” y “Opciones de Cadena”. Éstas son listas de reglas que han sido condensadas en una lista de atributos comunes en los “Encabezados de Cadena”, con las opciones de modificación de detección contenidas en las “Opciones de Cadena”. A continuación se muestra un ejemplo de la estructura lógica de una cadena de reglas:

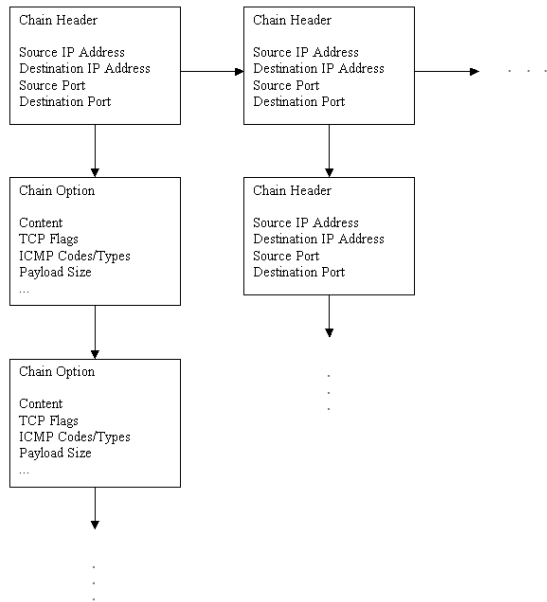


Fig. 9. Esquema de las reglas de Snort [30]

Estas cadenas de reglas son buscadas de manera recursiva para cada paquete, en ambas direcciones. El motor de detección sólo chequea las opciones de cadena que han sido establecidas por el intérprete de reglas en el momento de ejecución. La primera regla que coincida con un paquete decodificado en el motor de detección lanza la acción especificada en la definición de las reglas y retorna.

El subsistema de *logging/alertas* : Este subsistema es seleccionado en tiempo de ejecución mediante comandos de línea. Hay tres opciones de *logging* y 5 de alertas. Las opciones de *logging* pueden ser configuradas para almacenar paquetes decodificados o en formato binario, o ser desactivadas por completo. Las alertas pueden ser enviadas al *syslog*, escritas en un archivo de texto de alertas en dos formatos diferentes (rápido o completo), enviadas como mensajes de WinPopup o ser desactivadas por completo.

Finalmente, se debe mencionar que Snort cuenta con un sistema de *scripting* para definir nuevas reglas y nuevas firmas de posibles ataques, de forma tal que los usuarios pueden actualizarlas fácil y oportunamente.

NFR NID

Network Flight Recorder Network Intrusion Detection [31] es un sistema de detección comercial, que de manera discreta monitorea redes en tiempo real y genera alertas cuando algo sospechoso es detectado. Busca actividades tales como ataques conocidos, comportamiento anormal, intentos de acceso no autorizado y transgresiones a las políticas de seguridad, guardando la información asociada y generando alertas según sea necesario.

La actividad sospechosa puede ser categorizada en uso inapropiado y anomalías. Un uso inapropiado es cuando se presenta un ataque conocido. La detección de este tipo de intrusiones se efectúa comparando el tráfico de la red con patrones de ataques llamados *signatures*. Las anomalías se presentan cuando ocurre una actividad o un evento que se sale de lo ordinario; los sistemas para detectarlas “aprenden” o son entrenados sobre lo que constituye un comportamiento “normal”, desarrollando conjuntos de modelos que son actualizados continuamente, y las actividades después son comparadas con estos modelos.

NFR NID detecta tanto anomalías como usos inapropiados.

Los componentes de un sistema NFR NID son el (los) sensor(es) NFR, la Interfaz de Administración NFR y el Servidor de Administración Central NFR (en ambientes distribuidos).

El Sensor NFR es el software que analiza el tráfico de la red. Comprende:

1. Un motor de análisis (AE) que monitorea los paquetes a medida que pasan por la red que está siendo vigilada
2. Una librería que contiene *signatures* de ataques (identifican a un solo ataque) y filtros (un grupo de *signatures* que en conjunto pueden detectar cientos de ataques)
3. Un *Recorder* para almacenar las alertas y los datos de actividades asociadas.

Los sensores pueden ser implantados como dispositivos independientes en instalaciones pequeñas, o en múltiples instancias a lo largo de una red.

La Interfaz de Administración de NFR (NFR AI) es usada por el administrador para administrar y consultar un sensor y es la estación de trabajo en la que se muestran las alertas. Los sensores pueden ser administrados bien sea a través de una WAN o de manera remota usando una conexión *dial-up*. Se pueden instalar múltiples NFR AIs de ser necesario.

El Servidor de Administración Central (NFR CMS) es implantado en ambientes distribuidos para administrar múltiples sensores remotos. Proporciona información sobre la configuración y envía alertas al NFR AI. Los administradores acceden al NFR CMS desde un NFR AI para configurar y consultar los sensores remotos, en lugar de tener que comunicarse con cada uno de manera individual.

Un sistema NFR NID puede ser escalado desde un sensor NFR y NFR AI independientes hasta tener múltiples sensores NFR con múltiples NFR AIs, administrados por uno o más NFR CMSs.

Las siguientes gráficas ilustran las dos situaciones :

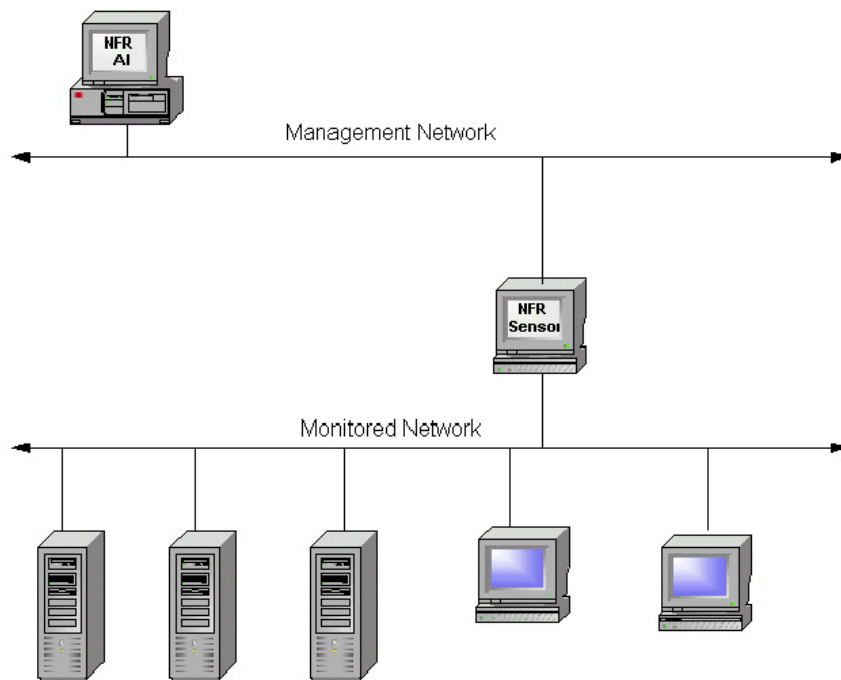


Fig. 10. En esta configuración, el sensor NFR y el NFR AI se comunican en sus propias subredes seguras, de forma tal que no puedan ser detectadas en la red monitoreada. El sensor NFR protege la red monitoreada y se comunica con el NFR AI sólo a través del Administrador de Red seguro. El sensor NFR y el NFR AI podrían ser implantados en la red monitoreada, pero un atacante podría detectar el tráfico del administrador. El tráfico del administrador entre el NFR AI y el sensor NFR es encriptado, permitiendo una administración remota segura, aún sobre redes no confiables o Internet. [31]

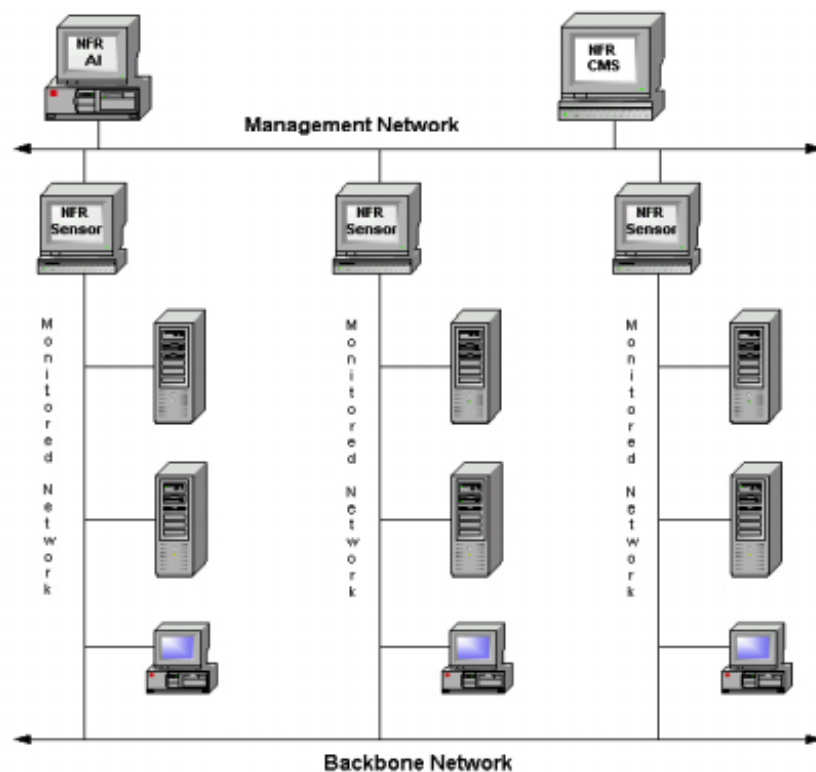


Fig. 11. En este ambiente distribuido, varios sensores NFR han sido implantados, cada uno monitoreando una red diferente. Los sensores NFR se comunican remotamente con el NFR CMS sobre la red de administración –al igual que con el NFR AI, nuevamente para mantener el tráfico de administración separado del tráfico operacional de la red. [31]

EMERALD

EMERALD (Event monitoring enabling responses to anomalous live Disturbances) es un *framework* para efectuar detección de intrusos escalable, distribuida e interoperable a nivel de *host* y a nivel de red [32]. Más que un IDS, es una propuesta de arquitectura para un IDS, que se supone debe contener componentes que permitan al sistema responder activamente ante amenazas, principalmente de atacantes externos a una organización, aunque no se excluye la detección de atacantes internos.

La arquitectura de EMERALD [32][14] está compuesta de una colección interoperable de unidades de análisis y respuesta llamadas “monitores”, que ofrecen una protección localizada de activos claves dentro de una red corporativa. Los monitores de EMERALD son computacionalmente independientes, proporcionando un cierto grado de paralelismo en el cubrimiento de su análisis, al mismo tiempo

ayudando a distribuir la carga computacional y la utilización del espacio. Al implantar monitores localmente, EMERALD ayuda a reducir las posibles demoras en análisis y respuesta que podrían ser consecuencia de la topología espacialmente distribuida de una red. Adicionalmente, introduce un esquema de análisis compuesto, en donde los análisis locales son compartidos y correlacionados en las capas más altas de abstractamente más altas.

La arquitectura de monitores de EMERALD pretende ser pequeña y rápida, y lo suficientemente general para ser implantada en cualquier capa dentro de su esquema jerárquico de análisis. El diseño inicial de la arquitectura de monitores de EMERALD se ilustra en la siguiente figura. En ella, los monitores demuestran un diseño de detección de intrusiones eficiente y descentralizado, que combina análisis de *signatures* con perfiles estadísticos para proporcionar protección localizada y en tiempo real a la infraestructura de servicios de red. El monitor consta de tres unidades computacionales: un motor basado en *signatures*, un motor de perfiles estadísticos y una unidad de respuestas conocida como el *resolver*. Los monitores incorporan un API versátil que mejora su habilidad para interoperar con la máquina objeto del análisis y con otros IDSs.

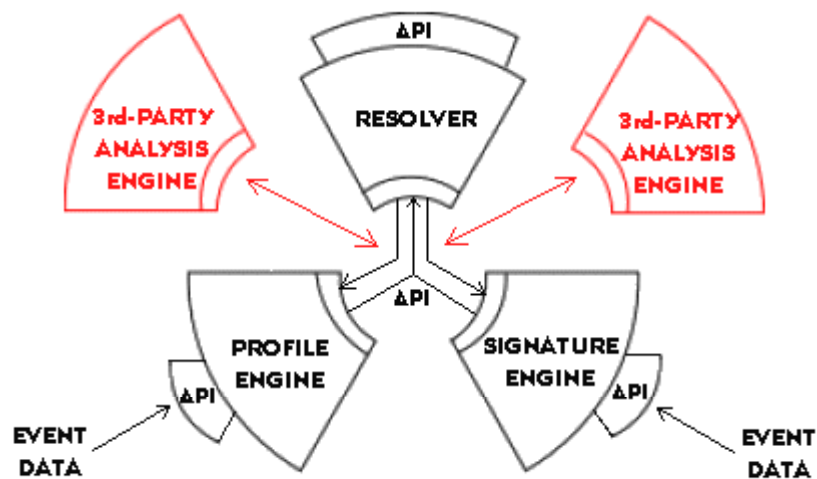


Fig. 12. Arquitectura de EMERALD [32]

Como detalle interesante, la propuesta arquitectónica de EMERALD usa una versión extendida del Common Intrusion Detection Framework (CIDF) como una base para comunicar la información entre los sensores independientes. Futuras extensiones del CIDF permitirán incorporar salidas de herramientas de administración de red y buscadores de vulnerabilidades, y posiblemente también permitirán la comunicación de los resultados de las correlaciones y evaluaciones, permitiendo generar advertencias oportunas e iniciando actividades de respuesta.

DIDS

DIDS (Distributed Intrusion Detection System) [14][33] es un prototipo que actualmente está siendo diseñado e implementado por la división de Computer Science de la Universidad de California, que combina monitoreo distribuido y reducción de datos (por medio de monitores individuales en *hosts* y LANs) con análisis centralizado de datos (por medio del DIDS *director*), para monitorear una red heterogénea de computadores.

La arquitectura DIDS combina monitoreo y reducción de datos distribuidos con análisis centralizado de datos. Esta aproximación es única entre los IDSs actuales. Los componentes de DIDS son el *DIDS director*, un sólo *host monitor* por host y un sólo *LAN monitor* por cada segmento LAN en la red monitoreada. Potencialmente, DIDS puede manejar hosts sin monitores, puesto que el *LAN monitor* puede reportar las actividades de red de dichos hosts. Los monitores de host y de LAN son los responsables primarios de recolectar evidencia sobre actividades sospechosas o no autorizadas, mientras que el *DIDS director* es el principal responsable de su evaluación. Los reportes son enviados de manera independiente y asíncrona desde los monitores host y LAN al *director* a través de la infraestructura de comunicaciones. Los protocolos de comunicaciones a alto nivel entre los componentes están basados en las recomendaciones del ISO “Common Management Information Protocol” (CMIP), permitiendo una futura incorporación de las herramientas administrativas CMIP a medida que sea necesario. La arquitectura también proporciona una comunicación bidireccional entre el *DIDS director* y cualquier monitor en la configuración. Esta comunicación consiste principalmente de eventos notables y reportes de anomalías provenientes de los monitores. El *director* también puede realizar peticiones solicitando información más detallada de los monitores por medio de la directiva “GET”, e impartir comandos para que los monitores distribuidos modifiquen sus capacidades de monitoreo mediante la directiva “SET”. Una gran cantidad de filtración a bajo nivel y una parte del análisis se realiza en el *host monitor* para minimizar el consumo de ancho de banda de la red en la transmisión de la evidencia al *director*. A continuación se muestra un esquema de la arquitectura de DIDS.

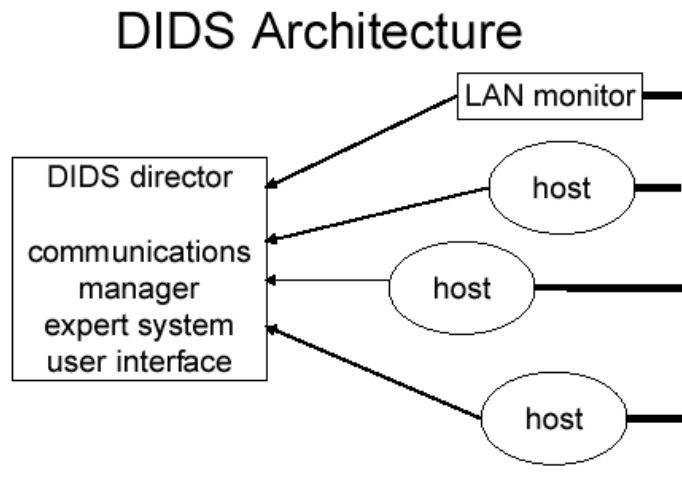


Fig. 13. Arquitectura de DIDS [29]

El *host monitor* comprende un *host event generator* (HEG) y un *host agent*. El HEG recolecta y analiza archivos de auditoría del sistema operativo del *host*. Éstos son examinados en busca de eventos notables, que son transacciones de interés independientes de otros archivos. Estos incluyen, entre otros, eventos fallidos, autenticaciones de usuarios, cambios en el estado de seguridad del sistema, y cualquier tipo de acceso a la red mediante comandos como *rlogin* y *rsh*. Estos eventos notables son enviados al *director* para profundizar en su análisis. El *host agent* maneja todas las comunicaciones entre el *host monitor* y el *DIDS director*.

Al igual que el *host monitor*, el *LAN monitor* consiste en un *LAN event generator* (LEG) y un *LAN agent*. El LEG es un subconjunto del Network Security Monitor de UC Davis. Su principal responsabilidad es observar todo el tráfico en su segmento de la LAN y monitorear conexiones *host-a-host*, los servicios usados y el volumen del tráfico. El *LAN monitor* reporta actividades de red tales como *rlogin* y conexiones de *telnet*, el uso de servicios relacionados con seguridad y cambios en los patrones de tráfico de la red.

El *DIDS director* comprende tres componentes principales, localizados en la misma estación de trabajo dedicada. Puesto que los componentes son procesos lógicamente independientes, también podrían estar distribuidos. El *communications manager* es el responsable de la transferencia de datos entre el *director* y cada uno de los monitores de *host* y de LAN. Acepta la información sobre eventos notables de cada uno de los monitores de *host* y de LAN, y los envía al *sistema experto*. Para beneficio del *sistema experto* y la interfaz de usuario, también es capaz de enviar

peticiones a los monitores de host y de LAN solicitando mayor información sobre un asunto en particular. El *sistema experto* es el responsable de evaluar y reportar sobre el estado de seguridad del sistema monitoreado. Recibe reportes de los dos tipos de monitores, y basándose en éstos reportes, realiza inferencias sobre la seguridad de cada *host* individual y del sistema como un todo. El *sistema experto* es un sistema basado en reglas con capacidades de aprendizaje simple. Finalmente, la *interfaz del usuario* le permite al administrador del sistema de seguridad acceder e interactuar con todo el sistema, permitiéndole observar actividades en cada host, mirar el tráfico de red y solicitar información específica de los monitores.

AAFID

Una última opción entre las arquitecturas enumeradas, es la de usar Agentes autónomos para detección de intrusiones. [9] Un agente de software se define como “Una entidad de software que funciona de manera continua y autónoma en un ambiente en particular, capaz de llevar a cabo actividades de una manera flexible e inteligente de forma tal que pueda responder a los cambios en el ambiente. Idealmente, debe aprender de su experiencia y ser capaz de comunicarse con otros agentes y procesos”. [35]

En el contexto de los IDSs, definimos un agente autónomo como un agente de software que realiza ciertas funciones de monitoreo de seguridad en un *host*. Se denominan autónomos por ser entidades que se ejecutan de manera independiente, aunque podrían necesitar de los datos generados por otros agentes para realizar su trabajo. Adicionalmente, los agentes podrían recibir comandos de control de alto nivel –como indicaciones de inicio y parada de ejecución, o cambiar algunos parámetros operacionales- provenientes de otras entidades.

Se propone una arquitectura llamada de “Autonomous Agents For Intrusion Detection” (AAFID) para construir IDSs que utilicen agentes como su elemento de recolección y análisis de datos de más bajo nivel, y que emplee una estructura jerárquica que permita escalabilidad.

En la figura 14 se muestra un ejemplo de un IDS sencillo, que se adhiere a la arquitectura AAFID.

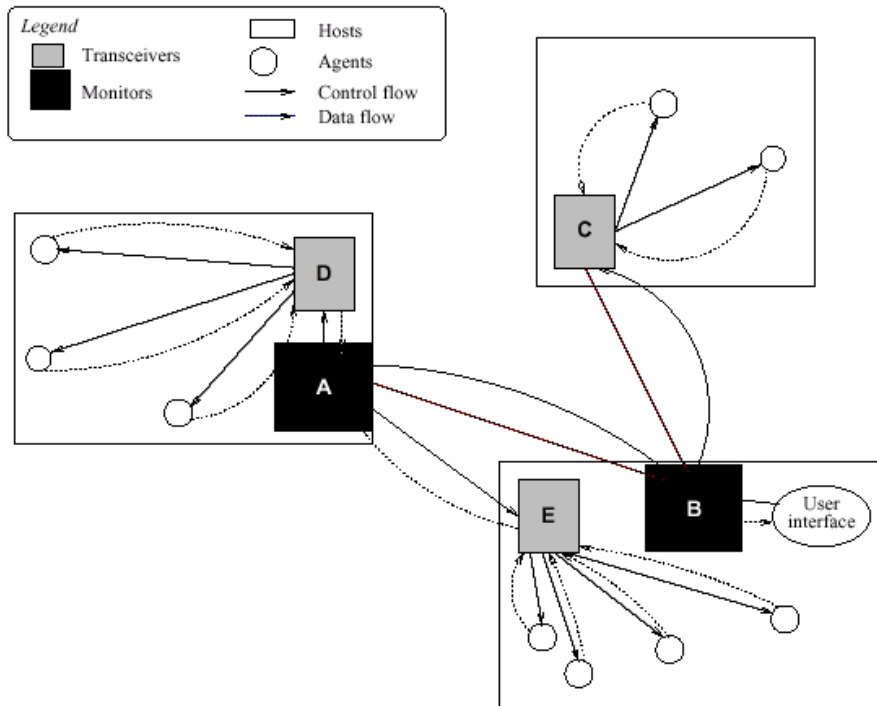


Fig. 14. Disposición física de los componentes en un sistema AAFID, muestra los agentes, los *transceivers* y los monitores, y los canales de comunicación y control entre ellos [9]

Esta figura muestra los tres componentes (entidades) esenciales de la arquitectura: agentes, *transceivers* y monitores.

Un sistema AAFID puede estar distribuido en un gran número de *hosts* en una red. Cada *host* puede contener cualquier número de agentes que monitoreen eventos interesantes que estén ocurriendo en el *host*. Todos los agentes en el *host* reportan sus hallazgos a un solo *transceiver*. Los *transceivers* son entidades, una por *host*, que supervisan la operación de todos los agentes ejecutándose en su *host*. Ejercen control sobre éstos y tienen la habilidad de iniciar, detener y enviar comandos de configuración sobre los agentes. También podrían realizar reducción de datos sobre los datos recibidos de los agentes. Finalmente, los *transceivers* reportan sus resultados a uno o más monitores.

Cada monitor supervisa la operación de varios *transceivers*. Tienen acceso a datos de toda la red, y por tanto pueden realizar correlaciones de alto nivel y detectar intrusiones que involucren varios *hosts*. Los monitores pueden estar organizados de manera jerárquica, de manera tal que un monitor podría reportarse ante un monitor de

más alto nivel. También, un *transceiver* podría reportarse a más de un monitor, para proporcionar redundancia y resistencia ante fallas de alguno de los monitores. En últimas, un monitor es el responsable de proporcionar información y obtener comandos de control provenientes de la interfaz del usuario. Esta organización lógica, correspondiente a la distribución física de la 6 , se observa en la figura 15.

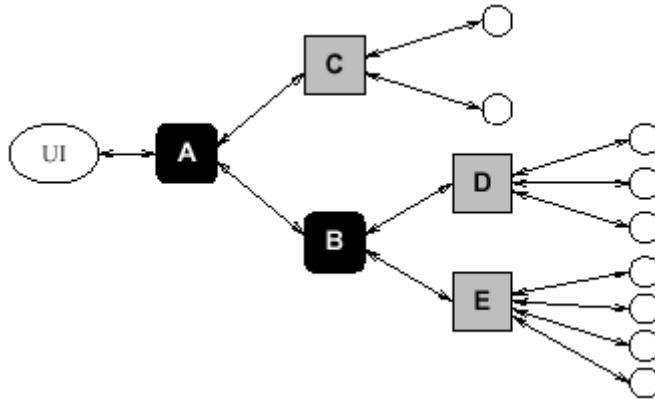


Fig. 15. Organización lógica del mismo sistema AAFID de la figura 6, mostrando la jerarquía de comunicaciones entre los componentes. Las flechas bidireccionales representan el control y el flujo de datos entre las entidades. Nótese que la organización lógica es independiente de la localización física de las entidades en los *hosts* [9]