VERIFICANDO LA INTEGRIDAD DE PROCESOS EN LINUX

Roberto Gómez, Erika Saucedo[1]

Actualmente la seguridad computacional juega un papel importante en el desempeño y desarrollo de las empresas. Muchos trabajos proponen aplicar el comportamiento del sistema inmunológico en el área de seguridad computacional. La protección que brinda el sistema inmunológico se basa en diferenciar lo propio de lo ajeno. La comprobación de la integridad de procesos es una forma de poder distinguir un proceso del sistema de uno ajeno al sistema. Este tipo de enfoques no ha sido muy explorado y puede aportar resultados interesantes en relación con el monitoreo de los sistemas. En este trabajo presentamos una forma de verificación de la integridad de procesos mediante el uso de huellas digitales.

Palabras clave: seguridad, criptografia, autenticación, sistemas operativos, redes

1. Introducción

La administración de la seguridad de los sistemas es una tarea muy importante en los sistemas informáticos actuales. Esta tarea se lleva a cabo de forma manual y para detectar ataques es necesaria la utilización de herramientas de monitoreo y detección de intrusos. El encargado de usar estas herramientas y tomar acciones de prevención y de respuesta es el administrador de sistemas o de seguridad. Esta tarea es muy tediosa, no está libre de errores y su efectividad depende del tiempo que tarda el administrador en reaccionar ante un ataque.

Una de las tendencias en el área computacional es que los sistemas computacionales sigan un comportamiento similar al del humano; así han surgido términos como *inteligencia artificial, redes neuronales, vida artificial* muy conocidos en el ámbito computacional. Un área aún no estudiada en su totalidad es la de *sistemas inmunológicos* basada en el concepto de sistema inmunológico del cuerpo humano, Este defiende al cuerpo contra agentes infecciosos y es capaz de reconocer un agente infeccioso empleando un mecanismo de defensa multinivel.

Se ha planteado por varios investigadores [1, 2, 3, 4, 7, 8] el aplicar el comportamiento del sistema inmunológico en el área de seguridad computacional. En particular es interesante la forma en que el sistema inmunológico resuelve el problema de distinguir lo *propio* de lo *ajeno*. Nosotros planteamos el problema desde el punto de vista de la integridad de un proceso. Si la integridad de un proceso se ve comprometida, consideramos que el proceso no es propio al sistema y se le tratará como un proceso ajeno a este.

Más específicamente, este trabajo propone un sistema para verificar la integridad de los procesos que están protegiendo un sistema de cómputo o una red. Esto permite que el sistema pueda saber cuando un proceso está realizando las operaciones para las que fue diseñado; es decir, no ha sido modificado en forma maliciosa. Para lograr lo anterior se añadió un campo de verificación a nivel núcleo; se escribió un programa de verificación.

[1] Dr. Roberto Gómez Cárdenas, Profesor Departamento Ciencias Computacionales, ITESM-Campus Edo. de México, rogomez@itesm.mx
Lic. Erika Saucedo, Consulto Ernst & Yong, erika.saucedo@mx.ey.com

Nuestro trabajo esta inspirado del sistema inmunológico, el cual es el sistema encargado de proteger al cuerpo de los patógenos externos como virus, bacterias y parásitos e internos como células viejas y tumores. Para lograr su objetivo lleva a cabo una defensa multinivel contra los invasores.

La defensa multinivel se compone de dos niveles de defensa. El primer nivel de defensa conocida como inmunidad innata es la primera respuesta tras la exposición a un agente extraño. Este nivel de defensa se compone de barreras (piel, mucosas, secreciones), el sistema fagocitario, el llamado complemento, los mecanismos humorales, las células asesinas naturales (NK), los eosinófilos. El segundo nivel de defensa o inmunidad adquirida actúa cuando el primer nivel de defensa ha sido superado y el antígeno entra al cuerpo. En este caso existen células inmunológicas llamadas linfocitos y pagocitos encargadas de reconocer y eliminar al invasor; cuando un antígeno invade el cuerpo sólo algunas células inmunológicas pueden reconocer al invasor lo que deriva en la destrucción o neutralización del antígeno. Esta información se guarda en una memoria para uso posterior esto se conoce como respuesta secundaria.

Los inmunólogos describen el problema que resuelve el sistema inmunológico como la distinción de lo propio de lo ajeno. Lo propio representado por las células y moléculas del cuerpo y lo ajeno representado por cualquier material externo, particularmente bacterias, parásitos y virus.

Actualmente, el sistema inmunológico natural es un área de gran interés de investigación debido a su potencial para procesar información. En particular, desarrolla muchos cálculos complejos en forma distribuida y paralela. Para explicar el fenómeno del sistema inmunológico existen varias teorías y modelos matemáticos basados principalmente en el empleo de ecuaciones diferenciales. Por otra parte, son cada vez mas los modelos computacionales que intentan simular componentes del sistema inmunológico. Los modelos computacionales basados en el sistema inmunológico más usados por los investigadores son el modelo de red inmunológica y el algoritmo de selección negativa con sus respectivas alternativas.

Las aplicaciones de los modelos anteriores crecen día a día. Entre ellas podemos mencionar; la detección y diagnóstico de fallos, reconocimiento de patrones e imágenes, verificación de firmas, seguridad computacional entre otras, ([9], [10]).En particular, nuestro interés se centra en las aplicaciones de estos sistemas al área de seguridad computacional.

La seguridad de sistemas computacionales depende de actividades como las de detectar usuarios no autorizados para un recurso específico, mantener la integridad de los archivos de datos, prevenir la expansión de virus, etc.

El problema de proteger un sistema computacional puede verse como una instancia del problema general de distinguir lo propio de lo ajeno. En este caso lo propio esta constituido por usuarios autorizados, datos íntegros, etc. y lo ajeno estaría representado por datos corruptos, usuarios no autorizados para hacer uso de los recursos de un sistema, virus, etc.

En este caso, también podemos representar las respuestas con las que cuentan los sistemas de seguridad computacional. Así nuestras respuestas innatas o específicas estarían constituidas por los firewalls, la criptografía, etc., es decir esta es nuestra primer respuesta para proteger el sistema ante un intruso. La respuesta adquirida encargada de aprender a reconocer intrusos nuevos y contar con una memoria de intrusos previos.

Existen muchas características del sistema inmunológico que se consideran interesantes en el ámbito de seguridad computacional. El cuerpo tiene una protección multicapas contra material ajeno a él; en este sentido muchos sistemas computacionales monolíticos en el sentido que definen una periferia dentro de la cual toda actividad es confiable. Cuando el sistema básico de defensa es violado en pocas ocasiones existe un mecanismo de respaldo para detectar la violación. Por otra parte el sistema inmunológico provee la detección y memoria altamente distribuida de patógenos externos lo puede ser explotado por un sistema computacional. Otra característica importante es que cada individuo en una población tiene un conjunto único de protección si esto lo llevamos al ámbito de seguridad

computacional, es evidente que la protección de un sistema en muchas ocasiones incluye a varios sitios incluyendo varias copias de software y varias computadoras en una red; cuando se encuentra una forma de anular la detección en un sitio, todos los sitios quedan vulnerables. Una forma de protección es proveer a cada sitio protegido un conjunto único de detectores. Una cuarta característica es la detección de material externo no visto previamente esto es, en el caso de infecciones nuevas el sistema inmunológico inicia una respuesta primaria proveyendo detectores nuevos para la infección; si una vez mas llevamos esto al área de la seguridad computacional muchos virus y métodos de detección de intrusos buscan sólo por patrones conocidos dejando a los sistemas vulnerables a ataques nuevos. Finalmente, la detección imperfecta no todos los antígenos son detectados de forma correcta por detectores existentes, el sistema inmunológico utiliza dos estrategias para resolver este problema, aprendizaje (durante la respuesta primaria) y la detección distribuida. Así la confiabilidad del sistema se garantiza a relativo bajo costo y con comunicación mínima a través de componentes.

Dentro de las aplicaciones que se han realizado en torno a seguridad computacional podemos mencionar la detección de virus y el monitoreo de procesos UNIX. En el primer proyecto Forrest [2] utiliza el algoritmo de selección negativa para detectar cambios en los datos y programas protegidos. Se realizaron varios experimentos bajo el ambiente DOS con distintos tipos de virus; y se mostró que se puede detectar con facilidad la modificación de archivos de datos así como la modificación por virus. Comparado con otros métodos de detección de virus, éste ofrece ventajas. En principio es probabilístico, puede ser distribuido y puede detectar virus que no han sido identificados previamente. Sin embargo, por la naturaleza de la información que se almacena en una computadora, este está método limitado solo a datos y software protegido o estático.

En [3] Forrest utilizó el algoritmo de selección negativa para monitorear procesos UNIX. El propósito es detectar usuarios peligrosos en un sistema computacional. El trabajo se basa en da por sentado que las llamadas de sistema de los procesos de root son más peligrosas que las de los procesos de usuario. Cada proceso se representa por su trazo, i.e. lista ordenada de llamadas al sistema usadas por dicho proceso de inicio a fin. Este trabajo mostró que el comportamiento normal de un programa se puede caracterizar por patrones locales en sus trazos y derivaciones de estos trazos pueden ser usadas para identificar violaciones de procesos en ejecución.

Nuestro objetivo es basarnos en las ideas anteriores para implementar un sistema que pueda reconocer procesos propios al sistema de los no lo son. Lo anterior nos obliga a programar el sistema a nivel del administrador de procesos, por lo que es necesario contar con un sistema operativo en el que se pueda acceder al código de su núcleo. Esto nos llevo a elegir Linux como nuestro sistema operativo.

Linux es una implementación de libre distribución UNIX para computadoras personales (PC), servidores, y estaciones de trabajo. Fue desarrollado para el i386 y ahora soporta los procesadores i486, Pentium, Pentium Pro y Pentium II, así como los clones AMD y Cyrix. También soporta máquinas basadas en SPARC, DEC Alpha, PowerPC/PowerMac, y Mac/Amiga Motorola 680x0. Como sistema operativo, Linux es muy eficiente y tiene un excelente diseño. Es multitarea, multiusuario, multiplataforma y multiprocesador; en las plataformas Intel corre en modo protegido; protege la memoria para que un programa no pueda hacer caer al resto del sistema; carga sólo las partes de un programa que se usan; comparte la memoria entre programas aumentando la velocidad y disminuyendo el uso de memoria; usa un sistema de memoria virtual por páginas; utiliza toda la memoria libre para cache; permite usar bibliotecas enlazadas tanto estática como dinámicamente; se distribuye con código fuente lo que permite la modificación del núcleo; usa hasta 64 consolas virtuales; tiene un sistema de archivos avanzado pero puede usar los de los otros sistemas; y soporta redes tanto en TCP/IP como en otros protocolos.

Linux puede manejar procesos en el sistema, cada proceso se representa por una estructura de datos llamada *task_struct*[2]. El vector de tareas es un arreglo de apuntadores a cada estructura de datos *task_struct* en el sistema. Así el número máximo de procesos en el sistema depende del tamaño del vector de tareas; el tamaño por omisión es 512.

[2] Tareas y procesos son términos que Linux utiliza de forma similar

La estructura de datos *task_struct* es una estructura complicada sin embargo sus campos se pueden dividir en áreas funcionales: estado, información de planificación, identificadores, comunicación entre procesos, enlaces, tiempos y temporizadores, sistema de archivos, memoria virtual y el contexto específico del proceso (ver [5]).

El estado de un proceso lleva a cabo sus cambios de estado de acuerdo a las circunstancias. Los procesos Linux tienen los siguientes estados: *ejecución, espera, suspendido* y *zombie*. Se dice que un proceso se encuentra en *ejecución* si el proceso es el proceso actual en el CPU o si está listo para ejecutarse y sólo espera tiempo del CPU. El proceso esta en *espera* si está esperando por un evento o por un recurso. Un proceso está *suspendido* si este ha sido suspendido por una señal. Por último un proceso se encuentra en estado *zombie* si está detenido y por alguna razón aún tiene estructura de datos *task_struct* en el vector de tareas.

La información de planificación es la información usada por el planificador para decidir imparcialmente que proceso en el sistema merece ejecutarse.

Cada proceso en Linux cuenta con identificadores; un identificador de proceso, éste es simplemente un número, un identificador de grupo y un identificador de usuario, éstos se utilizan para controlar el acceso de estos procesos a archivos y dispositivos en el sistema.

Otra área es la comunicación entre procesos, IPC, Linux soporta los mecanismos clásicos de señalización UNIX IPC, tuberías y semáforos y los mecanismos de memoria compartida del Sistema V IPC semáforos y colas de mensajes.

En un sistema Linux ningún proceso es independiente de cualquier otro proceso, por lo que es importante información referente a enlaces de un proceso. Todo proceso en el sistema excepto el proceso inicial tiene un proceso padre. Los procesos nuevos no son creados, éstos son copiados o clonados de procesos previos. Cada *task_struct* que representa un proceso guarda apuntadores a su proceso padre, a sus hermanos y a sus procesos hijos.

El núcleo guarda el tiempo de creación de un proceso y el tiempo de CPU que consume durante su tiempo de vida por lo que en la estructura *task_struct* se debe tener también información acerca de tiempos y temporizadores.

Los procesos pueden abrir y cerrar archivos como lo deseen *task_struct* contiene apuntadores a las descripciones de cada archivo abierto así como apuntadores a dos inodos VFS. Cada inodo VFS únicamente describe un archivo o directorio en un sistema de archivos y también provee una interfaz uniforme a los sistemas de archivos subyacentes. Esta información se encierra en el área de sistema de archivos.

La mayoría de los procesos tienen algo de memoria virtual (los hilos y los demonios no) y el núcleo de Linux debe dejar huella de cómo se está mapeando la memoria virtual a la memoria física del sistema.

Siempre que un proceso esté ejecutando éste está ejecutando registros, stack, entre otros del procesador. Este es conocido como contexto de un proceso y cuando se suspende un proceso, todos los contextos específicos del CPU se deben guardar en la estructura *task_struct* del proceso. Cuando un proceso se reinicia por el planificador su contexto se restablece.

Todas estas áreas engloban la estructura *task_struct* en su totalidad. Cada estructura de datos *task_struct* describe un proceso o tarea en el sistema. Esta se encuentra en el subdirectorio */usr/src/linux/include/linux/sched.h* dicho archivo debe ser modificado para agregar el campo a la estructura *task_struct*.

3. Metodología

La autenticación permite comprobar la identidad de un sistema, de entidades homólogas, o del origen de los datos en modo no conexión, es decir poder comprobar que una entidad en realidad es quien dice ser. Existen varios mecanismos para llevar a cabo la autenticación entre los cuales podemos mencionar; login-password, Kerberos, S/Key, Kryptoknight, huellas digitales, firmas digitales, encriptado convencional, Single Sign On, entre otros.

Para lograr una autenticación de procesos es necesario modificar la estructura *task_struct,* Esta estructura define a un proceso y recompilar el núcleo de Linux. La modificación implica aumentar un campo que sirva como verificador y que no se afecte el comportamiento de los procesos. En éste campo será incluido una huella digital MD5 para poder garantizar la integridad del proceso.

Una *huella digital* es la salida que produce una función hash aplicada a un documento. Un punto importante de las huellas digitales es que un cambio mínimo en el documento produce un cambio evidente en la huella digital. Existen varios algoritmos para producir huellas digitales como SHA-1, MD5, MD4, MD2, RIPE MD-160, HMAC, N-Hash, Havalk, entre otros. Debido a su estabilidad y reputación se escogió MD5 para ser usado en el sistema.

El algoritmo MD5 (RFC-1321) fue desarrollado por Ron Rivest. La base del algoritmo es tomar un archivo o un mensaje de longitud arbitraria y producir una salida de 128 bits conocida como Message-Digest. Con este algoritmo es prácticamente imposible obtener dos mensajes que produzcan la misma huella digital ya que la probabilidad es uno en 2^{64} y también es prácticamente imposible el producir un archivo o un mensaje que arroje una huella predefinida. Una explicación más detallada del algoritmo MD5 y del concepto de huella digital se encuentran en [6].

Linux es un sistema operativo que proporciona el código fuente. Esto permite modificar la estructura *task_struct.* Para lo cual es necesario modificar el archivo */linux/include/linux/sched.h,* ahí en la estructura *task_struct* se le añade la parte de autenticación con la línea *char auth_pro[128]* ya que MD5 genera un compendio de 128 bits. Añadida esta parte, es necesario la inicialización del nuevo campo en la estructura para lo cual se edita el macro INIT_TASK ubicado en el mismo archivo *sched.h*.

La modificación implicó añadir un campo que tomó el rol de verificador de integridad de manera que no se afecte el comportamiento de los procesos una vez añadido el campo. En dicho campo se almacena la huella digital generada por el algoritmo MD5 para poder garantizar la integridad del proceso.

Para identificar los campos que se utilizan para llevar a cabo la verificación de integridad fue necesario identificar aquellos campos que no se modifican durante la vida del proceso. Aunque existen campos que en condiciones normales no son modificados estos pueden modificarse por medio de llamadas al sistema lo que permite que este método de verificación de integridad pueda aplicarse a objetivos específicos en caso de que no se quiera que se hagan ciertas llamadas al sistema. Para efectos de este trabajo solamente se consideraron aquellos datos que permanecen estáticos y que no pueden ser modificados por llamadas al sistema. A continuación se presentan los campos seleccionados para verificar la integridad de un proceso en ejecución.

A. Dentro de la estructura task_struct tenemos los campos que entran en la categoría de identificadores estos cuentan con los campos denominados identificadores efectivos (euid, egid) estos no cambian durante la vida de un proceso y son los que mantienen el identificador con el que el proceso inició y no debe haber ningún cambio en caso que lo haya quiere decir que algo fuera de lo normal está ocurriendo con el proceso.
B. Dentro del área de planificación tenemos otro de los campos que pueden considerarse datos estáticos durante la vida de un proceso; este campo es el que se refiere a la política de planificación (policy) que esta siendo aplicada a este proceso ya sea round robin o first in first out no debe modificarse durante la vida del proceso.
C. Otro de los elementos de la estructura que no deben modificarse durante la vida del proceso es el campo del modo en que se abre un archivo por un proceso (f_mode); este campo se encuentra en la estructura files_struct y apunta hasta 255 archivos.

D. También se tomo en cuenta el código del programa, en la estructura task_struct existe un apuntador a la estructura mm que define el área de memoria virtual de un proceso, esta área contiene el código ejecutable y el área de datos del proceso estableciéndose. El apuntador a la estructura mm_struct tiene apuntadores al área de código y datos del proceso; es en esta área donde se realiza la autenticación en conjunto con los datos fijos antes mencionados de la estructura task_struct.

Los datos anteriores se almacenaron en un archivo que se concatena con el área de código ejecutable y se genera la huella del archivo utilizando MD5.

Es posible realizar la verificación de integridad en tres distintas formas; la primera de ellas se refiere a la verificación de toda la estructura que define a un proceso, la segunda se refiere a la verificación del código ejecutable que define al mismo y finalmente una opción que es una mezcla de las anteriores involucra a datos estáticos de la estructura del proceso y el código del mismo.

La primera opción tiene que ver con la verificación de integridad de toda la estructura sin embargo, esta contiene datos que están en constante cambio. Como ya se mencionó el cambio mínimo en algo de lo que se genera el compendio proporcionará una huella totalmente distinta por lo tanto requeriría un trabajo exhaustivo del procesador y los resultados en cuanto a eficiencia del sistema no diferirían mucho de los que se obtienen con la opción elegida.

La segunda opción, la verificación de integridad del código ejecutable es una mejor opción con respecto a la primera con relación a la eficiencia del sistema; ya que protege de aquellos ataques que implican la modificación de código como los virus. Sin embargo, quedan sin considerar aquellos ataques que se resumen a la modificación de algunos bits de una o más banderas en el estado de un proceso para ganar acceso como root. Estos ataques no quedan cubiertos con esta opción anterior para tal razón decidimos eso consideremos la tercera y última opción.

La tercera opción se refiere a la verificación de integridad tanto del código ejecutable del proceso y los campos en la estructura task_struct que permanecen sin cambio durante el tiempo de vida del proceso para así garantizar la integridad del ejecutable y del estatus del proceso. Ahora definiremos los campos que se utilizaran de la estructura task_struct.

4. Resultados y discusiones

En la Figura 1 se presenta el diagrama básico de la arquitectura de red que se consideró para los propósitos de este trabajo. Se trabajó sobre una red de cinco computadoras, donde todas las máquinas tenían instalado y corriendo el sistema operativo Linux, el cual se modificó para integrar el campo que fungirá como verificador de procesos.

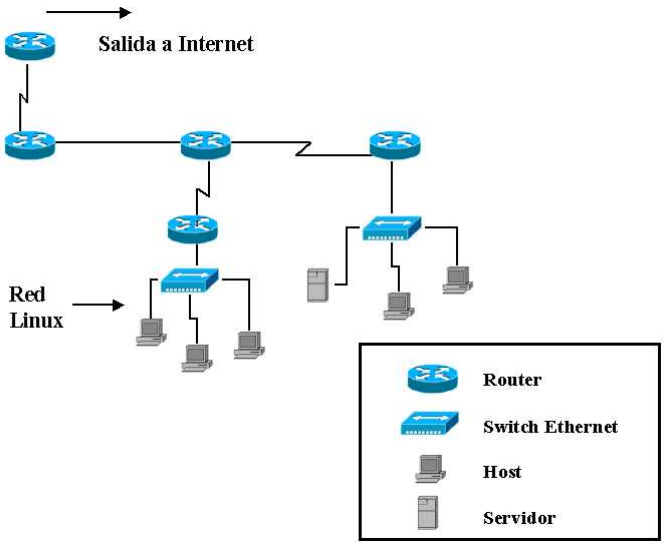

Figura 1. Esquema básico de la red propuesta

En las pruebas realizadas se implementaron tres escenarios. En el primero de ellos se consideró un proceso que esta viajando dentro de la red con un comportamiento similar al de un gusano, verificando su integridad cada vez llegaba a una máquina. En el segundo escenario se llevo a cabo la verificación de todos los procesos que se están ejecutando en la máquina. En el tercero se comprobó la integridad de solamente los procesos que estaban protegiendo a otros procesos del sistema.

En el primer escenario se implementó un proceso que "viajo" a través del puerto 1717. Cada vez que llegaba a una máquina, la integridad de este se verificaba. La verificación no causó ninguna sobrecarga a la máquina y se tuvo éxito en un 100% de los casos en que la integridad del proceso se comprometía.

En el segundo escenario la verificación de integridad se llevó a cabo en todos los procesos que están ejecutándose en ese momento en el sistema iniciando por el proceso inicial init del sistema operativo y siguiendo un recorrido de árboles en forma normal se realiza el escaneo de todos los procesos que se están ejecutando en esa máquina en un momento determinado. En la Figura 2 se presenta el esquema de la forma en que el sistema operativo maneja los procesos y como se presenta el recorrido de árbol sobre los procesos que se están ejecutando. Esta opción implicó un desgaste excesivo del sistema, ya que se consumieron recursos de forma excesiva.

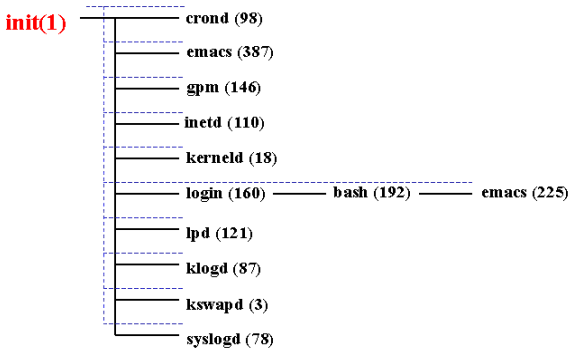

Figura 2. Proceso inicial y procesos ejecutándose

La tercera opción implicó la verificación de integridad de él o los procesos que están protegiendo los procesos normales del sistema y la verificación del proceso “viajero”, dejando el trabajo de resguardar la seguridad de los demás procesos a estos procesos encargados de la seguridad del sistema. Se eligieron hasta un máximo de diez procesos, los cuales no afectaron el desempeño del sistema

Para la verificación de huellas se usó un archivo que contenga la colección de huellas que a su vez haya sido procesado por medio de una función hash. Se generó un archivo que contiene todas las huellas que a su vez esta firmado digitalmente para así evitar que sea modificados sin autorización se utiliza para esto MD5 una vez mas para garantizar la integridad del archivo de huellas.

En la Figura 3 se muestran los elementos básicos para llevar a cabo la autenticación de procesos en el sistema propuesto

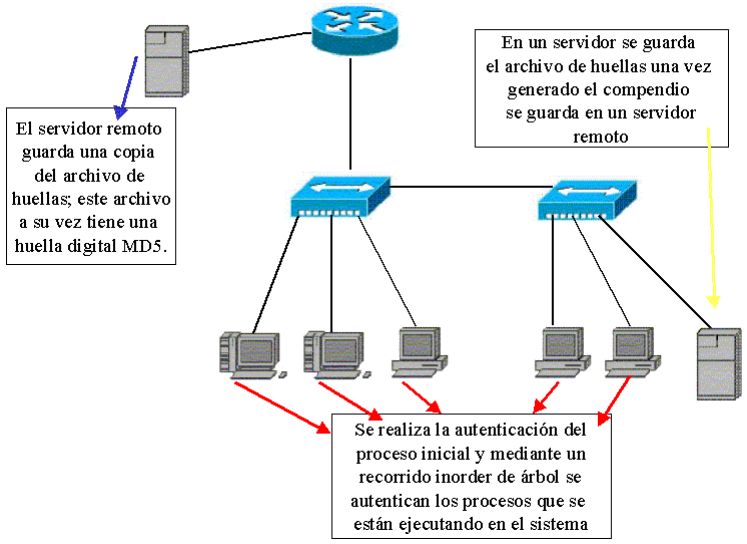

Figura 3. Esquema básico del sistema de autenticación planeado

4. Conclusiones

Se presentó e implementó un esquema de verificación de integridad de procesos en Linux. Para tal efecto se modificó el núcleo, y se llevaron a cabo pruebas con diferentes procesos, unos cuya integridad esta intacta y otros con valores aleatorios en el campo de verificación. Salvó en uno de los casos, en general no se afecto el comportamiento normal del sistema.

Se tuvo un 100% de éxito la verificación de la integridad de los procesos. No se presentaron falsos positivos ni falsos negativos. El sistema se comportó tal y como se esperaba.

La red en que se realizaron las pruebas es pequeña y se espera hacer pruebas sobre redes más grandes y con usuarios corriendo aplicaciones comunes.

La seguridad computacional ha tomado especial importancia en estos últimos años debido a las vulnerabilidades propias de los sistemas y a los ataques cada vez mas sofisticados que explotan dichas vulnerabilidades. El tratar de imitar lo que hace el sistema inmunológico para proteger al cuerpo humano es un área importante que se ha estado estudiando. Hasta el momento se ha realizado la parte de modificación del núcleo quedando por realizar el protocolo de comunicación para que estos policías se comuniquen.

5. Referencias

[1] Dipankar Dasgupta, University of Memphis, pp. V-IX, 3-21h

[2] Forrest Stephanie, Hofmeyr Steven A., Somayaji Anil. *Computer Immunology*. October 1997/Vol 40 No. 10 Communications of the ACM

[3] Warrender Chistina, Forrest Stephanie, Pearlmutter Barak. *Detecting Intrusions using system calls*. Dept. of Computer Sciences. University of New Mexico

[4] Burgess Mark. *Computer Immune Systems*. Centre of Science and Technology Oslo College.

[5] Rusling David A. *The Linux Kernel*. pp 35-51

[6] Satllings, William. *Network and Internetwork Security*, Ed. Prentice Hall, E.U.A., 1995.

[7] http://www.iu.hioslo.no/~mark/research/immune.html

[8] http://www.iu.hioslo.no/~mark/research/AIdrift/AIdrift.html

[9] Digital Biology, Peter J. Bentley, Simon & Schuster; 1st edition, 2002

[10] The Complete Computer Virus Handbook, Price Waterhouse, Hyperion Books; 1989