Defensas frente a ataques DoS a baja tasa contra servidores basadas en políticas de gestión de colas

Rafael Alejandro Rodríguez-Gómez, Gabriel Maciá-Fernández, Pedro García-Teodoro, Jesús Esteban Díaz-Verdejo.
Departamento de Teoría de la Señal Telemática y Comunicaciones, CITIC-UGR,
Universidad Granada,
Calle Periodista Daniel Saucedo Aranda s/n CP 18071 (Granada-España)
rodgom@correo.ugr.es, gmacia@ugr.es, pgteodor@ugr.es, jedv@ugr.es

***Resumen*- En este artículo se evalúa el uso de defensas contra ataques de denegación de servicio a baja tasa contra servidores basadas en políticas de gestión de colas y la viabilidad de su implementación. En un sistema real, para este último fin, se modifica el núcleo del sistema operativo Linux proporcionando un marco de trabajo que permite, de forma flexible y simplificada, la introducción del código que implementa las políticas citadas.**

Se propone una política de gestión de colas y se muestra que su implementación es factible en este núcleo modificado.

Por último, se realizan una serie de pruebas con dicha política de gestión de colas y, a la luz de los resultados obtenidos, se comprueba que es eficaz frente a ataques DoS contra servidores, ya que mitiga sus efectos de forma considerable.

***Palabras Clave*- seguridad, denegación de servicio, gestión de colas, conexión TCP.**

I. INTRODUCCIÓN

Hoy en día se llevan a cabo multitud de operaciones a través de Internet. Muchas de ellas implican tránsito de dinero o de información crucial, y esto conlleva la necesidad de garantizar ciertos niveles de seguridad para que los clientes realicen dichas operaciones. Pero, de la misma forma que existe esta seguridad, existen también una serie ataques que pretenden vulnerarla. La mayoría de éstos tratan de penetrar en un sistema y, de este modo, poder acceder a información privada, e.g., contraseñas de correo electrónico, información de una investigación aún no patentada, números de cuentas bancarias, etc.

Entre los diferentes tipos de ataques existentes, el de *denegación de servicio* (en inglés Denegation of Service o DoS) pretende evitar el acceso a un servicio ofertado de modo total o parcial, mediante el envío de ciertos mensajes hacia uno de los participantes en la comunicación o el propio canal. La mayoría de estos ataques se basan en el envío masivo de peticiones a un participante de la comunicación consiguiendo de esta forma saturarlo, pero se han desarrollado ataques DoS más evolucionados que consiguen su objetivo mediante el envío de peticiones a una tasa de tráfico reducida. El primero en ser desarrollado fue el ataque *Shrew*, diseñado por A. Kuzmanovic [1]. También se pueden citar el ataque de *Reduction of Quality* (*RoQ*) descrito por Guirguis en [2][3], el que M.C. Chan desarrolló para aprovechar las actualizaciones automáticas (descrito en [4]) y G. Maciá-Fernández describió en [5][6] el ataque *DoS a baja tasa contra servidores* (en inglés *Low-Rate DoS Attack against Server, LoRDAS*), que tiene como objetivo reducir la disponibilidad del servidor pudiendo dejar prácticamente sin servicio a todos sus clientes.

En este trabajo se explora un tipo de medidas de defensa frente a este último grupo de ataques (ataques LoRDAS), basada en la aplicación de *políticas de gestión de colas*, entendiéndose éstas como el procedimiento mediante el cual se otorga cierta "inteligencia" a la gestión de las colas del servidor. Se pretende dificultar la tarea parte del atacante de conseguir ventaja sobre los clientes legítimos y así mitigar los efectos del ataque LoRDAS.

En primer lugar, se comprueba si realmente este tipo de medidas de defensa son efectivas contra los efectos dañinos del ataque LoRDAS. Para esto, se diseña una política de gestión de colas sencilla, con el fin de evaluar si ésta es capaz de mitigar los efectos del ataque.

En segundo lugar, se comprueba si las políticas de gestión de colas son una solución factible en cuanto a su implementación en un sistema real. Para este fin, se desarrolla un entorno capaz de incluir en el sistema operativo Linux, de forma flexible y sencilla, cualquier otra política de gestión de colas desarrollada.

Para exponer estas ideas, se divide el artículo en las siguientes secciones. En la Sección II se presentan los fundamentos del ataque DoS contra servidores. En la Sección III se expone la solución propuesta junto con el desarrollo de una política de gestión de colas sencilla. Posteriormente, en la Sección IV, se presenta la implementación de un entorno para inclusión de políticas de gestión de colas en un sistema real. A continuación, en la Sección V, se exponen los resultados derivados de la evaluación de una sencilla política de gestión de colas con objeto de demostrar la eficacia de este tipo de medidas. Por último, en la Sección VI se presentan las conclusiones y unas posibles líneas de trabajo futuro basadas en este trabajo.

II. MODELADO DEL ATAQUE LORDAS

El ataque DoS, en una red de comunicación, tiene como objetivo eliminar o reducir la disponibilidad de un determinado activo mediante la ejecución de determinadas

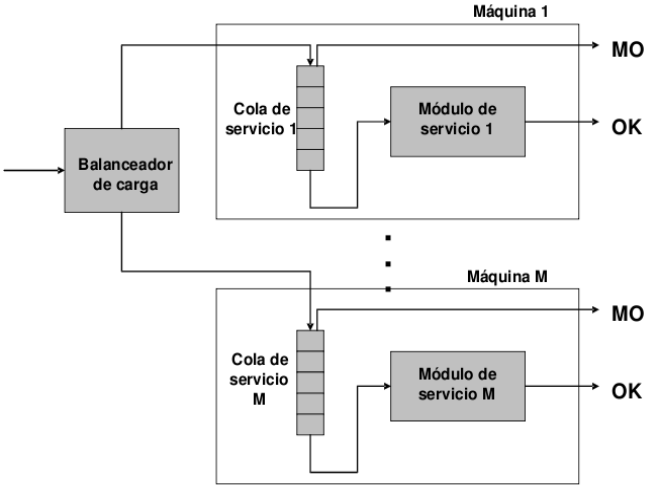

Fig. 1: Modelo de servidor (tomado de [5]).

acciones maliciosas dirigidas a la fuente de información, al canal de comunicación o a ambos. Este tipo de ataques impide o inhibe el uso normal o la gestión de recursos de comunicaciones.

Existen dos métodos fundamentales para la realización de un ataque DoS: la explotación de una *vulnerabilidad* mediante el envío de uno o varios mensajes con una construcción determinada, de manera que sean capaces de aprovecharse de una debilidad existente en el equipo previamente conocida y estudiada y la *inundación* con mensajes de apariencia legítima que acaba por consumir determinados recursos críticos para el funcionamiento correcto del sistema (p.e. tiempo CPU, ancho de banda de red, etc).

Inicialmente los ataques DoS más extendidos eran los de vulnerabilidad. La utilización de estos, con el paso del tiempo, ha ido decreciendo dando paso a un aumento del segundo grupo, los ataques DoS de inundación. Típicamente, para producir la inundación, estos ataques generan una alta tasa de tráfico, lo que los hace detectables por parte de ciertos sistemas de seguridad que basan su detección en un tráfico anómalo de la red. Para evitar la detección por parte de estos sistemas de seguridad, además de para poder atacar sin necesidad de unos recursos abundantes, aparecen los ataques DoS a baja tasa. Este tipo de ataques DoS utiliza la técnica de inundación, pero con la gran diferencia de que no necesitan una alta tasa de tráfico para saturar a la víctima del ataque. Precisamente es esta baja tasa de tráfico la que dificulta la detección de este tipo de ataques.

Para estudiar el ataque contra servidores se define un modelo de servidor consistente en un módulo que recibe mensajes o peticiones, los procesa y emite las respuestas que sean necesarias. Los mensajes recibidos en este servidor pueden proceder bien de los usuarios legítimos, bien de los usuarios malintencionados. El servidor tiene dos tipos de respuestas: los eventos OK y los eventos MO.

- Los eventos OK representan la respuesta que da el servidor a una petición determinada de un cliente en un estado del sistema normal.
- Los eventos MO *mensaje de desbordamiento* (de las siglas en inglés: *Message Overflow*) son la respuesta que da el servidor a los clientes en un estado de sobrecarga de las colas internas o de la memoria de la aplicación.

El modelo completo se ilustra en la Fig. 1 y se ha dividido en las siguientes secciones: un balanceador de carga y M máquinas servidoras, cada una de ellas con una cola de servicio y un módulo de servicio.

Se puede resumir el funcionamiento del modelo siguiendo el camino de una conexión entrante al mismo.

1. La petición llega al balanceador de carga, que elige, según su política interna, la máquina de entre las M posibles a la que enviará dicha petición, exceptuando aquellas máquinas cuya cola se encuentre ocupada completamente. En el caso de que todas las colas se hallen en esta situación el balanceador decidirá aleatoriamente cualquier máquina, ya que el resultado será igualmente la devolución de un evento MO (de desbordamiento).
2. La petición es encolada en la cola de servicio de la máquina elegida por el balanceador de carga siempre y cuando quede alguna posición libre. En caso contrario la petición se descarta y se envía un evento MO.
3. Transcurrido el tiempo necesario en la cola, la petición pasa al módulo de servicio en el que será procesada y tras lo cual se responderá con un evento OK al cliente que inició la petición.

En el ataque LoRDAS, el objetivo básico consiste en evitar la disponibilidad del servidor, haciendo que el tiempo libre de las colas de servicio tienda a cero. Para ello, el atacante debe ser capaz de predecir la aparición de un espacio libre y de insertar una nueva petición suya en cuanto dicha posición aparezca. De este modo, la probabilidad de que los clientes realicen una conexión en ese breve espacio de tiempo es mínima. El atacante estará consiguiendo así que el servidor únicamente sirva sus peticiones y logrará, como consecuencia de esto, denegar el servicio a todos los demás clientes.

En esta situación, el servidor continúa trabajando con total normalidad sirviendo los mensajes que tiene encolados, con la única salvedad de que todos los mensajes que sirve son del mismo cliente, el atacante. Este funcionamiento del servidor, en régimen de normalidad, hace aún más compleja la detección de este tipo de ataques.

El proceso que sigue el atacante para realizar un ataque DoS a baja tasa contra servidores puede resumirse en los siguientes dos pasos:

1. *Saturación* de las colas de servicio. Este paso consiste en llenar todas las colas de servicio de peticiones del atacante.
2. *Captura* del máximo número de posiciones posibles. La captura es el acto de introducir una petición en el momento en el que se libera una posición de alguna cola de servicio, ocupando la posición liberada.

Los procesos de captura y saturación, detallados en [5][6], pueden entenderse como un ataque de vulnerabilidad, aunque el ataque en conjunto es un ataque de inundación al saturar la cola de servicio de la aplicación. El atacante se aprovecha del conocimiento de que en la cola de servicio se libera una posición siempre y cuando el módulo de servicio termine de procesar una petición y envíe el mensaje de respuesta al cliente correspondiente, de modo que, justo tras esta acción, el módulo de servicio que ha respondido,

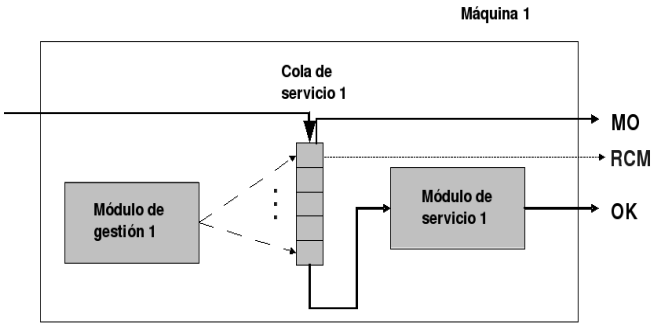

Fig. 2: Modelo de una máquina del servidor con módulo de gestión.

extraerá la siguiente petición de su cola de servicio y generará una posición libre en ella. Esta es la vulnerabilidad que aprovecha el atacante para realizar con éxito el ataque DoS a baja tasa contra servidores.

III. Efecto de las políticas de gestión de colas

La idea presentada en este artículo con objeto de mitigar los efectos del ataque DoS contra servidores a baja tasa se basa en eliminar la ventaja que tiene el atacante sobre los clientes legítimos. Esto implica eliminar el conocimiento que posee el atacante acerca del momento en el que se generará una posición libre en la cola de servicio. Para esto, se propone la inclusión de una inteligencia en la gestión de la cola de servicio, una llamada *política de gestión de colas*.

Denominamos política de gestión de colas a aquel mecanismo que realiza un manejo de las colas de servicio con un fin concreto. En nuestro caso, descartar o priorizar peticiones de la cola de servicio ante la aparición de determinados estados en la misma. La elección de las peticiones que son descartadas o priorizadas sigue, también, una estrategia determinada. Esta actuación pretende conseguir ecuanimidad entre los usuarios, evitando, por tanto, que los clientes maliciosos capturen mayor número de posiciones en la cola que los clientes legítimos.

Para incluir la funcionalidad de las políticas de gestión de colas, al modelo de servidor presentado anteriormente en la Fig. 1 se añade un módulo denominado *módulo de gestión*, que será el encargado de aplicar la política de gestión de colas (Fig. 2). Por tanto, se añade también un evento más a las respuestas del servidor, el *evento RCM solicitud de modificación de conexión* (de las siglas en inglés *Request Connection Modification*). Este evento se produce en el servidor cuando el módulo de gestión determina que una conexión de las encoladas debe ser priorizada o descartada de la cola de servicio. Éste también podría dar como resultado el envío, al cliente correspondiente, de un mensaje *RCM* indicando que su petición será descartada o priorizada. Éste envío es opcional y dependerá de la política de gestión de llevarlo a cabo o no.

En resumen, el funcionamiento del módulo de gestión consiste en monitorizar la cola de servicio y, tras la recogida de cierta información de interés sobre el estado de la cola, procesarla y actuar de acuerdo a su política interna. La política interna que rige el módulo de gestión es la clave del éxito de esta solución. Una buena política de gestión de colas será aquella que extraiga el mayor número de peticiones malintencionadas del atacante, descartando, además, el menor número de peticiones de los clientes legítimos. Si se consigue que los clientes puedan seguir realizando conexiones con normalidad se habrán logrado mitigar los efectos del ataque DoS.

A. Diseño de una política de gestión de colas

A continuación se propone una política de gestión de colas desarrollada para comprobar la efectividad de este tipo de medidas frente a los ataques LoRDAS. El proceso seguido por la política propuesta se puede ver gráficamente en la Fig. 3, en la que se presenta el diagrama de flujo de la misma.

La notación seguida es la siguiente:

- `n_act`: número de peticiones encoladas en la cola de servicio.
- `n_max`: número de posiciones totales de la cola de servicio.
- `n_umb`: umbral de la cola de servicio a partir del cual se aplicará una determinada estrategia de descarte.
- `N`: número de posiciones de la cola que serán descartadas (`N<n_act`).

La política desarrollada se denomina *Descarte Aleatorio de Conexiones* (en inglés *Random Throw Connections RTC*). Se basa en monitorizar la ocupación de la cola hasta que ésta supere el umbral establecido `n_umb`. Entonces, cuando `n_act>n_umb`, se eligen `N` de las conexiones encoladas al azar y se descartan. El umbral se especifica como una proporción del tamaño máximo de la cola.

Se presenta a continuación el pseudo-código de la política RTC:

```
repetir siempre{
   si n_act es mayor que n_umb{
      elegir al azar N peticiones de n_act;
      descartarlas;
   }
}
```

Como se puede comprobar, esta política es una estrategia muy sencilla que no ha sido diseñada para optimizar su rendimiento, sino que se presenta con el único objetivo de evaluar si este tipo de medidas es eficaz frente al ataque LoRDAS. Por tanto, la exploración exhaustiva de las posibles medidas de este tipo no es el objetivo de este trabajo.

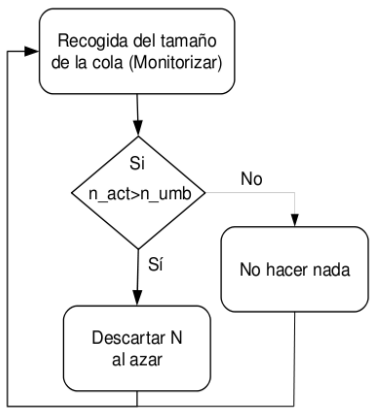

Fig. 3: Diagrama de flujo de la política de gestión de colas RTC.

Como ejemplo de lo anteriormente dicho, nótese que, en la estrategia propuesta, la monitorización del estado de la cola se basa exclusivamente en la observación del número de posiciones ocupadas en cada instante. Sin embargo, podrían existir otras alternativas como, por ejemplo, monitorizar el número de posiciones de un mismo usuario, el número de posiciones de diferentes usuarios dentro de una misma subred, etc.

B. Mitigación de los efectos DoS

Ahora se aborda la cuestión de cómo influye la política RTC en la mitigación del efecto dañino del ataque LoRDAS, es decir, cómo tras la aplicación de la política de gestión de colas, un cliente legítimo puede acceder a los recursos del servidor con una mayor disponibilidad que en el caso de que esta política no fuera aplicada.

Como se ha visto anteriormente el ataque LoRDAS se basa en dos procesos principalmente:

- Mantener las colas siempre *saturadas*, sin posibilidad de que nuevos clientes puedan hacer conexiones.
- *Capturar* todas las posiciones que se vayan habilitando en la cola de servicio prediciendo los instantes en que éstas se generan.

La política RTC dificulta ambos procesos. La saturación de las colas se hace más compleja para el atacante, ya que la política determina un tamaño umbral a partir del cual, cuando el número de peticiones en cola lo supera, un cierto número de peticiones de las encoladas son expulsadas. De esta manera, se dificulta la tarea de mantener las colas siempre saturadas, ya que el servidor libera posiciones siempre que se supere el umbral y estas posiciones pueden ser ocupadas tanto por el atacante como por cualquier cliente legítimo.

Adicionalmente, esta política dificulta la captura de todas las posiciones de la cola por parte del atacante ya que, aunque éste pueda predecir los instantes en que el servidor envía respuestas a los clientes y, por tanto, las posiciones de la cola liberadas de una manera normal por el servidor, no sucede lo mismo para las peticiones que son expulsadas por la política de gestión de colas, ya que esta expulsión genera `N` posiciones libres. Por tanto, con la aplicación de esta política, se consigue que la vulnerabilidad que supone conocer la aparición de una posición libre en la cola de servicio tras cada petición servida por el servidor disminuya de importancia, ya que ahora, no es éste el único modo en el que aparecen posiciones libres.

Esta política de gestión de colas, aunque sencilla, es capaz de hacer frente a complejos mecanismos utilizados por los atacantes para predecir el momento en el que una posición será liberada de la cola. En los resultados mostrados en la Sección V se expondrá en qué medida es capaz de reducir realmente los efectos del ataque LoRDAS.

IV. ENTORNO PARA LA INCLUSIÓN DE POLÍTICAS DE GESTIÓN DE COLAS

Para evaluar la viabilidad de la implementación de la política RTC o cualquier otra política de gestión de colas en un sistema real se diseña y desarrolla un entorno que permite la inclusión de éstas. En este trabajo nos centramos en Linux por ser éste un sistema operativo de código abierto, lo que permite modificar su núcleo adaptándolo a nuestras necesidades.

En esta implementación se supondrá que las peticiones del ataque LoRDAS corresponden a simples peticiones de conexión TCP SYN a un puerto de escucha de la aplicación, tal y como se sugiere en [5]. Por ello, previamente a la implementación de la solución, se estudia cómo se realiza en el núcleo de este sistema operativo el establecimiento de conexión del protocolo TCP/IP.

A. Cola de servicio para TCP en el núcleo 2.6 de Linux

Utilizamos el núcleo en su versión 2.6.24 por ser el más actual en el momento de inicio de este trabajo. Para este núcleo la implementación de la cola de servicio para conexiones TCP se divide en dos colas diferentes [7] (ver Fig. 4):

- *Cola de conexiones incompletas.* Tiene una entrada por cada segmento TCP SYN recibido. Para estas conexiones el servidor se encuentra en el estado SYN_RCVD.
- *Cola de conexiones completadas*. En ésta se encuentra una entrada por cada conexión para la que se ha completado correctamente la negociación en tres pasos, con lo que se encuentra en el estado ESTABLECIDA.

Si bien la cola de conexiones incompletas afecta al comportamiento de la entrada de las conexiones a la aplicación, la implementación que realmente corresponde a la cola de servicio del modelo del servidor es la de conexiones completadas. A esta cola se accede por dos vías diferentes (Fig. 4):

- *Acceso desde la red:* se añaden conexiones a la cola de conexiones completadas al pasar del estado SYN_RCVD al estado ESTABLECIDA. Esto se produce al recibir el ACK como respuesta final de la negociación en tres pasos, siempre y cuando quedase espacio en la cola.
- *Acceso desde la aplicación:* se extraen peticiones por parte de la aplicación por medio de la llamada al sistema *accept*. En este caso, la primera conexión de la cola pasa al espacio de usuario.

La cola de conexiones completadas es una estructura del tipo `request_sock_queue` (ver Fig. 5) que representa una lista doblemente enlazada. Los punteros de la lista apuntan a

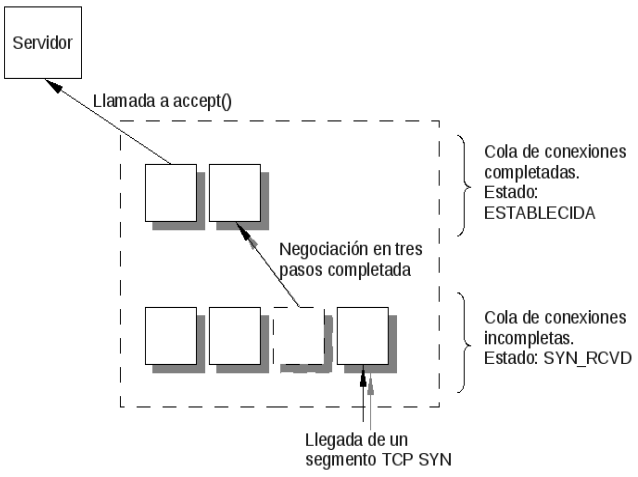

Fig. 4: Colas de conexiones en el núcleo 2.6 de Linux.

estructuras del tipo `request_sock`, que son los elementos que componen la cola y que, por tanto, representan las conexiones establecidas. Éstos contienen básicamente un puntero al siguiente elemento de la cola (`dl_next` en Fig. 5) y un puntero a una estructura `sock` que es la que contiene la información de la conexión.

La sección de código del núcleo en la que se encuentra implementada la fase final de la negociación en tres pasos corresponde a la función `tcp_v4_syn_recv_sock`. Esta función es especialmente adecuada para incluir en ella las políticas implementadas porque es la que se ejecuta tras el término correcto de la negociación en tres pasos, devolviendo el nuevo `socket` que será añadido a la cola de peticiones completadas. Es, por tanto, un lugar estratégico desde el que monitorizar la cola de conexiones establecidas y tomar, dependiendo de la política de gestión de colas que se haya implementado, una decisión u otra.

B. Arquitectura de la implementación.

Decidido el lugar en el que se incluirá la modificación del núcleo, se estudia el sistema operativo Linux y las alternativas que existen para modificar aquél.

El núcleo del citado sistema operativo está organizado siguiendo una arquitectura monolítica, en la cual todas sus partes (sistemas de ficheros, manejadores de dispositivos, protocolos de red, etc.) están enlazadas como una sola imagen que es la que se carga y ejecuta en el arranque del sistema.

Esta estructura lo hace bastante eficiente, ya que disminuye los cambios de contexto, es decir, los cambios de espacio de direcciones de usuario a núcleo, para poder ejecutar instrucciones en modo privilegiado. Por contra, ser un sistema monolítico implica también que existe una menor fiabilidad. A modo de ejemplo, la gestión de ficheros del sistema puede tener acceso a la gestión de la tarjeta de red y esto podría ser nefasto e incluso constituir una vulnerabilidad a explotar desde el punto de vista de un posible atacante. También podría dar lugar a un sistema poco flexible ya que, cualquier funcionalidad que se le quisiera añadir al núcleo del sistema, requeriría una recompilación completa del mismo. Para mitigar en gran medida este problema, en la versión 1.2 de Linux, se incluyó el soporte para la carga dinámica de módulos en el núcleo (en inglés *Loadable Kernel Module LKM*). Este nuevo avance permitió la incorporación de una modificación al núcleo sin la necesidad de reiniciar el sistema completo.

La implementación del entorno para la inclusión de políticas de gestión de colas se desarrolla en un módulo del núcleo debido a la versatilidad que éstos ofrecen. Además, la implementación de este entorno en un módulo del núcleo permite un nivel mayor de abstracción, gracias al cual el usuario no tiene porqué introducir el código del entorno en un archivo del núcleo y dentro de él en una función concreta. Tan sólo es necesario implementar la política de gestión de colas en el lenguaje de programación C en un archivo diferente.

C. Módulos de seguridad de Linux (LSM) y garfios.

El núcleo está preparado para insertar módulos en escenarios concretos como *drivers* para dispositivos, sistemas de ficheros o llamadas al sistema. Realizar modificaciones en otro sentido, como por ejemplo en otro lugar concreto del núcleo, requiere de un análisis exhaustivo del código.

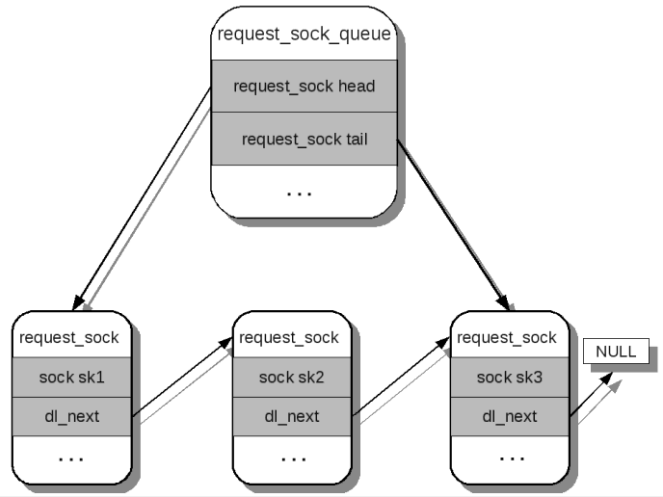

Fig. 5: Estructura de la cola de conexiones establecidas (`request_sock_queue`).

Para incluir una política de gestión de colas se propone modificar el núcleo en la función `tcp_v4_syn_recv_sock`. Sin embargo, modificar el núcleo implicaría un núcleo diferente para cada política de gestión de colas desarrollada, con lo que la solución escogida es una adaptación de un modo concreto de modificación mediante módulos descrito en [8]. Esta modificación es posible gracias al concepto de los garfios (*hooks*), utilizado en los módulos de seguridad de Linux o LSM (del inglés: *Linux Security Modules*).

En el trabajo citado [8] se hace uso de una llamada al sistema creada para este fin y de unas bibliotecas que no son necesarias para la inclusión de políticas de gestión de colas. Por este motivo, ha sido necesario adaptar el método de modificación del núcleo utilizado.

El objetivo de LSM se basa en que ciertas políticas de seguridad sean aplicadas en puntos concretos del núcleo en los que se comprueban, en caso de estar cargado el núcleo, una serie de condiciones para permitir el acceso o no de una petición. Este objetivo es fácilmente extrapolable a las necesidades de un entorno que permita la inclusión de políticas de gestión de colas. Entrando en un mayor nivel de detalle se puede resumir que el objetivo de un garfio es el de, en el punto concreto del código en el que se incluya, llamar a la función del módulo que analiza el estado de la información del núcleo y decide qué actuación llevar a cabo. La gran ventaja que brinda la utilización de los garfios frente a la modificación convencional por medio de módulos es que permiten modificar el núcleo en puntos concretos de su código y, como se ha comentado en el inicio de este punto, éste es precisamente el requisito necesario para la implementación del entorno para la inclusión de políticas de gestión de colas.

Una modificación mediante garfios es una solución intermedia entre la modificación directa del núcleo y un módulo del núcleo. Esto es así porque es necesaria una modificación del núcleo inicial en la que se incluye el garfio en el punto en el que se llamará a la política de gestión de colas mientras que toda la implementación relativa a la política en sí se realiza en un módulo del núcleo.

La modificación llevada a cabo en el núcleo consiste en un elemento de programación que realizará la función de garfio: una estructura denominada `access_control` (cuando se hable de una instanciación concreta de dicha estructura se la llamará `access_cont`). El contenido de esta estructura es de un único elemento: un puntero a función. Este puntero a función es el encargado de apuntar a la llamada `función_inútil`, que no aplica ninguna política hasta que el módulo es cargado, momento en el que pasa a apuntar a la `función_útil` descrita en el módulo que implementa la política de gestión de colas (Fig. 6).

Todo módulo del núcleo debe tener, al menos, dos funciones: `init_module` (ejecutada al cargar el módulo) y `cleanup_module` (ejecutada al descargar el módulo). Para que el garfio funcione correctamente estas funciones deben realizar las siguientes acciones:

- `init_module`: en esta función se guarda el valor del puntero a función de la estructura `access_cont` (que apunta a la `función_inútil`) y se apunta a la `función_útil`.
- `cleanup_module`: se apunta de nuevo el puntero a función a la `función_inútil`, utilizando el valor almacenado en la función `init_module`.

Adicionalmente, para la modificación existen unas consideraciones a tener en cuenta:

- Incluir el fichero de definición de la estructura `access_control`.
- Definir una instancia de esta estructura con el mismo nombre que se definió en el núcleo (`access_cont`) para que se reconozca como el mismo objeto.
- Esta instancia debe estar precedida por la palabra clave `extern`, que indicará que este objeto ha sido implementado externamente.

V. EVALUACIÓN DE LA POLÍTICA RTC

Expuestos los conceptos básicos del modelo de servidor y el ataque LoRDAS, explicada la política de gestión de colas implementada, presentadas las estructuras del núcleo de Linux que intervienen en el desarrollo del entorno para la inclusión de políticas de gestión de colas y analizado en sí el entorno y el método utilizado para incluirlo en el núcleo del sistema operativo Linux, se puede abordar la evaluación de la política propuesta.

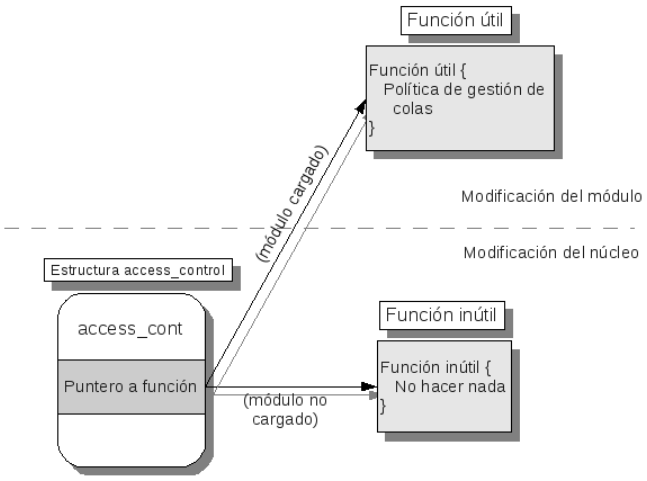

Fig. 6: Funcionamiento de un garfio.

A. Descripción del entorno de pruebas

Para hacer posible la evaluación de la política de gestión de colas y, por ende, del entorno de inclusión de medidas de defensa ante ataques DoS, se implementa un sistema cliente-servidor multihebra. Éste tendrá la finalidad de recoger datos del funcionamiento del sistema frente a un ataque LoRDAS, siendo aplicada la política de gestión de colas como sin serlo.

Se implementa un servidor que acepta conexiones cada 5 segundos. Esto implica que cada 5 segundos, si hay alguna conexión en la cola de conexiones establecidas (o en el modelo general en la cola de servicio), se atenderá la primera conexión de la cola, habilitándose, por tanto, una posición libre en ésta.

Se generan dos programas cliente. Uno de ellos será el atacante y otro el cliente legítimo. Ambos clientes, para poder ser comparados, tendrán la misma tasa de generación de peticiones de conexión, siendo ésta de 1 conexión cada 5 segundos. Esto provocará un desbordamiento de la cola de conexiones establecidas, puesto que a ésta llegará, en media, una petición de conexión cada 2,5 segundos, mientras que la tasa de servicio es de 1 conexión cada 5 segundos. El objetivo es evaluar si ambos clientes tienen las mismas oportunidades de ser servidos y, por tanto, si se encuentran en una situación justa, o si, por el contrario, el atacante es servido con una prioridad superior.

El cliente legítimo es implementado de modo que no genera solicitudes de conexión de una manera determinista, sino que el tiempo entre solicitudes sigue una distribución de de Poisson de media 5 segundos. Para representar la acción del atacante, éste utilizará algún mecanismo que le permita tener ventaja sobre el cliente legítimo y así, dificultar o impedir las conexiones de éste (objetivo del ataque DoS). Concretamente, el cliente malicioso realiza un ataque de tipo LoRDAS como se ha descrito con anterioridad. En nuestro caso la implementación del ataque consiste en enviar una petición de conexión en el momento justo en el que el servidor finaliza el servicio de una petición y, en consecuencia, extrae una nueva petición de la cola de servicio. Como se ha comentado previamente en la Sección II, el conocimiento de la liberación de posiciones de la cola se consigue realmente a través de un complejo mecanismo [5] [6]. En nuestro caso solamente se pretende simular que esto sucede, de modo que en los instantes en los que el servidor acepta una conexión y, por tanto, libera una posición de la cola de conexiones establecidas, nos limitaremos a escribir en un archivo llamado *secreto*. El atacante monitoriza este archivo y, en el instante en el que descubre una modificación en él, realiza un intento de conexión con el servidor. De esta forma, el atacante aumenta la probabilidad de adelantarse al cliente legítimo, capturando la inmensa mayoría de las posiciones de la cola y consiguiendo denegar el servicio.

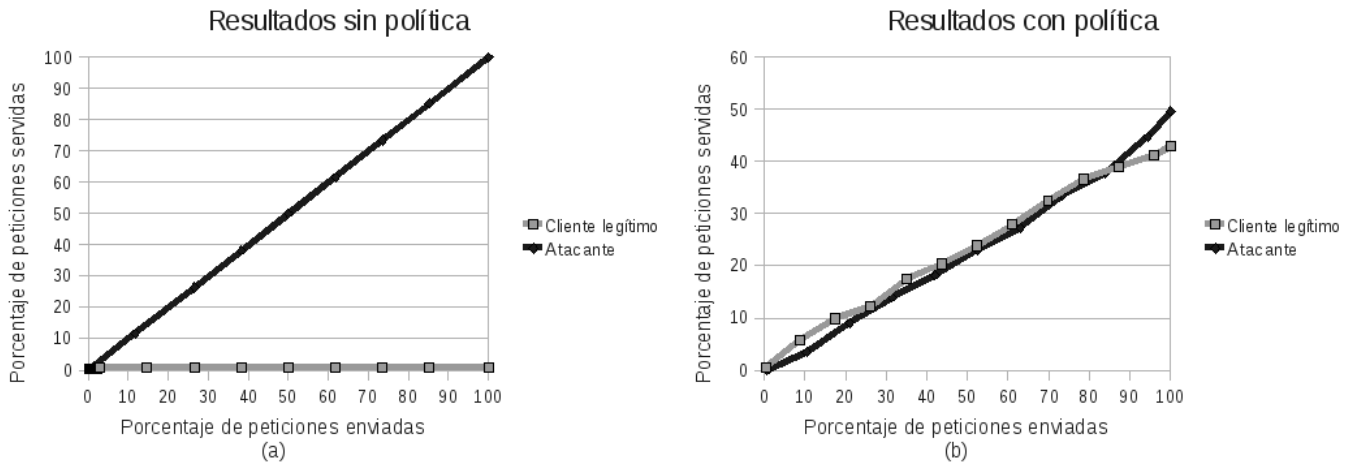

Fig. 7: Porcentaje de conexiones servidas con un tamaño de cola igual a 3: (a) sin aplicar la política y (b) con la política activa.

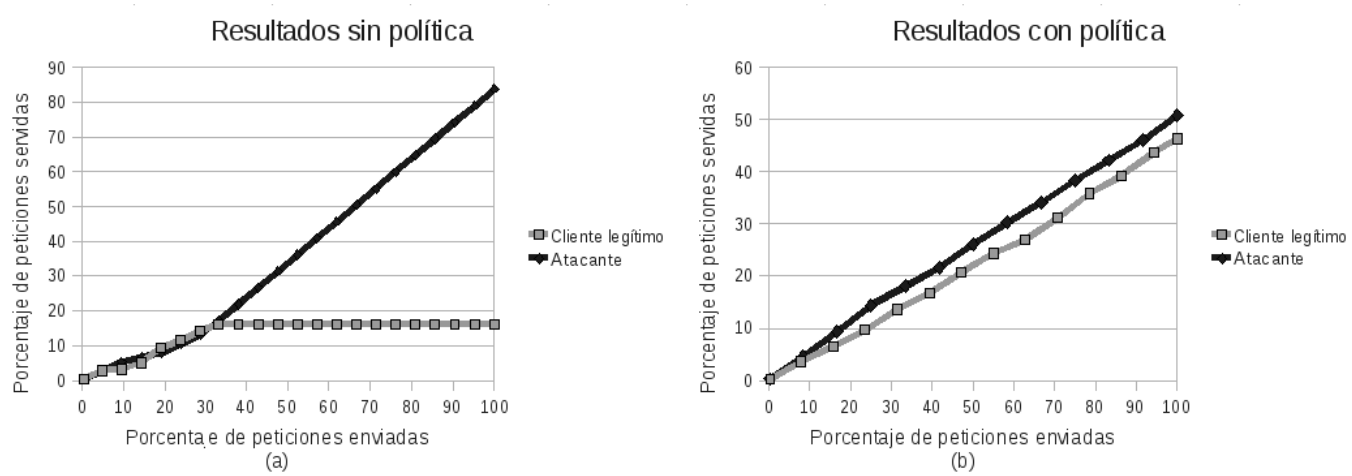

Fig. 8: Porcentaje de conexiones servidas con un tamaño de cola igual a 51: (a) sin aplicar la política y (b) con la política activa.

B. Resultados

Las pruebas realizadas consisten en lanzar la aplicación servidora junto con ambos clientes (legítimo y atacante), con el módulo descargado en primera instancia (política no activa). Tras un tiempo de ejecución suficientemente elevado para que la información recogida sea relevante, se interrumpen todas las aplicaciones y se lanzan de nuevo, pero en esta ocasión se carga el módulo, lo que implica que se aplicará la política de gestión de colas. El entorno implementado imprime en su ejecución el número de peticiones servidas con respecto a los intentos realizados.

Se realizan 2 pruebas: una con tamaño de cola de conexiones establecidas igual a 3 y otra con un valor igual a 51. En la Fig. 7 se ilustran los resultados del tamaño de cola 3. Para tamaño 51 éstos se pueden ver en la Fig. 8.

La notación utilizada puede resumirse en:

- $E(t)$: número de peticiones enviadas desde el inicio de la evaluación hasta el instante t.
- E_T: número de peticiones enviadas en total.
- $S(t)$: número de peticiones servidas desde el inicio de la evaluación hasta el instante t.
- S_T: número de peticiones servidas en total.

En las Fig. 7 y 8, el eje de ordenadas representa el porcentaje de peticiones enviadas hasta el instante *t* respecto del total ($100*E(t)/E_T$) y el eje de abscisas representa el porcentaje de peticiones servidas hasta el instante *t* respecto al total ($100*S(t)/S_T$). Esto quiere decir que todas las representaciones recorrerán todo el eje de ordenadas, ya que van desde un porcentaje de intentos cero al total (100%). Se presentan los resultados en porcentaje precisamente para que el eje de ordenadas sea el mismo tanto para el cliente legítimo como para el atacante. Esto es de utilidad ya que no realiza los mismos envíos de peticiones de servicio un cliente que el otro aún teniendo la misma tasa. En efecto, el cliente legítimo, al generar las conexiones según una distribución de Poisson, en ocasiones realizará más de un intento cada 5 segundos y en otras ocasiones menos, aunque en media el número de intentos será el mismo.

Los resultados obtenidos para la ejecución de las pruebas, ya sea con tamaño de cola igual a 3 o igual a 51, en caso de no aplicarse la política de gestión de colas, demuestran que el atacante consigue realizar con éxito su ataque, ya que el cliente no es capaz de realizar conexiones a partir del instante de saturación de la cola. Esto se debe a que el atacante captura con éxito todas las posiciones que se liberan de la cola de conexiones establecidas y mantiene siempre saturada dicha cola. Para el caso de 51 posiciones de cola, aún sin activar la política de seguridad, sucede lo mismo, pero con la diferencia de que la cola de conexiones establecidas es de un

tamaño mayor, permitiendo de este modo observar el período en el que la cola es saturada (hasta un porcentaje de peticiones enviadas ≈ 30%). A partir de este porcentaje, el atacante consigue capturar todas las posiciones que se liberan de la cola evitando el acceso a ésta del cliente legítimo y denegando, por tanto, su servicio. Este hecho puede verse claramente en las Figs. 7(a) y 8(a) ya que el cliente legítimo, una vez se han saturado las colas, no varía su porcentaje de conexiones servidas.

En el caso de ser cargada la modificación del núcleo y, por consiguiente, de aplicarse la política de gestión de colas RTC, el atacante no obtiene el mismo éxito en su ataque DoS, como puede verse en las Figs. 7(b) y 8(b). De hecho, al término de la ejecución de la aplicación, el porcentaje de peticiones enviadas respecto al de peticiones servidas de ambos clientes ronda el 50 %, lo que implica que la política ha conseguido otorgar las mismas posibilidades de acceso a la cola a los dos clientes y mitigar, por consiguiente, los efectos del ataque LoRDAS.

VI. Conclusiones y trabajo futuro

Como conclusión de este trabajo, podemos resaltar que se ha demostrado la eficacia de la aplicación de políticas de gestión de colas frente a ataques DoS contra servidores a baja tasa. Para evaluar esta eficacia se ha implementado una política de gestión de colas (RTC) que, aunque sencilla, ha sido capaz de mitigar en gran medida los efectos de los ataques LoRDAS. También es reseñable el diseño e implementación de un entorno que hace factible la aplicación de este tipo de medidas de defensa en un sistema real, concretamente el núcleo del sistema operativo Linux.

Como trabajos futuros podemos destacar:

- La generación de políticas de seguridad que introduzcan probabilidades de descarte en casos concretos, que utilicen mecanismos de detección de atacantes para realizar el descarte de la conexión, etc. La evaluación de estas políticas para determinar aquella que dificulte, de la mejor forma posible, los efectos de la realización de un ataque DoS.
- La adaptación del entorno para la inclusión de medidas de defensa frente a ataque DoS basadas en políticas de gestión de colas a otros sistemas operativos de código abierto, como puede ser OpenBSD.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el MICINN mediante el proyecto TEC2008-06663-C03-02.

Referencias

[1] Kuzmanovic, A., Knightly E., Low-rate TCP-targeted denial of service attacks and counter strategies. IEEE/ACM Trans Network 14(4): pp. 683-96, 2006.

[2] Guirguis M, Bestavros A, Matta I, Zhang Y. Reduction of quality (RoQ) attacks on internet end-system, INFOCOM 2005. In: 24th Annual joint conference of the IEEE computer and communications societies; 2005. pp. 1362-72.

[3] Guirguis M, Bestavros A, Matta I, Zhang Y. Reduction of quality (RoQ) attacks on dynamic load balancers: vulnerability assessment and design tradeoffs, INFOCOM 2007. pp. 857-65.

[4] Chan MC, Chang E, Lu L, Ng S. Effect of malicious synchronization, ACNS, Singapore, Jun 6-9, 2006. In: Springer Lecture Notes in Computer Science, vol. 3989; 2006. p. 114-29.

[5] Maciá-Fernández, G., Díaz-Verdejo, J.E., García-Teodoro, P., Evaluation of a Low-Rate DoS Attack Against Application Servers, *Computers & Security*, Vol. 27; pp. 335-354, 2008.

[6] Maciá-Fernández G., Díaz-Verdejo J.E., García-Teodoro P., Evaluation of a low-rate DoS attack against iterative servers. Computer Networks 2007;51(4):1013-30

[7] Stevens, W.R., Fenner, B., and Rudoff, A. M., (2004) UNIX Network Programming, the sockets networking API, Volume 1. Addison-Wesley Professional, 3rd edition, ISBN: 0-13-141155-1.

[8] Wright, C., Cowan, C., Morris, J., Smalley, S., Kroah-Hartman, G., Linux security modules: General security support for the Linux kernel, Proceedings of the 11th USENIX Security Symposium, pp. 17-31, 2002.