DISEÑO DE UN ENTORNO CONFINADO PARA DIFERENTES PERFILES DE USUARIOS EN AMBIENTES EMPRESARIALES Y ACADÉMICOS UTILIZANDO SANDBOX Y VIRTUALIZACIÓN

LUIS CARLOS GÓMEZ DÍAZ
GISELLE GRAZT MEDINA

UNIVERSIDAD PONTIFICIA BOLIVARIANA
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

DISEÑO DE UN ENTORNO CONFINADO PARA DIFERENTES PERFILES DE USUARIOS EN AMBIENTES EMPRESARIALES Y ACADÉMICOS UTILIZANDO SANDBOX Y VIRTUALIZACIÓN

LUIS CARLOS GÓMEZ DÍAZ
ID. 204125
GISELLE GRAZT MEDINA
ID. 202643

Proyecto de grado para optar por el título de
Especialista en Seguridad Informática

DIRECTOR
MSc. URBANO ELIECER GÓMEZ PRADA

UNIVERSIDAD PONTIFICIA BOLIVARIANA
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

DEDICO ESTE PROYECTO

A DIOS por darme la oportunidad de continuar con mis estudios y darme la fuerza de continuar con mis objetivos, para así poder finalizar metas propuestas y seguir cosechando frutos del esfuerzo realizado.

A mi Padre y a mi Madre por apoyarme y alentarme a seguir adelante en este como en todos los proyectos que me propongo.

A mi abuelo Carlos Julio Díaz Pinto que en paz descanse, quién nos dejó muchos buenos recuerdos que se quedarán en nuestra memoria y enseñanzas como el valor del trabajo porque nos mostró con sus actos que se pueden lograr muchas cosas en la vida con trabajo, fortaleza, sensatez y actuando de una manera correcta.

A mis hermanos por ser una luz y una fuerza que me acompaña en todo momento.

A mi tía Isaura Gómez Arenas, por brindarme su respaldo y confianza en la realización de esta especialización.

Luis Carlos Gómez Díaz

A la esencia sutil del universo, al principio eterno del AMOR, a ti DIOS por inspirarme y darme todo este aprendizaje.

A mi Familia por brindarme su amor y apoyo.

Giselle Grazt Medina

AGRADECIMIENTOS

A mi compañera y amiga Giselle Grazt Medina por su apoyo y contribución en esta etapa de la vida.

A los profesores de la Especialización, por brindarnos sus conocimientos y experiencia durante el curso de la especialización.

Al nuestro director proyecto, el profesor M.Sc. Urbano Eliécer Gómez Prado, por brindarnos su conocimiento y guiarnos en este propósito.

A la Universidad Pontificia Bolivariana UPB por brindarnos este espacio de aprendizaje y por su apoyo en el desarrollo y crecimiento de nuestro perfil profesional.

Luis Carlos Gómez Díaz

Gracias Luis Carlos Gómez Díaz por tu esfuerzo y aporte a la realización de este proyecto.

Al director del proyecto, el profesor M.Sc. Urbano Eliécer Gómez Prado, por su colaboración y gentil comprensión.

A la Universidad Pontificia Bolivariana y a cada uno de los profesores de la Especialización por permitirnos desarrollar estas nuevas habilidades profesionales en el campo de la Seguridad Informática.

A mi hermana Johana G. Grazt Medina y a mi amiga Aliria Álvarez Delgado por creer en mí y proporcionarme el respaldo en este proyecto.

Giselle Grazt Medina

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

RESUMEN GENERAL DE TRABAJO DE GRADO

TITULO: DISEÑO DE UN ENTORNO CONFINADO PARA DIFERENTES PERFILES DE USUARIOS EN AMBIENTES EMPRESARIALES Y ACADÉMICOS UTILIZANDO SANDBOX Y VIRTUALIZACIÓN

AUTOR(ES): LUIS CARLOS GÓMEZ DÍAZ
GISELLE GRAZT MEDINA

FACULTAD: Esp. en Seguridad Informática

DIRECTOR(A): URBANO ELIECER GÓMEZ PRADA

RESUMEN

En este documento se muestra el diseño de un prototipo de Seguridad creado a partir de las necesidades identificadas en los ambientes de desarrollo de software empresariales y académicos. Para diseñar el prototipo se utiliza el concepto de defensa en profundidad o seguridad por niveles y también se utilizan diferentes herramientas disponibles en el mercado de tipo freeware. Primero se realizó investigación de las herramientas que podrían adaptarse al modelo que se planteó, y entre estas se analizó y se eligieron las que eran las mejores opciones y las que ofrecían mejores aportes de seguridad al prototipo. Entre las herramientas seleccionadas están Linux-Vserver y util-Vserver, que fueron utilizadas para aprovechar las características de aislamiento y administración de recursos hardware que ofrece la Virtualización, y también la herramienta SCPOnly que fue utilizada para obtener un segundo nivel de aislamiento, creando un espacio confinado en disco, al cual solo se permite acceso a las personas que lo requieran, y lo cual debería ser validado por el administrador del sistema. En las secciones del documento se muestra como se realizó la elección de herramientas, el proceso de diseño del prototipo, la Implementación del diseño planteado, las pruebas realizadas y el producto final. Por último se advierte que el prototipo creado en este proyecto puede ser optimizado, añadiéndole mejoras en los objetivos de seguridad aquí trabajados, y en los objetivos de seguridad que no se tuvieron en cuenta en el prototipo, para así obtener una maduración de este y hacer que sea más robusto.

PALABRAS CLAVES:

Prototipo, Seguridad, Diseño, Aislamiento, Confinamiento, Virtualización, Sandbox, Recursos, Hardware, Jaulas.

V° B° DIRECTOR DE TRABAJO DE GRADO

GENERAL SUMMARY OF WORK OF GRADE

TITLE: DESIGN OF A CONFINED ENVIRONMENT FOR DIFERENT USER ROLES IN BUSINESS AND ACADEMIC ENVIRONMENTS USING SANDBOXING AND VIRTUALIZATION

AUTHOR(S): LUIS CARLOS GÓMEZ DÍAZ
GISELLE GRAZT MEDINA

FACULTY: Esp. en Seguridad Informática

DIRECTOR: URBANO ELIECER GÓMEZ PRADA

ABSTRACT

In this paper the design of a security prototype that was created from the identified needs in business and academic software development environments is shown. The defense in depth or layered security concept is used for the prototype design and different freeware tools also were used. First, an investigation was conducted in search of tools that could be adapted to the proposed model, and among these were analyzed and were chosen which were the best options and offered best security contributions to the prototype. Among the selected tools are Linux-Vserver and Util-Vserver, that were used to take advantage of isolation and hardware resource management features offered by virtualization, and also the SCPOnly tool that was used to obtain a second isolation level, creating a confined disk space, in which access is permitted only to people to need it, and which should be validated by the system administrator. In the paper sections is shown how the tools choice is done, the prototype design process, the implementation of proposed prototype, the performed tests, and the final product. Finally, it´s advised that the prototype created in this project can be optimized, by adding improves to the security objectives worked here, and also to the security objectives which were not taken into consideration for the prototype, in order to obtain a maturation of this and make it more robust.

KEYWORDS:

Prototype, Security, Design, Isolation, Confinement, Virtualization, Sandbox, Resources, Hardware, Jails.

V° B° DIRECTOR OF GRADUATE WORK

1. INTRODUCCIÓN

En ambientes de producción empresariales soportados en las modernas *TICs*, se hace necesario tener un estricto control de todos los procesos que dependen del correcto funcionamiento y el mantenimiento en buen estado de estas tecnologías, con el fin de que igualmente los objetivos de negocio empresariales resulten de forma esperada, y evitar al máximo que de improviso se pueda perder el control de estos procesos, y estos se vean afectados hasta un punto donde pueda llegar a ser crítico y pueda estar por debajo del límite de aceptación de riesgo permitido por las políticas empresariales, lo cual podría tener efectos considerables o para casos extremos irreversibles en los objetivos de negocio.

Este proyecto hace énfasis en empresas donde en alguna de sus áreas tienen implementados procesos de desarrollo de software como parte de su campo de negocios, como también hace énfasis en el estudio e implementación de una solución mediante el uso de un sistema *Sandbox* que se adapte de la mejor manera posible al entorno en el cuál se implementa, seleccionada de acuerdo a las necesidades y requerimientos de estas organizaciones entre varias soluciones de este tipo que se encuentren disponibles.

El concepto de sistema *Sandbox* en la informática hace referencia a la protección de los recursos informáticos primordialmente los recursos críticos para el negocio, como la información, aplicaciones, elementos software y configuraciones vitales, todo esto mediante el aislamiento de los recursos que sean estrictamente necesarios para uso en procesos de alto riesgo, como lo es especialmente en este campo los procesos que corresponden a la etapa de pruebas de software, y también mediante la limitación de privilegios de usuario, de tal modo que se puedan realizar los procesos necesarios para el desarrollo de software correctamente en un ambiente confinado sin necesidad de exponer innecesariamente los recursos críticos, y buscando prevenir que se puedan presentar diferentes improvistos en el funcionamiento del software por causas diferentes, como inestabilidad del software, errores de programación o incluso virus incluidos dentro del software por actuaciones de mala fe de los desarrolladores.

Durante el documento se realiza la elección e implementación de las mejores soluciones posibles formando un sistema *Sandbox* dentro de un equipo perteneciente al área de desarrollo de software, donde se crea un ambiente confinado limitando todos los recursos y privilegios necesarios con el fin de brindar un nivel esperado de seguridad, que sea apropiado para este tipo de entorno crítico en la empresa. El diseño de este sistema de seguridad y sus implementaciones se realizan con las mejores configuraciones visibles en pro de llegar a un alto nivel de seguridad mediante el uso de esta solución sin descartar que puedan ser necesarias medidas de seguridad adicionales dependiendo de la criticidad de los recursos a proteger y el nivel de riesgo aceptable que se establezca.

Finalmente se ejecutan las pruebas necesarias para confirmar el buen funcionamiento de la solución, analizando los resultados, verificando el cumplimiento de los requisitos de seguridad que se habían establecido, para implementar posibles mejoras y por último sacar las conclusiones que dejo el proyecto en la creación de la *Sandbox*.

2. ESPECIFICACIÓN DE LA SITUACIÓN PROBLEMÁTICA

Comúnmente los ambientes de desarrollo de software por su naturaleza, se convierten en un entorno vulnerable, en donde cualquier tipo de implementación que se realice puede conllevar a que su infraestructura se desestabilice o presente algún problema de seguridad.

Las empresas que cuentan con un área destinada para el desarrollo de software, ya sea con fines de uso de la misma empresa o para su comercialización, utilizan diferentes tecnologías e implementan diferentes tipos de controles que permitan cumplir con sus expectativas en los niveles de seguridad deseados. Los entornos en donde se requieren mejores controles, es en donde los riesgos presentados tienen que ver y pueden afectar los procesos del negocio, ya que en estos casos una amenaza materializada en un entorno puede impedir que se logren los objetivos de negocio y tener un gran impacto en las finanzas de la organización. De no tener implementados los controles adecuados que protejan este entorno de desarrollo contra las amenazas existentes se pueden ver afectados los principios fundamentales de seguridad para los sistemas de información: confidencialidad, integridad y disponibilidad, por lo cual se deberían reforzar los controles por ejemplo utilizando diferentes herramientas para implementar los principios básicos de la seguridad: Principio de defensa en profundidad, principio del menor privilegio, etc.

Por ejemplo, se necesita realizar pruebas a un software que se está desarrollando, pero se sabe que este software puede ser inestable y puede causar daños, algunos de ellos graves, al sistema donde se está ejecutando, como eliminar archivos del sistema operativo, modificar información crítica de la organización o impedir la disponibilidad de servicios prestados a causa de un uso inadecuado de los recursos del sistema, todo esto en el equipo que se ejecuta el software o en cualquier equipo accesible desde la red, entonces, si no se tiene una implementación de controles adecuados para garantizar un buen nivel de seguridad en el sistema, se facilita la materialización de los riesgos existentes.

Hay que tener en cuenta que las amenazas no siempre existen por errores de los trabajadores de la organización o por causas involuntarias, sino que también puede ser provocados por agentes internos o externos de manera premeditada, como lo puede ser cualquier software al que le sea incluido algún tipo de malware que afecte el sistema, software que pueda crear puertas traseras en el sistema y que puedan ser vulneradas por la red, etc.

Una excelente opción a tomar para mitigar este tipo de riesgos es la implementación de un entorno controlado dentro del ambiente en el que se realicen las pruebas del software o también en cualquier ambiente de desarrollo de software, en donde se limiten los recursos del sistema a los que pueda acceder el software y se puedan limitar privilegios de usuario en el sistema, y así impedir en gran medida que este software inestable o desconfiable pueda causar daños al sistema por fuera de los límites establecidos por el administrador del sistema. Esto se puede hacer mediante la implementación de una *Sandbox* en un ambiente virtualizado dentro de los equipos destinados para pruebas de software empresarial, lo cual es tema de este documento.

3. JUSTIFICACIÓN

El aislamiento de procesos, el control de los recursos y el control de privilegios se hacen indispensables en el contexto de la seguridad informática, a la hora de compartir estos mismos recursos, no sólo por buscar un mejor rendimiento sino por establecer un orden de acuerdo a las necesidades del rol que tiene cada usuario para evitar en lo posible daños al sistema operativo anfitrión, modificaciones no autorizadas de archivos o que indirectamente se afecten otros sistemas en la red.

Estas características hay que tenerlas en cuenta especialmente en ambientes de desarrollo de software, en donde se crea y se requiere probar e implementar el software con seguridad y preferiblemente de manera aislada.

Las *Sandbox* ofrecen una solución, permitiendo implementar un ambiente confinado en donde se pueden asignar recursos a los desarrolladores y mantener el control limitando la funcionalidad y alcances de los usuarios en el sistema.

4. OBJETIVOS

4.1 Objetivo general

- ✓ Diseñar e implementar un sistema de *Sandbo*x para aislar procesos y asignar recursos en el sistema operativo para usuarios con diferentes roles en ambientes empresariales y académicos en casos de prueba específicos con el fin de aprovechar mejor los recursos, evitar acciones no autorizadas y efectos inesperados de software no confiable.

4.2 Objetivos específicos

- ✓ Clasificar algunas de las aplicaciones con característica *Sandbox* más conocidas en el mercado de acuerdo a su funcionalidad en el sistema operativo.

- ✓ Diseñar una solución *Sandbox* que controle la funcionalidad y los recursos para usuarios según su rol preestablecido en ambientes GNU/Linux o Windows.

- ✓ Realizar un prototipo y pruebas de funcionamiento de la *Sandbox* utilizando diferentes perfiles de usuarios en los casos de prueba específico.

5. MARCO TEÓRICO

5.1 Virtualización

5.1.1 Imitación de sistemas[1]

La imitación de un sistema puede abarcar los términos Simulación y Emulación. Al parecer el término más ampliamente conocido en la comunidad de investigadores es el de simulación, por lo que en primer lugar se describe la diferencia entre simulación y emulación. Haciendo una abstracción extrema, se puede decir que un simulador imita la micro-arquitectura del computador, y si la imitación es suficientemente precisa, el comportamiento se preservará como consecuencia. Un emulador, en cambio, imita el comportamiento en primer lugar y la micro-arquitectura puede ser imitada así como se van requiriendo más detalles en la emulación.

Los Emuladores son más apropiados cuando el comportamiento es importante como un todo, desde el punto de vista de entrada/salida, en donde no importa que organización interna y que estructura tenga el sistema.

5.1.2 ¿Qué es la virtualización?

La virtualización en informática se refiere a la simulación de un recurso tecnológico (hardware y/o software) sobre la cual otro software se ejecuta. El resultado de la simulación es comúnmente llamado "máquina virtual".[2]

Existen dos tipos principales de virtualización: virtualización a nivel de hardware y virtualización a nivel de software; los cuales se subdividen en diferentes clases.

5.1.3 Virtualización a nivel de hardware[3]

La tecnología de virtualización a nivel de hardware hace referencia a la simulación completa o parcial de un hardware con el objetivo de ejecutar un sistema operativo invitado sin modificar sobre éste.

5.1.3.1 Tipos de virtualización a nivel de hardware

A continuación se describen los diferentes tipos de virtualización a nivel de Hardware:

- ✓ Virtualización completa

---

[1] **JAKUBCO, Peter.** y **SLAVOMÍR, Simonák.** Utilizing GPGPU in Computer Emulation: Introduction. En : Journal of Information and Organizational Sciences: JIOS. VOL 36, NO. 1 (2012); p 39-53. ISSN 1846-9418 (online) - ISSN 1846-3312 (print).

[2] **SCARFONE, Karen. SOUPPAYA, Murugiah.** y **HOFFMAN, Paul.** Guide to security for full virtualization technologies: Recommendations of the National Institute of Standars and Technology. En : National Institute of Standards and Technology NIST. Special Publication 800-125 (Ene, 2011). p. ES-1.

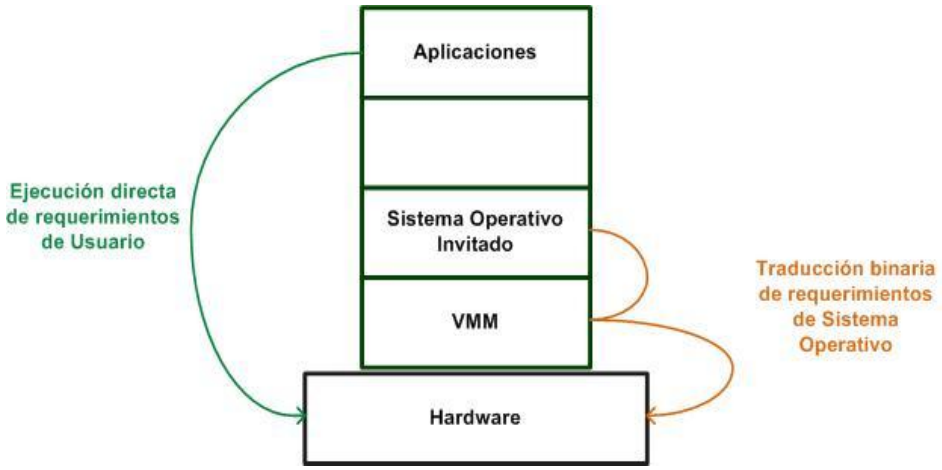
[3] **YU, Yang.** OS-level Virtualization and Its Applications. Dissertation Presented to The Graduate School in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science. Stony Brook University U.S.A. 2007. P 8-9.

La Simulación completa del sistema emula toda la arquitectura de conjunto de instrucciones de una plataforma a otra, traduciendo las instrucciones del sistema operativo invitado al conjunto de instrucciones del sistema anfitrión para así después de esto ejecutarlas directamente en el hardware subyacente. Debido a la emulación completa de la arquitectura, y a que cada instrucción del sistema invitado (la máquina virtual) debe ser interpretada y traducida, este tipo de virtualización conlleva un considerable declive en el rendimiento del sistema. Ejemplos de este tipo de virtualización son *Bochs* (Sistema que emula procesadores x86 sobre varias de las plataformas más populares) y la versión de Virtual PC para algunas plataformas MAC OS que eran implementadas sobre arquitectura RISC.

- ✓ Virtualización Nativa

La virtualización nativa simula un ambiente de hardware completo para soportar un sistema operativo invitado sin modificar, y para el mismo tipo de CPU. Este tipo de virtualización es una combinación entre traducción binaria (para las instrucciones más privilegiadas que no pueden ser ejecutadas desde el SO invitado directamente en el hardware) y ejecución directa (usualmente la mayoría de las instrucciones de menor privilegio, las cuales pueden ser enviadas desde la máquina virtual hacia el hardware sin modificación para ser ejecutadas)[4]. Durante el proceso, el monitor de máquina virtual (*hypervisor*) intercepta las instrucciones, las clasifica y realiza los procesos de traducción y envío, o reenvío directo, de las instrucciones hacia el hardware. Debido a que se evita el traducir determinadas instrucciones, se reduce la sobrecarga de CPU, con lo cual se mejora el rendimiento de las máquinas virtuales.
Con este tipo de virtualización se realiza una abstracción completa del sistema operativo invitado (completamente desacoplado) de la capa subyacente de hardware por medio del hipervisor (capa de virtualización), y así el sistema operativo invitado no sabe que está siendo virtualizado y no tiene que ser modificado. Ni la virtualización nativa ni la virtualización completa necesitan asistencia del sistema operativo o del hardware para virtualizar las instrucciones privilegiadas ya que el hipervisor (emulador) traduce estas instrucciones.
A continuación en la Figura 1. Virtualización Nativa se describe la comunicación entre las diferentes capas vinculadas a la virtualización nativa y su ubicación en los niveles de privilegios para acceso del hardware en donde las aplicaciones tienen nivel 3 y el VMM (*Virtual Machine Monitor*) se ubica en el nivel 0.

Figura 1. Virtualización Nativa



[4] **Vmware.** Understanding Full Virtualization, Paravirtualization, and Hardware Assist. [En línea] [Citado el: 21 de Julio de 2015.] http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

Fuente. Autores

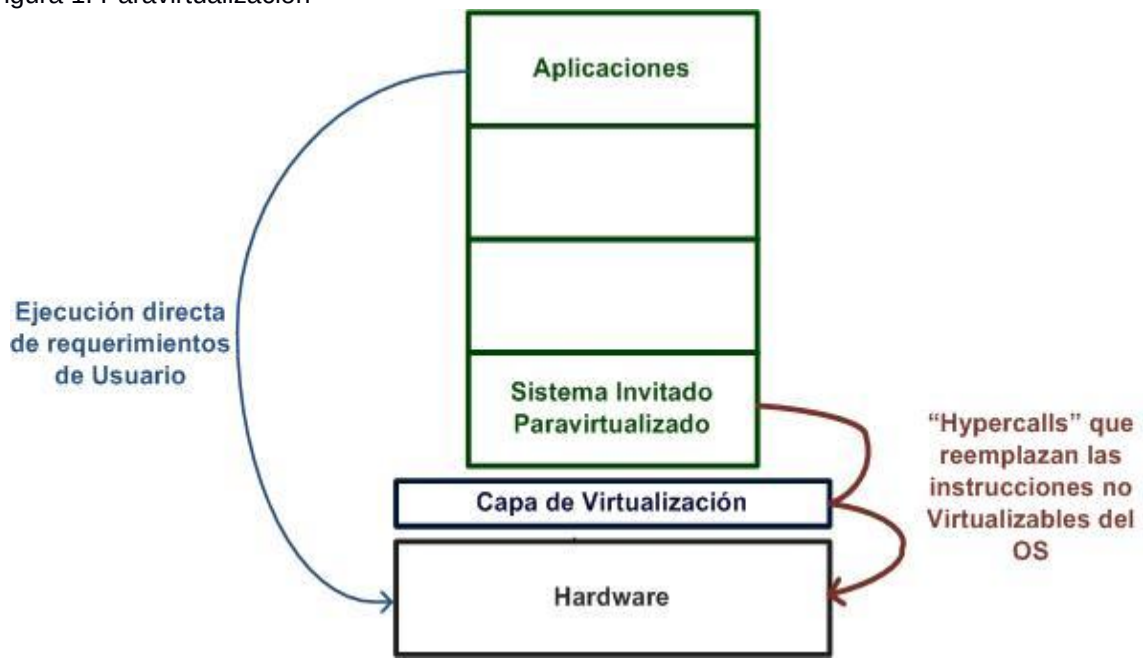
- ✓ Paravirtualización

Con la tecnología de paravirtualización se añaden instrucciones especiales en el núcleo "*kernel*" del sistema operativo invitado llamadas "*hypercalls*" con el fin de reemplazarlas por las instrucciones que no pueden comunicarse directamente sin modificar desde el *kernel* del sistema invitado hacia el hipervisor "instrucciones no virtualizables". El hipervisor también provee otras interfaces "*hypercall interfaces*" para manejar otras instrucciones críticas del kernel como manejo de memoria, administración de interrupciones y mantenimiento del tiempo "*time o time keeping*". Un ejemplo de paravirtualización es el proyecto de código abierto o "*open source"* XEN, el cual virtualiza procesamiento y memoria utilizando un kernel Linux modificado y virtualiza las operaciones E/S utilizando drivers de dispositivos personalizados en el sistema operativo invitado.

Otro ejemplo de paravirtualización es la implementación del *ESX server*, el cual es un sistema operativo ligero y modificado, que está basado en un núcleo *Linux Red Hat Enterprise server*, y tiene embebidos los componentes para la ejecución del *hipervisor* y componentes de virtualización de VMware además de otros servicios y herramientas de gestión autónomos e independientes.

La virtualización completa, la virtualización nativa y la paravirtualización (Figura 1. Paravirtualización) ubican la capa de virtualización en un nivel cercano al hardware. La ventaja de estas técnicas es el gran aislamiento que genera entre las diferentes máquinas virtuales y el sistema anfitrión.

Figura 1. Paravirtualización



Fuente. Autores

5.1.4 Virtualización a nivel de Sistema Operativo[5]

La forma más intrusiva de virtualización es a nivel de sistema Operativo. Por el contrario de la virtualización completa, la virtualización nativa, y la paravirtualización, la virtualización a nivel de sistema operativo no necesita un hipervisor. En lugar de esto, el sistema Operativo es modificado

[5] **YU.** Op. cit., p. 9-15.

para aislar de forma segura múltiples instancias de un sistema operativo dentro de una misma máquina anfitriona. Las instancias de sistemas operativos invitados son comúnmente referidas como VPS (*Virtual Private Servers*). La ventaja de este tipo de virtualización es que tiene un rendimiento óptimo, ya que contrario a los otros tipos de virtualización que consumen bastantes recursos al tener que soportar múltiples *kernels* por separado ya que no se tiene que hacer intercepción o traducción de instrucciones por parte del hipervisor, típicamente el rendimiento se acerca a las velocidades de sistemas no virtualizados. La desventaja principal de este tipo de virtualización es el tener un *kernel* compartido, por lo que si el *kernel* falla o es comprometido, todas las instancias de sistema operativo van a ser comprometidas de igual forma.

Existen diferentes formas de realizar virtualización a nivel de sistema operativo: virtualización de servidor, virtualización de aplicación, versionado de sistemas de archivos y capas de compatibilidad. Estos mecanismos desprenden o crean ambientes virtuales en la capa superior de un *kernel* simple de sistema operativo.

La capa de virtualización por sistema operativo puede ser a nivel de las librerías de las interfaces de aplicación del sistema, de las interfaces de llamadas al sistema o de una pila de controladores del sistema de archivos.

A continuación se describen en detalle los diferentes tipos de virtualización a nivel de sistema operativo:

- ✓ Virtualización de Servidor

La virtualización de servidor hace referencia a la creación de ambientes operativos virtuales completos, incluyendo cuentas de usuarios, herramientas de administración, etc.

Un aspecto importante en mira de confinar múltiples instancias de un sistema operativo, es crear un espacio de sistema de archivos separado para cada instancia. La llamada del sistema *chroot()* característica de los sistemas UNIX configura el directorio raíz para los procesos llamados y sus subprocesos. Este conjunto de procesos por lo tanto establece su propio espacio de sistema de archivos y no pueden percibir archivos por fuera de la nueva raíz. Muchos de los sistemas de virtualización a nivel de sistema operativo para sistemas basados en UNIX utilizan esta llamada para virtualizar su espacio de sistemas de archivos.

- ✓ Virtualización de aplicación

La virtualización de aplicación hace referencia a una aplicación en particular o a un subconjunto de la virtualización completa, tal y como lo son los programas *Sandboxing* para aplicaciones no confiables. Las *Sandbox* se usan para crear ambientes virtuales aislados sobre un sistema operativo anfitrión, confinando la ejecución de los programas de aplicación seleccionados. Estos ambientes virtuales son exclusivamente usados para cuatro propósitos: resolver conflictos de aplicaciones, migrar procesos en ejecución, aislar programas no confiables, y soportar ambientes de aplicaciones portables.

Por ejemplo, para resolver conflictos entre aplicaciones, una aplicación no es físicamente instalada en el computador; En lugar de esto, su ambiente de ejecución es empaquetado dentro de una máquina virtualizada a nivel de sistema operativo.

- ✓ Versionado de sistemas de archivos

El versionado de sistema de archivos permite a múltiples versiones de archivos coexistir y además provee flexibilidad para deshacer cambios al sistema de archivos. Los sistemas de recuperación de intrusiones utilizan sistemas de archivos con autenticación extensiva y versionado para reparar automáticamente un servidor de archivos en red comprometido. Los servidores de archivos transaccionales agrupan actualizaciones de archivos dentro de una transacción y esconde todas las actualizaciones de aplicaciones fuera de la transacción hasta que esta es cumplida.

- ✓ Capa de compatibilidad

Muchas técnicas intentan crear una capa de compatibilidad a nivel de sistema operativo para emular la ejecución de archivos binarios de otros sistemas operativos. Ejemplos de tales proyectos es la aplicación *Wine*, la cual emula aplicaciones Windows en sistemas basados en *Unix*, o la herramienta Interix (*Microsoft Windows services for UNIX*), que emula un ambiente *Unix* en un *kernel* de Windows NT. Estas técnicas implementan todas las librerías del sistema que traducen las llamadas del sistema de la aplicación emulada dentro de llamadas al sistema soportadas por el sistema operativo anfitrión, y mapea los espacios de nombres de la aplicación emulada a espacios de nombres completamente diferentes del sistema host.

5.2 Definición de Sandbox

El término en inglés *Sandbox* se traduce al español como caja de arena[6], *sand* (arena), *box* (caja). En el libro *Computer Security Art and Science*)[7] define una *Sandbox* como un ambiente en el cual se restringen las acciones de los procesos de acuerdo a una política de seguridad, y en esta misma fuente se define modelo de seguridad como un modelo que representa una política o un conjunto de políticas. Según esto una *Sandbox* puede considerarse como un modelo de seguridad.

En el mismo libro *Computer Security Art and Science* [8] se hace la comparación de una caja de arena de patio de recreo o en inglés "*playground Sandbox*" y una caja de arena o *Sandbox* en computación. Señala que una caja de arena de patio de recreo provee un ambiente de seguridad a un niño que juega en ella. Si el niño abandona la caja de arena sin supervisión, podría hacer cosas que se supone no debería hacer. El concepto de caja de arena o *Sandbox* en la computación es similar. En el contexto de la computación, *Sandbox* provee un ambiente de seguridad para los programas que se ejecutan dentro de ésta. Si los programas "abandonan" la *Sandbox*, podrían hacer cosas que se supone no deberían o no están autorizados por las políticas de uso del administrador. Ambos tipos de caja de arena restringen las acciones de sus ocupantes.

Así mismo, en *Botnets - The killer web applications*, Shiller, C. et al, (2007)[9], se definen las *Sandbox* como un concepto común en la seguridad de computadores, e indica que son usadas para ejecutar código que proviene de fuentes no verificadas o desconfiables.

---

[6] **Wikipedia.** *Sandbox*. [En línea] [Citado el: 28 de Mayo de 2015.] http://es.wikipedia.org/wiki/Sandbox.

[7] **BISHOP, Matt.** Computer Security: Art and Science. U.S.A., Addison-Wesley Professional, 2002. ISBN: 0201440997. p. 449.

[8] Ibid. p. 449

[9] **SCHILLER, Craig A. BINKLEY, Jim. HARLEY, David. EVRON, Gadi. BRADLEY, Tony. WILLEMS, Carsten.** y **CROSS, Michael.** Botnets: The Killer Web App. U.S.A., Syngress Publishing, Inc., 2007. ISBN-10: 1-59749-135-7. p. 346

5.3 Aplicaciones de una Sandbox

Existen diferentes formas posibles de realizar limitaciones a las acciones o alcances que tienen los procesos generados por un programa o por un usuario en particular. Por ejemplo, en el documento *user-level resource-constrained Sandboxing*[10] se describe el diseño de una *Sandbox* utilizando una estrategia para imponer restricciones cuantitativas sobre tres recursos representativos: CPU, memoria y red sobre Windows NT. Este mismo diseño de una *Sandbox* igualmente puede ser proyectado a distintos sistemas operativos, tal y como se pretende en este proyecto.

Otro ejemplo de *Sandbox* es native client. Native Client tiene el objetivo de dar a las aplicaciones basadas en navegadores web el rendimiento computacional de aplicaciones nativas sin comprometer la seguridad. Native client manifiesta las limitaciones de la *Sandbox* directamente en la máquina nativa en lugar de usar una máquina virtual u otra representación de arquitectura de set de instrucciones portable[11].
Entre los beneficios[12] que permite la *Sandbox* están:

- Navegación Web segura: Ejecutar el navegador Web bajo la protección de una *Sandbox* significa que todos los programas maliciosos descargados por el navegador están confinados dentro de estas y se pueden eliminar de forma trivial sin impactar negativamente el sistema operativo anfitrión.
- Aumento de Privacidad: El historial de navegación, las cookies, el caché y los archivos temporales recogidos durante la navegación Web permanecen en el recinto confinado por la *Sandbox*.
- Correo electrónico seguro: Los virus y otros programas maliciosos que pueden estar escondidos en el correo electrónico no pueden salir de la *Sandbox* y no pueden infectar el sistema anfitrión.
- Estabilidad del sistema anfitrión: Previene posibles fallas en el sistema anfitrión debido a la instalación de software poco confiable o de prueba, situación que permite estudiar el comportamiento de dicho software de prueba.

5.4 Ambientes de desarrollo de software

El desarrollo de software es la construcción de sistema o una aplicación software que pueden ser implementados en computadores para ejecutar tareas repetitivas; en otras palabras un software que automatiza tareas manuales. A menudo el desarrollo de software es relacionado con la programación, o escribir un código fuente mediante el uso de un determinado lenguaje de programación, pero también puede incluir todas las demás tareas, desde una idea inicial hasta la implementación de esta idea. Implementación en este caso significa la especificación de todos los

[10] **USENIX WINDOWS SYSTEMS SYMPOSIUM.** (Agosto, 2000: New York, U.S.A.). User-level Resource-constrained Sandboxing. New York: New York University, 2000. 11 p.

[11] **IEEE SYMPOSIUM ON SECURITY AND PRIVACY.** (2009: U.S.A.). Native Cient A Sandbox for portable Untrusted X86 Native Code. U.S.A., 2009. 15 p.

[12] **Sandboxie.** Sandboxie trust no program. [En línea] [Citado el: 28 de Mayo de 2015.] http://www.Sandboxie.com.

detalles o algoritmos, diseño, programación, prueba, implementación y mantenimiento[13]. Según la descripción anterior, se puede entender un ambiente de desarrollo de software como un entorno computacional apto para realizar un desarrollo de software.

Estos entornos deben cumplir determinados requerimientos dependiendo del usuario o los usuarios que van a trabajar en él, las necesidades que estos tengan en cuanto a recursos; hardware, memoria, CPU, espacio en disco, infraestructura de red, etc., y necesidades de aplicaciones como editores, compiladores, depuradores, herramientas de análisis, software de control de versiones, etc. Estos ambientes deben ser controlados ya que cualquier tipo de software que se ejecute puede representar una amenaza para el sistema al que pertenecen y también para los sistemas a los que este pueda acceder en la red.

Para controlar estos entornos de manera que permanezcan completamente funcionales y tengan una seguridad robusta existen diferentes métodos que se pueden implementar, entre ellos están el control de privilegios a los usuarios de desarrollo, configuración de herramientas utilizadas, restricción de cambios en las configuraciones y la implementación de mecanismos de seguridad para control de recursos entre otros. Por ejemplo en "*The Protection of Information in Computer Systems*"[14], se describen 8 ejemplos de principios de diseño en el desarrollo del software: "*Economy of mechanism*", "*Separation of privilege*" y "*Least privilege*" entre otros, que pueden guiar el diseño y contribuyen a una implementación sin fallas de seguridad. Varios de estos aspectos se pueden controlar mediante el empleo de una *Sandbox*, que es el objetivo de este proyecto.

[13] **BULAJIC, Aleksandar. SAMBASIVAM, Samuel.** y **STOJIC, Radoslav.** An Effective Development Environment Setup for System and Application Software. En: Issues in Informing Science and Information Technology (IISIT), 2013 Volume 10. Online ISSN: 1547-5867, Print ISSN: 1547-5840.

[14] **AMC SYMPOSIUM ON OPERATING SYSTEM PRINCIPLES.** (Octubre, 1973: Charlottesville, U.S.A.). The Protection of Information in Computer Systems. Charlottesville: Virginia University, 1973. 30 p.

6. CLASIFICACIÓN DE LAS APLICACIONES CON CARACTERÍSTICA SANDBOX MÁS CONOCIDAS EN EL MERCADO DE ACUERDO A SU FUNCIONALIDAD EN EL SISTEMA OPERATIVO

Para realizar la clasificación de las herramientas más adecuadas a las necesidades del proyecto primero se realizó un estudio del estado del arte de las herramientas existentes para cada sistema operativo que permitan crear un ambiente confinado para la creación de la *Sandbox*. Primero se decidió utilizar una herramienta de virtualización para crear la primera "capa" de aislamiento, para lo cual se va a utilizar la virtualización de sistema operativo por sus ventajas en cuanto a rendimiento se refiere, debido a que la virtualización se desprende de un único *kernel* compartido para crear las instancias de máquinas virtuales y por esto tiene alto consumo de recursos adicionales.

En la consulta y estudio del arte de las herramientas con las cuales podemos configurar una Sandbox por medio de la virtualización del sistema operativo se encontró un resumen de algunas de ellas, consolidadas en la **¡Error! No se encuentra el origen de la referencia.**2.

Adicional a estas herramientas se consultaron otras más y se consolidaron en la Tabla 1. clasificadas según el sistema operativo en el cual se pueden implementar.

Tabla 1 Herramientas de virtualizar por sistema operativo.

GNU LINUX	WINDOWS	MAC	SOLARIS/OTROS
*PARALLELS VIRTUOZZO CONTAINERS	*PARALLELS VIRTUOZZO CONTAINERS	*PARALLELS VIRTUOZZO CONTAINERS	*PARALLELS VIRTUOZZO CONTAINERS
PARALLES BARE METAL	PARALLES BARE METAL	VIRTUALBOX	SOLARIS CONTAINERS (SOLARIS – ZONES)
VIRTUALBOX	VIRTUALBOX		FREEBSD JAIL (FreeBSD)
lmctfy	iCore Virtual Accounts		Sysjail (OpenBSD, NetBSD)
LXC	*SANDBOXIE		*WPARs (AIX)
LXD	*SPOON		*HP-UX Containers (SRP) (HPUX)
CHROOT	*VMWARE WORKSTATION		
LINUX –VSERVER (Security context)	VMWARE PLAYER		
OPENVZ	*WINDOWS SEREVER 2008 R2 HYPER-V		
DOCKER	EASYVMX		
XEN	VIRTUAL PC		

Fuente: Autores

* Software privativo

Como primer criterio de selección para escoger la herramienta para configurar el prototipo de Sandbox fue que estas herramientas que se pudieran implementar en el sistema operativo GNU/Linux y que no fueran software privativo.

Luego se consultaron sus respectivas características y funcionalidades de manera, que estas cumplieran con los requerimientos del prototipo de Sandbox que se deseaba generar.

Tabla 2. Herramientas de virtualización de Sistema Operativo y sus características principales.

Mechanism	Operating system	License	Available since/between	Features							
				File system isolation	Copy on Write	Disk quotas	I/O rate limiting	Memory limits	CPU quotas	Network isolation	Partition checkpointing and live migration
chroot	most UNIX-likeoperating systems	Proprietary	1982	Partial	No	No	No	No	No	No	No
iCore Virtual Accounts	Windows XP	Proprietary/ Freeware	2008	Yes	No	Yes	No	No	No	No	No
Linux-Vserver (security context)	Linux	GNU GPL v.2	2001	Yes	Yes	Yes	Yes	Yes	Yes	Partial	No
LXC	Linux	GNU GPL v.2	2008	Partial	Partial. Yes with Btrfs.	Partial. Yes withLVM or Disk quota.	Yes	Yes	Yes	Yes	No
OpenVZ	Linux	GNU GPL v.2	2005	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Parallels Virtuozzo Containers	Linux, Windows,Mac OS X	Proprietary	2009	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Container/Zone	Solaris andOpenSolaris	CDDL	2005	Yes	Partial. Yes with ZFS	Yes	Partial. Yes with Illumos	Yes	Yes	Yes	No
FreeBSD Jail	FreeBSD	BSD	1998	Yes	Yes (ZFS)	Yes	No	Yes	Partial	Yes	No
sysjail	OpenBSD, NetBSD	BSD	- no longer supported as of 03-03-2009	Yes	No	No	No	No	No	Yes	No
WPARs	AIX	Proprietary	2007	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
HP-UX Containers (SRP)	HPUX	Proprietary	2007	Yes	No	Partial. Yes with logical volumes	Yes	Yes	Yes	Yes	Yes
Sandboxie	Windows	Shareware	2004	Yes	Yes	No	N/A	N/A	N/A	N/A	N/A

Fuente**:** Wikipedia [15]

[15] **Wikipedia.** Operating System Level Virtualization. [Citado el: 21 de Julio de 2015.] http://en.wikipedia.org/wiki/Operating_system%E2%80%93level_virtualization

6.1 Linux VServer

La herramienta escogida fue Linux – Vserver, ya que esta es muy robusta por su funcionalidad y permite realizar la configuración de la Sandbox deseada, aunque conceptualmente es similar a Solaris Containers, con su tecnología de aislamiento de Solares Zones, o FreeBSD Jail o incluso OpenVZ.

Linux Vserver, es una herramienta de virtualización que permite aislamiento a nivel del *kernel*, lo que permite crear un ambiente suficientemente aislado para garantizar un buen nivel de seguridad a la vez que se utilizan los recursos de manera eficiente ya que corre sobre el mismo *kernel*.

Este modelo de servidor privado virtual en particular se implementa a través de una combinación de:

- ✓ Contextos de seguridad "*security contexts*", es decir, cada partición o directorio confinado tiene un contexto de seguridad, y el sistema virtualizado dentro de este es el servidor privado virtual. Para acceder a estos contextos se utiliza chroot.

- ✓ Enrutamiento segmentado "*segmented routing*",

- ✓ *chroot*, cuotas extendidas "*extended quotas*" y otras herramientas estándar.

Se decidió utilizar la herramienta *Linux Vserver* por su naturaleza de Software libre, licencia GPL, porque ofrece varias características de control de recursos y aislamiento útiles para el proyecto, donde se busca que en los servidores privados virtuales se alojen sitios web, segregar cuentas de clientes, agrupar recursos eficientemente y restringir o evitar cualquier brecha de seguridad potencial, por medio de un mecanismo de jaula en donde se pueden usar de manera segura el sistema de archivos, tiempo de la CPU, direcciones de red y memoria, debido a esto los procesos ejecutados no pueden realizar un ataque de denegación de servicio sobre algo que este fuera del acceso de su partición o directorio confinado.

Además para su selección, también se tuvo encuenta que posee una completa documentación en internet.

A parte de esto el sistema operativo del servidor anfitrión es Debian, una distribución de GNU/Linux compatible completamente con la herramienta *Linux Vserver*, debido a que es un sistema robusto, también de licencia software libre y con una comunidad de desarrollo que se extiende a nivel mundial, la cual permite y proporciona un excelente soporte. Además de lo anterior, existe bastante documentación de la herramienta *Linux VServer* sobre la plataforma *Debian*.

Así mismo, se decidió utilizar otras herramientas como *Util-Vserver*, *Open-MPI* y *Scponly* para complementar el prototipo, para facilitar su configuración y administración, para realizar pruebas de funcionamiento, y para brindar un mayor nivel de seguridad como se describe en los siguientes capítulos del presente documento.

6.2 Ventajas de Linux- VServer

- ✓ No tienen ningún consumo por emulación, por tanto, los servidores virtuales son muy rápidos y livianos, comparten la misma interfaz de llamada del sistema.
- ✓ Respecto a la eficiencia en cuanto memoria y E/S, es más eficiente que la emulación de un sistema completo, ya que los procesos dentro del servidor virtual se ejecutan como procesos regulares en el sistema anfitrión, y en una emulación completa, no se puede entregar memoria sin uso o compartir un cache de disco con el servidor anfitrión.
- ✓ Se comparte un sistema de archivos común entre los servidores virtuales.

6 .3 Desventajas de Linux- VServer

- ✓ Requiere modificar el kernel del servidor anfitrión a manera de parche.
- ✓ El límite de velocidad de E/S no se puede configurar por servidor virtual.
- ✓ Los servidores virtuales están expuestos a los mismos bugs y potenciales agujeros de seguridad, ya que comparten el mismo núcleo o kernel del sistema operativo.
- ✓ Están sin virtualizar algunas llamadas al sistema, como son aquellas que están relacionadas con el reloj de tiempo real y las partes de los sistemas de archivos /proc y /sys. Esto impide que algunas distribuciones de GNU/Linux, entre ellas, Gentoo, no inicien apropiadamente dentro de un vserver sin modificaciones especiales.
- ✓ No tiene por el momento soporte para IPv6
- ✓ El núcleo del sistema operativo anfitrión y como tal el computador que funciona como servidor físico anfitrión son un único punto de falla para todos los servidores virtuales, como ocurre con las herramientas de Xen y UML, pues Linux VServer no tiene funcionalidad como cluster o migración de procesos.
- ✓ La red está parcialmente virtualizada por el momento en las actuales versiones del software, y los servidores virtuales están relacionados con alias asignados de la misma interfaz de red, esto no permite que cada servidor virtual forme su enrutamiento interno o configuración de cortafuegos particular.

7. DISEÑO DE UNA SOLUCIÓN SANDBOX QUE CONTROLE LA FUNCIONALIDAD Y LOS RECURSOS PARA USUARIOS SEGÚN SU ROL PREESTABLECIDO EN AMBIENTES GNU/LINUX O WINDOWS

7.1 Especificación general del diseño

Para el diseño de la solución *Sandbox* se tuvo en cuenta principios de seguridad que cubren necesidades comúnmente presentadas en las áreas de desarrollo de software:

1. El principio de defensa en profundidad en búsqueda de lograr el mayor punto de aislamiento y virtualización, implantando una estrategia con múltiples líneas de defensa. Ver 7.2.1 Principio de defensa en profundidad.
2. El principio de segregación de funciones en búsqueda de dividir responsabilidades de cada usuario de acuerdo a su campo de acción en el sistema. Ver 7.2.2 Segregación de funciones.
3. El principio del menor privilegio en búsqueda de asignar los privilegios estrictamente necesarios a los usuarios del sistema que le permita ejercer sus funciones, y denegar el resto de privilegios innecesarios. Ver 7.2.3 Principio del menor privilegio.
4. El principio de seguridad en caso de fallo en búsqueda de tener un plan de continuidad para el sistema en caso éste sea vulnerado o presente alguna falla, e igualmente en búsqueda de que esta falla no afecte al sistema por fuera de los límites de confinamiento realizados. Ver 7.2.4 Principio de Seguridad en caso de fallo.

Con el uso de estos principios se procura mejorar el control sobre el ámbito de desarrollo y disminuir los riesgos propios de este campo como inclusión de *malware*, intrusiones malintencionadas, software inestable, etc., sin tener pérdidas en la funcionalidad de la aplicación y el ambiente de trabajo.

Entre los objetivos claves de la solución están la gestión de recursos, el control de acceso y la seguridad por capas.

7.1.1 Esquema general propuesto

El prototipo diseñado se expone en forma de capas representando la posición relativa de los marcos de confinamiento proporcionados, y dentro de las cuales se ubican las diferentes herramientas y elementos utilizados en el entorno confinado *(la* Figura 2. Prototipo del entorno confinado

presenta el prototipo propuesto). A continuación se describen los marcos de confinamiento establecidos:

- Un Servidor físico que actúa como *host* del servidor virtual y que tiene determinados recursos hardware.
- Dentro del *host* se encuentra el servidor virtual, el cual tiene ciertas restricciones de recursos hardware que proporcionan límites en su uso, y donde se encuentran las aplicaciones y herramientas necesarias para el desarrollo de software.
- Dentro del espacio de archivos del Servidor virtual se encuentra la *Sandbox*, que es utilizada para cargar las aplicaciones web desarrolladas de manera confinada.

Figura 2. Prototipo del entorno confinado

Fuente. Autores

7.2 Principios de seguridad que se tuvieron en cuenta para el diseño

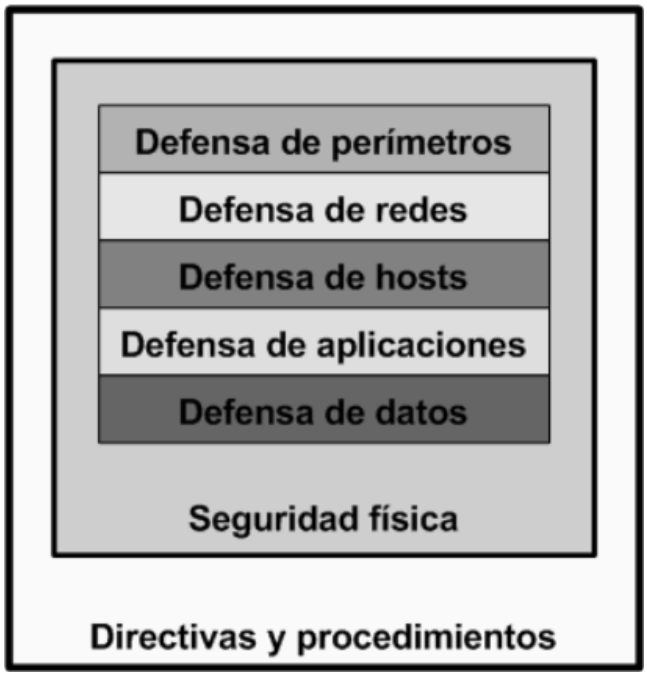
Algunas de los principios de seguridad más utilizados que se tuvieron en cuenta para el diseño de la solución, y fueron utilizados buscando la reducción del riesgo de intrusión, o daños en el ambiente de desarrollo de aplicaciones. A continuación se describe brevemente como se aplicó cada uno.

7.2.1 Principio de defensa en profundidad

Se provisionaron dos capas de Seguridad y aislamiento. La primera capa es la Virtualización con la utilización de la herramienta *Linux Vserver*. Con esta Herramienta se creó un servidor Virtual al que se asignaron recursos necesarios para el desarrollo de aplicaciones de manera restringida, y en el cual se pueden instalar las Herramientas que sean necesarias por parte de los desarrolladores de software. A parte de esto se creó una segunda capa de aislamiento con la utilización de *SCPonly* creando un ambiente confinado para implementar las aplicaciones desarrolladas.

Estas dos capas implementadas se apuntan a los niveles de defensa de redes, defensa de host, defensa de aplicaciones y defensa de datos. Se recomienda que aparte de la seguridad ofrecida por el prototipo, se integre con otras medidas de seguridad tanto dentro de estos mismos niveles de defensa como también dentro de los niveles que no se tuvieron en cuenta (Defensa de perímetros, Seguridad física). En la Figura 3. Defensa en profundidad en el contexto de la seguridad informática se describen los niveles de defensa en profundidad en el contexto de la seguridad informática.

Figura 3. Defensa en profundidad en el contexto de la seguridad informática



Fuente. Microsoft [16]

7.2.2 Segregación de funciones

Para este diseño también se tuvo en cuenta la segregación de funciones por roles. Se planteó un esquema en donde existe un ambiente para la etapa de programación e integración, y otra para la etapa de implementación y mantenimiento. El ambiente para la programación está limitado por el servidor Virtual, en donde se tienen todas las herramientas necesarias para la programación y compilación de código, y a donde solo tiene acceso el personal encargado de estas tareas. El ambiente de Implementación y mantenimiento está restringido en conjunto por la *Sandbox* creada para este fin (al ejecutar la aplicación esta no podrá afectar elementos por debajo de la raíz asignada en el sistema de archivos), y también está restringida por la máquina virtual en cuanto al uso de demás recursos hardware (CPU, memoria, red), y su uso es el probar la ejecución de las aplicaciones web desarrolladas por los programadores (pruebas, implementación, mantenimiento, de las aplicaciones web).

7.2.3 Principio del menor privilegio

El prototipo se diseñó para asignar los menores privilegios. A cada rol de cada etapa en el ciclo de desarrollo se le da solamente los accesos y permisos necesarios. Para el rol de los programadores se da el acceso al servidor virtual con las herramientas de desarrollo. Al personal de

[16] **Microsoft.** Guía de operaciones de seguridad para Windows 2000 Server. [En línea] [Citado el: 5 de Agosto de 2015.]
http://www.microsoft.com/spain/technet/seguridad/2000server/chapters/ch02secops.aspx

implementación y mantenimiento se le da acceso a la *Sandbox* para que realicen la carga de las aplicaciones desarrolladas cada vez que sea necesario.

7.2.4 Principio de Seguridad en caso de fallo

El diseño también abarca este principio ya que en caso de que se presente un fallo en la aplicación, esta debería únicamente afectar la *Sandbox*, o como máximo los recursos asignados al servidor virtual, para lo cual se presume que se están realizando *backups* de la aplicación, y preferiblemente también de la máquina virtual. En caso de que la aplicación tenga un comportamiento que afecte la seguridad del prototipo, sería conveniente proceder con la creación de una nueva *Sandbox* para incluir un *backup* de la aplicación que funcione de manera estable, o restaurar un *backup* de la máquina virtual (archivos del sistema virtual, y archivos de configuración del servidor virtual). Con esto se tiene previsto mejorar la Gestión de Incidentes y la Gestión de continuidad del negocio.

7.3 Requerimientos para el diseño de una Sandbox en los casos de prueba específicos

Para el levantamiento de requerimientos se utilizaron las técnicas de entrevista, lluvia de ideas y escenarios. Para la técnica de entrevista se diseñó el Formato 01 para registrar los requerimientos (Ver Anexo 1).

Figura 4. Formato 01. Relación básica de requerimientos

FORMATO 01 Código: _____	RELACIÓN BÁSICA DE REQUERIMIENTOS				Versión 1
Fecha de creación del formato: Abril 2015	Formato solicitud de creación/modificación de requerimientos				Página 1 de __
Nombre de la dependencia					
Responsable de la dependencia					
Cargo					Ext.
E-mail del responsable de la dependencia					Firma :
Nombre del recolector					
E-mail del recolector					
Número	**Fecha recolección**	**Fuente**	**Descripción**	**Usuarios involucrados**	**Observación (Prioridad/Estado/Tipo)**

Fuente: Autores

Para listar los actores involucrados en los casos específicos se diseñó el Formato 02 (Ver Anexo 2):

Figura 5. Formato 02. Listado de actores en los casos de prueba específicos.

FORMATO 02 Código:_____	LISTA DE ACTORES		Versión 1
Fecha de creación del formato: Abril 2015			Página 1 de __
Caso de prueba especifico		Código	
Nombre del recolector			
E-mail del recolector		Fecha	

Número	Actor (Rol)	Nombre Usuario	Observación/Permisos

Fuente: Autores

7.3.1 Caso de prueba específico No. 1

Para este caso de prueba se contempló un escenario con ambiente académico, donde se requiere dar mayor seguridad al servidor web de una universidad, evitando la ejecución de *scripts* no autorizados e intromisión o ingreso de los usuarios desarrolladores en directorios no autorizados para su ingreso o manipulación, se podría restringir con alguna herramienta. Dicho servidor está destinado a alojar sitios web en desarrollo por parte de los estudiantes tesistas de pregrado de Ingeniería de Sistemas e Informática y Maestría, los cuales pueden tener un nivel de conocimiento más amplio de seguridad informática y mayor capacidad para realizar intromisiones no autorizadas en el servidor, que otros usuarios finales en otras áreas del conocimiento. Uno de los sitios web es un aula virtual, la cual los desarrolladores deben modificar los *scripts* y subirlos nuevamente al servidor, ya que se busca probarlos directamente en un servidor de prueba con la configuración espejo del servidor de producción. Un tipo de usuario son los desarrolladores, otro es el administrador del servidor anfitrión y de los servidores virtuales, cada uno con sus respectivas restricciones, y otro es el usuario del profesor que es director del proyecto de servidores que evalúa los scripts.

7.3.1.1 Análisis de los requerimientos del Caso de prueba específico No. 1

La idea sobre este caso de prueba surge de la necesidad real de un profesor universitario que tiene a su cargo servidores web, donde tiene que permitir el ingreso especialmente a un servidor de pruebas a diferentes usuarios para que modifiquen algunos scripts, ejecuten específicamente otros, de acuerdo a las tareas asignadas, pero se debe controlar de alguna manera el acceso de estos a directorios específicos, y que utilicen determinados recursos de hardware del servidor.

Para el levantamiento de requerimientos se utilizó la combinación de diferentes técnicas, entre ellas, una entrevista al profesor director del proyecto de servidores, una lluvia de ideas entre los

autores del presente proyecto, y la elaboración de un posible escenario de acción para la utilización del prototipo de Sandbox generado. Luego de obtener toda la información se pudieron identificar los requerimientos (Ver Anexo 3) tecnológicos físicos y lógicos esenciales para el diseño de una Sandbox, como solución a la necesidad de restringir el acceso de los usuarios a determinados directorios y la utilización de recursos restringidos en el servidor de acuerdo a la administración del mismo.

Se identificaron como actores principales de los requerimientos (ver Anexo 4): al Director del Proyecto de Servidores Web, al Administrador del Servidor y al Estudiante desarrollador de software web.

7.3.2 Caso de prueba específico No. 2

Para este caso de prueba se contempló un escenario con ambiente empresarial, donde se desea asignar a diferentes usuarios desarrolladores máquinas virtuales con sus respectivas restricciones para que prueben software, ejecuten los scripts que desarrollen para la empresa, en un servidor real y puedan evaluar las incidencias que estos tienen en el sistema o tengan los recursos suficientes para ejecutarlos y se aprovechen más eficientemente los recursos. De esta manera los usuarios se pueden conectar a una determina máquina virtual y probar el software y scripts sin riesgo a dañar o afectar el sistema operativo del servidor anfitrión. Esto con el objeto de probar el software en una máquina lo más parecida al servidor de producción si se necesita, ya que el servidor que sirve para las pruebas puede ser una configuración muy parecida al servidor de producción. Además la idea es que este equipo anfitrión sea lo suficientemente robusto como para poder asignar los recursos de hardware y software que requieran los desarrolladores.

7.3.2.1 Análisis de los requerimientos del Caso de prueba específico No. 2

Este caso de prueba se escogió para probar la implementación del diseño de una Sandbox ya que muchas empresas poseen en su equipo de trabajo, desarrolladores web, que tiene que probar constantemente sus scripts o software nuevo, y sería muy útil el empleo de este esquema de Sandbox para dar solución al servidor de pruebas. Ya que no siempre el desarrollador tiene a su disposición todo los recursos de hardware o software que necesita, y no todo el tiempo los está utilizando, de manera que mientras tanto se pueden asignar esos recursos a otro usuario desarrollador que lo requiera.

Para el levantamiento de requerimientos se utilizó también la combinación de diferentes técnicas, entre ellas, una lluvia de ideas entre los autores del presente proyecto, y la elaboración de un posible escenario de acción para la utilización del prototipo de Sandbox generado.

Luego de obtener toda la información se pudieron identificar los requerimientos (Ver Anexo 5) principales para el diseño de una Sandbox, como solución a la necesidad de restringir el acceso de los usuarios a directorios no autorizados en el servidor de pruebas y a la utilización de recursos restringidos en el servidor de acuerdo a la administración del mismo y a las necesidades de cada desarrollador para probar el software o scripts.

Se identificaron como actores principales de los requerimientos (ver Anexo 6): Jefe del Departamento, Administrador del Servidor y el desarrollador o desarrolladores empresariales.

8. IMPLEMENTACIÓN Y PRUEBAS DE LA SOLUCIÓN SANDBOX

8.1 Herramientas Utilizadas

Se utilizaron diferentes herramientas hardware y software para la realización del prototipo del ambiente confinado, las cuales se detallan a continuación.

8.1.1 Software Utilizado

A continuación se relaciona el Software utilizado para la solución, y se explica brevemente cada uno:

- ✓ *Debian OS*: Sistema Operativo con licencia "*Open Source Software*" (de uso libre), basado en *kernel Linux*. Es uno de los más conocidos a nivel mundial y es soportado por una amplia variedad de arquitectura de computadores, incluyendo los modelos más viejos.
- ✓ *Linux-Vserver*: Aplicación que provee virtualización a los sistemas GNU/Linux mediante aislamiento a nivel del *kernel*. Esto permite ejecutar múltiples unidades virtuales a la vez, las cuales se aíslan de manera suficiente para garantizar la seguridad requerida, y a la vez utilizando los recursos disponibles de manera eficiente.
- ✓ *Util-Vserver*: Paquete de instalación que provee herramientas de usuario para crear y administrar la tecnología de virtualización *Linux-Vserver*.
- ✓ *Open MPI*: *MPI* viene del inglés "*Message Passing Interface*" que traducido al español significa Interfaz de paso de mensajes. *MPI* se refiere a una *API* estandarizada usada para la computación en paralelo y/o distribuida, que ha sido publicada en varias versiones. *Open MPI* en una implementación de software con licencia de uso libre para las especificaciones *MPI*.
- ✓ *Scponly*: Shell alternativa y restringida para los administradores de sistemas que desean dar acceso remoto a usuarios para leer y escribir archivos locales sin proporcionar ningún tipo de privilegio de ejecución remota.
- ✓ *Apache Http Server*: Proyecto software de la fundación *Apache Software* "*The Apache Software Foundation*", que es un esfuerzo para desarrollar y mantener un servidor web con licencia de uso libre para los sistemas operativos modernos incluyendo *UNIX* y *Windows NT*.
- ✓ *Win SCP*: Aplicación con licencia de uso libre, que funciona como cliente *SFTP* gráfico para Windows que utiliza *SSH*. Su función principal es facilitar la transferencia segura de archivos entre dos sistemas informáticos, el local y uno remoto que ofrezca servicios *SSH*.

8.1.2 Hardware Utilizado

La instalación se realizó en un equipo con las siguientes características:

- ✓ Procesador Intel core I5-2410M
- ✓ Disco Duro 640 GB
- ✓ Tarjeta de red 100/1000
- ✓ Memoria DDR3 4GB

8.2 Instalación de *Linux Vserver*

Para la instalación de la herramienta *Linux Vserver* se efectuaron los siguientes pasos:

- ✓ Se realizó la instalación del *kernel Linux Vserver* con el comando que se describe a continuación:

```
aptitude install linux-image-2.6.32-5-vserver-amd64
```

- ✓ Se instaló el paquete *ssh*:

```
aptitude install ssh
```

✓ Se instalaron las *util-vserver*:

Se puede instalar de cualquiera de las siguientes dos formas, y se eligió instalar con la segunda opción:

1. Instalación mediante aptitude:

```
aptitude install util-vserver
```

2. Instalacion *util-vserver last no stable* con compilación mediante make

Se eligió instalar usando esta opción. Se descargó la última versión disponible de útil-vserver, se descomprimió el archivo descargado y se procedió a instalar

```
./configure
make
make check
make install
make installcheck
make install-distribution
```

Después de instalar y reiniciar, al ejecutar el comando uname –r dio una salida como la que sale a continuación:

```
Linux DEBIANUPBSBX 2.6.32-5-vserver-amd64 #1 SMP Mon Mar 4 23:03:09 UTC 2012 x86_64 GNU/Linux
```

Como se puede observar, el k*ernel* ya aparece como *VServer*.

8.3 Instalación Sistema Invitado

Se observan las ayudas de la opción *build* de *VServer*, la cual se utiliza para Instalar los sistemas Invitados:

```
Vserver – build –-help
```

Utilizando el siguiente comando se realiza la instalación nuevo servidor virtual:

```
Vserver vserver3 build –m debootstrap –-context 45 –-hostname vserver3.mydomain.com –-interface eth0:192.168.0.100/24
-- -d squeeze –m http://ftp.us.debian.org/debian -- --arch amd64
```

Al ejecutar el comando anterior se comienza la instalación del sistema base y opciones del vserver1 *guest*:

```
…
I: Base system installed successfully
root@debian6:~#
```

Al finalizar aparecen las líneas anteriores.

8.3.1 Postinstalación

Una vez se construye el *vserver* es posible iniciarlo con la siguiente instrucción:

```
vserver vserverX start
```

Después de iniciado es posible utilizar las siguientes opciones de comandos para realizar acciones dentro del *vserver*:

vserver vserverX enter
ó
vserver vserverX exec ...

Para parar el *vserver* se debe utilizar:

vserver vserverX stop

8.3.2 Eliminación de un *vserver* dañado:

Utilizando la siguiente instrucción se puede eliminar manualmente un *vserver* creado anteriormente:

vserver vserverX delete

8.4 Configuraciones y administración de los *vserver*

A continuación se describen las opciones de configuración y administración de los *VServer*.

8.4.1 Configuración de la Interfaz Virtual del *VServer*

Lo primero que se configuró fueron las Interfaces de red de *vserver*. La información de las interfaces con sus respectivas IP, máscaras de red, nombre de la Interfaz, etc. de los *vserver* se pueden editar en el directorio:

/usr/local/etc/vservers/**(nombre del vserver)/**interfaces/**(Id de la interface)**

En donde:

Nombre del *vserver*: por ejemplo vserver1
Id de la interface: la primera por default es 0 (cero).

8.4.2 Monitoreo de los VServer

Para ver información general de los *vservers* arrancados se utiliza el siguiente comando:

vserver-stat

Para copiar archivos en *vserverx* se copian desde el host en la ruta correspondiente a la raíz del *Vserver*:

/var/lib/vservers/vserverx/...

Si se presenta el problema (*procfs-security*) al iniciar *vserverX*, se resuelve ejecutando el siguiente script:

/usr/local/etc/init.d/vprocunhide start

8.5 Aplicando límites de CPU a los vserver

Se aplicó restricción de CPU mediante CGROUPS. Para aplicar límites de CPU primero se verificó

que el *kernel* se encontraba configurado para permitir aplicar estos límites mediante el siguiente comando:

```
# cat /boot/config-2.6.32-5-vserver-amd64 | grep CONFIG_VSERVER_HARDCPU
  CONFIG_VSERVER_HARDCPU=y
```

Como se puede ver la opción de configuración de *CPU* del *VServer* está habilitada.

Para las pruebas de *CPU* se utilizó un script llamado script suma_paralelo_2.c, el cual tiene alto consumo de *CPU*, y que se puede ejecutar con una instancia para que haga uso de una sola *CPU* o se puede ejecutar con varias instancias del script para uso de varias *CPUs* entre las disponibles. Este script utiliza la librería mpi de procesamiento en paralelo. Se copió el script desde el host hacia el *guest vserver3* (/var/lib/vservers/vserver3/).

8.5.1 Instalación de paquetes necesarios para la ejecución del script dentro del servidor virtual

Se Instalaron los paquetes necesarios para la ejecución del script con los comandos descritos a continuación:

```
aptitude install g++

aptitude install openmpi-bin openmpi-common libopenmpi1.3 libopenmpi-dbg libopenmpi-dev
```

8.5.2 Compilación y ejecución del script para pruebas de procesamiento con *MPI*:

Se ejecutó el Script con los comandos descritos a continuación:

Compilación:

```
mpicxx suma_para.cc -o suma_para -DMPICH_IGNORE_CXX_SEEK
```

Ejecución:

```
mpiexec -n 4 ./suma_para
```

8.5.3 Limitando recursos CPU con cgroup:

Para aplicar límites mediante *cgroup* se realizaron las siguientes acciones:

- ✓ Se creó una carpeta llamada *cgroup* dentro de la ruta de configuración del *vserver*:

```
mkdir /usr/local/etc/vservers/vserver3/cgroup
```

- ✓ Se creó un archivo en esta carpeta *cgroup* con el nombre *cpuset.cpus* que contiene las *CPUs* deseadas. p.e: “0” o “0,1” o “0-2” (sin comillas).

Por ejemplo:

0--> solo usar cpu 0
0,2 → usar cpus 0 y 2
0-2 → usar cpus 0,1 y 2
1-3 → usar cpus 1,2 y 3

8.5.3.1 Límites temporales de CPU

Para que los límites con *cgroup* funcionen temporalmente (mientras no se reinicie de nuevo el host) se monta el *cgroup* de la siguiente forma:

✓ crear carpeta *cgroup* en los dispositivos:

```
mkdir /dev/cgroup
```

✓ montar *cgroup* de la siguiente forma:

```
mount -t cgroup -o cpu,cpuset none /dev/cgroup
```

Después del montaje se observa que el script solo utiliza *CPUs* indicadas.

8.5.3.2 Límites permanentes de CPU

Si se quiere que los límites de CPU con *cgroup* funcionen permanentemente, se debe añadir un registro de montaje en la *fstab* (*/etc/fstab*) de la siguiente forma:

✓ agregar la línea **"none /dev/cgroup cgroup cpu,cpuset 0 2"** en la fstab

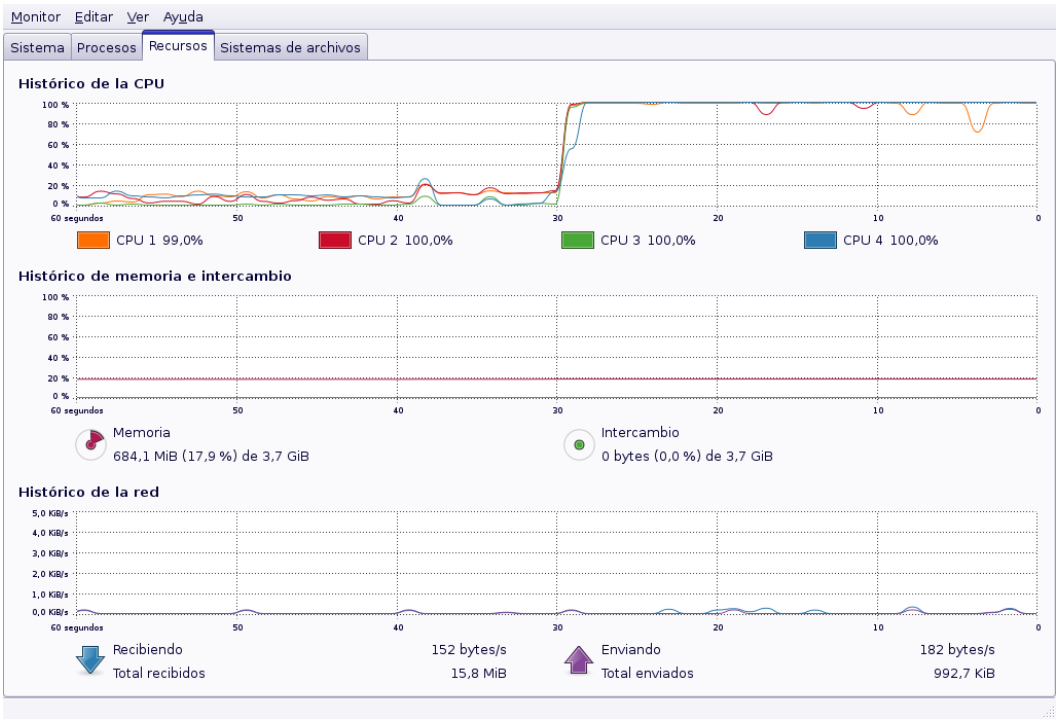
NOTA: Al reiniciar el host se pierde el punto de montaje: */dev/cgroup*, se borra la carpeta al reiniciar. Para esto se generó un script de inicio que cree esta carpeta al iniciar el Sistema Operativo.

8.5.3.3 Pruebas

Se probaron los límites de CPU aplicados utilizando el comando *mpiexec* del software *OpenMpi* para lanzar 4 instancias de un script ejecutable llamado suma_paralelo:

Se ejecutó el comando "mpiexec -n 4 ./suma_paralelo_2_exe_v1"(con 4 instancias) antes de aplicar límites de CPU y se obtuvo el siguiente resultado al monitorear el uso de procesamiento:

Figura 6. Prueba uso CPU sin límite de recursos

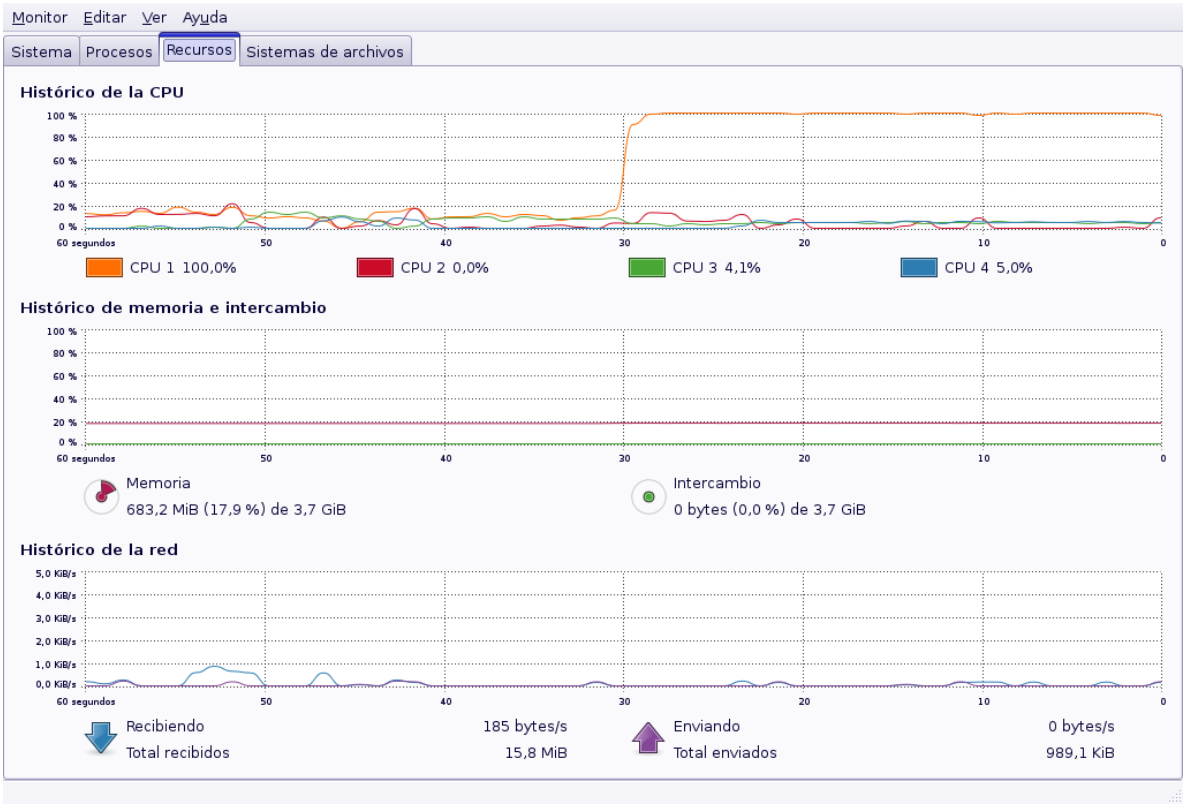


Fuente. Autores

Como se puede ver en la Figura 5, al ejecutar el comando sin restricciones de *CPU*, este utilizó todo el procesamiento disponible del host. Se puede ver que los cuatro núcleos del procesador (*CPU1, CPU2, CPU3* y *CPU4*), identificados con diferentes colores, se utilizan aproximadamente al 100% al ejecutar el script para procesamiento en paralelo con la ayuda de *mpiexec*.

Al ejecutar el mismo comando realizando límite de CPU por *cgroup*, con restricción de uso únicamente de una CPU (la CPU 0) se obtiene el siguiente resultado:

Figura 7. Prueba uso CPU restricción un procesador

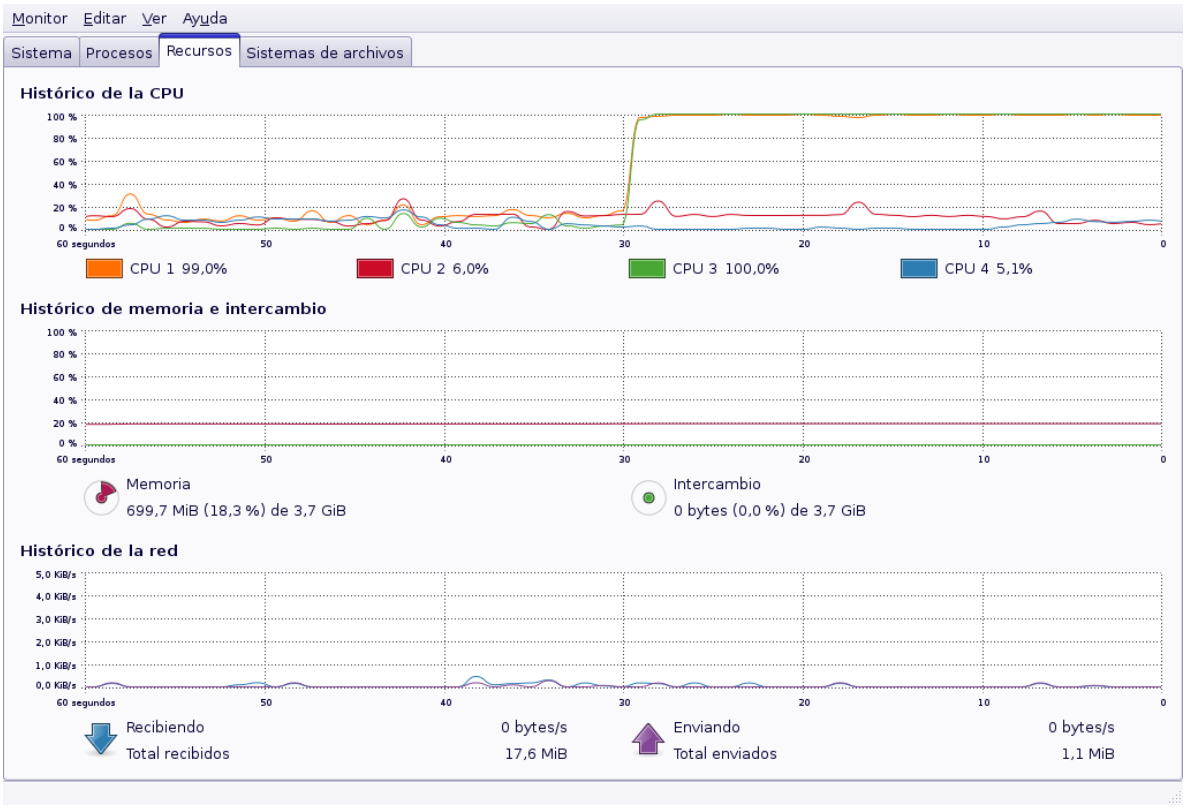


Fuente. Autores

Como se puede ver en la Figura 6., se hizo efectiva la restricción de *CPU* a únicamente la *CPU 0* del host (*CPU 1* en el gráfico e identificada con color naranja). Se observa que ejecutando el mismo comando de procesamiento en paralelo en el servidor virtual que antes de la restricción consumía todos los recursos de procesamiento (los cuatro núcleos *CPU*), y después de restringir dentro del mismo servidor virtual al uso de un solo procesador (*CPU 1*), los gráficos de uso de procesador del host que lo contiene muestra que solo se está haciendo uso del recurso *CPU* que se asignó (línea naranja en la gráfica correspondiente al consumo del 100% de recurso de la *CPU1*).

Ejecutando el mismo comando pero restringiendo el uso de *CPUs* únicamente a las *CPUs* 0 y 2 (dando valor de 0,2 al fichero *cpuset.cpus*), se obtiene el siguiente resultado:

Figura 8. Prueba uso *CPU* restricción dos procesadores



Fuente. Autores

Como se puede ver, se hizo efectivo la restricción de *CPU* a únicamente las *CPU* 0 y 2 del host (*CPU* 1 y 3 en el gráfico). Se ven las líneas naranja y verde de la gráfica que corresponden a la *CPU* 1 y *CPU* 3 que muestran un uso del 100% de estos recursos, mientras que los otros dos núcleos de procesamiento (*CPU2* y *CPU4*) que no fueron asignados al servidor virtual, no son utilizados para procesar el script de procesamiento en paralelo ejecutado dentro de la instancia de virtualización.

8.6 Límites de Memoria en los VServer

8.6.1 Revisión de la configuración de memoria antes de la restricción de los límites

Primero que todo, se examinó la configuración del tamaño de página inicial dentro del vserver para esto se ejecuta:

```
echo 'int main () { printf ("%dKiB\n", getpagesize ()/1024); return 0; }' | gcc -xc - -o getpagesize && ./getpagesize
```

El resultado del comando fue:

```
<stdin>: In function ‘main’:
<stdin>:1: warning: incompatible implicit declaration of built-in function ‘printf’
4KiB
```

Con el resultado anterior se observa que el tamaño de página es 4KB.

Después con el uso de otros comandos que se describen a continuación se verificó la asignación de memoria. Sin aplicar límites de memoria la salida del comando *vlimit* para mostrar restricción de memoria desde el host fue la siguiente:

```
vlimit -c 45 -a -d | grep RSS
RSS        N/A              inf              inf
```

La salida del comando free desde el *guest* sin aplicar límites (por default muestra los del host) fue la siguiente:

```
root@vserver3:/# free
             total       used       free     shared    buffers     cached
Mem:       2053396     753188    1300208          0      83300     398080
-/+ buffers/cache:     271808    1781588
Swap:      3905528          0    3905528
```

Para que tome los valores del servidor virtual ejecutar el siguiente comando y reiniciar el *guest*:

```
echo "VIRT_MEM" >> /usr/local/etc/vservers/vserver3/flags
```

Después de esto al ejecutar el comando free muestra lo siguiente (lo mismo en el total):

```
free
             total       used       free     shared    buffers     cached
Mem:       3907756       3912    3903844          0     172424          0
-/+ buffers/cache: 18014398509313472    4076268
Swap:      3905528          0    3905528
```

8.6.2 Aplicando límites temporales de memoria

Las siguientes conversiones se utilizaron para asignar los límites de memoria:

memoria RAM total = 280000 KB= 70000 pag * 4 KB/pag = aprox. 280 MB
memoria SWAP total = 120000 KB = 30000 pag * 4 KB/pag = aprox. 120 MB

memoria total = 400000 KB = 100000 pag * 4 KB/pag = aprox. 400 MB

Lo anterior se debe aplicar al contexto 45 (que es el contexto del vserver3), los valores son en páginas (100000 paginas = 400000 KB)

8.6.2.1 Límites temporales de memoria

Para aplicar límites de memoria al vserver3 temporalmente se ejecutan los siguientes comandos desde el host:

```
/usr/sbin/vlimit -c 45 --rss 70000
/usr/sbin/vlimit -c 45 –S --rss 30000
```

Las opciones de los comandos corresponden a lo siguiente:

--rss → rss.hard
-S --rss → rss.soft

La configuración de memoria del Vserver3 queda de la siguiente forma:

cantidad de RAM = rss.soft = 70000
cantidad de SWAP = rss.hard – rss.soft = 30000 (hard menos soft)

8.6.2.2 Límites permanentes de memoria

Para aplicar los límites de memoria permanentemente (al iniciar el *vserver*) se creó la carpeta rlimits:

```
mkdir /usr/local/etc/vservers/vserver3/rlimits
```

Dentro de esta carpeta se crearon los archivos para límites *soft* y *hard*:

```
cd /usr/local/etc/vservers/vserver3/rlimits
vi rss.hard
vi rss.soft
```

rss.hard es el archivo que contiene el *hard limit* p.e 100000 para este caso
rss.soft es el archivo que contiene el *hard limit* p.e 70000 para este caso

8.6.2.3 Pruebas

Primero se comprobaron los límites asignados desde el host:

```
root@Sandbox1:/usr/local/etc/vservers/vserver3# vlimit -c 45 -a -d | grep RSS
RSS     N/A        70000        100000
```

Se observó que ya aparecían los límites.

Después se comprobaron los resultados dentro del *vserver*:

```
root@vserver3:/# free –k
              total      used          free              shared   buffers          cached
Mem:         280000      4872        275128                   0   153976                0
-/+  buffers/cache:     18014398509332880        429104
Swap:        120000         0        120000
root@vserver:/#
```

En la Imagen anterior se puede ver que ya aparecen los límites aplicados dentro del *VServer* según las configuraciones realizadas.

8.7 Límites de disco en los *vserver*

8.7.1 Adaptación inicial del disco para utilización de los *VServers*

Se realizó primero límite a la ubicación raíz en donde van a quedarían los *guest*, con el fin de montar ésta ubicación en una partición física o lógica, con lo cual el límite lo da la partición.
Se decidió reasignar tamaño (reducir) de la partición de *linux debian /dev/sda3* la cual tenía un sistema de archivos ext4. Inicialmente esta partición tenía aproximadamente 26 GB de tamaño y estaba usando aproximadamente 7GB. Para este fin se utilizó la herramienta *parted magi*c (*cd boteable linux*), y desde esta específicamente las herramientas *clonezilla* (para realizar *backup* de la partición) y *gparted* (para el reparticionamiento).

Con el espacio removido de la partición sda3 (14 GB que después de la reasignación quedaron como espacio unallocated), se creó la partición sda4 con formato ext4, que queda de la siguiente forma:

```
fdisk /dev/sda
Disco /dev/sda: 640.1 GB, 640135028736 bytes
255 heads, 63 sectors/track, 77825 cylinders
Units = cilindros of 16065 * 512 = 8225280 bytes
```

```
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x47454745
Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
/dev/sda1   *          1        8924    71678278+   7  HPFS/NTFS
/dev/sda2          12262       77824   526628865    f  W95 Ext'd (LBA)
/dev/sda3           8924       10713    14371840   83  Linux
/dev/sda4          10713       12262    12440576   83  Linux
/dev/sda5          12749       77824   522722938+   7  HPFS/NTFS
/dev/sda6          12262       12748     3905536   82  Linux swap / Solaris
```

8.7.2 Aplicando límites de disco para los *guest*

Se montó con la opción de *tag* la partición sda4 en un nuevo directorio creado como /vservers (tener cuidado al usar la opción de *tag*, esta debe ser usada en una partición diferente a la de root):

```
root@Sandbox1:/dev# mount -o tag /dev/sda4 /vservers
root@Sandbox1:/dev# df -h /dev/sda4
S.ficheros            Size  Used Avail Use% Montado en
/dev/sda4              12G  158M   11G   2% /vservers
root@Sandbox1:/dev# df -T /dev/sda4
S.ficheros    Tipo Bloques de 1K   Usado    Dispon Uso% Montado en
/dev/sda4     ext4    12245176    161456  11461692   2% /vservers
```

La opción *tag* se utiliza para etiquetar esta partición y así permitir la creación de *quotas* para limitar espacio en disco de manera personalizada.

Se copiaron los vservers que ya estaban creados en la nueva partición montada:

```
root@Sandbox1:/# cd vservers/
root@Sandbox1:/vservers# ls
vserver1  vserver3
```

De esta forma ya los *Vservers* creados no podrán superar los límites de la Partición en donde se encuentran. Ahora la creación de los límites de disco a cada uno de los *vservers* de forma individual se utilizó la asignación de cuotas y se configuraron para aplicar los límites de manera permanente.

Para este fin se realizaron las siguientes acciones:

Se etiquetaron "*tag*" todos los archivos del *guest* con el contexto de este utilizando el comando *chxid*:

```
root@Sandbox1:/vservers# chxid --help
Usage:  chxid -c <ctx|vserver> [-RxU] [--] <file>+

 Options:
   -R  ...  recurse through directories
   -c  ...  assign the given context/vserver to the file(s)
   -x  ...  do not cross filesystems
   -U  ...  skip unified files

root@Sandbox1:/vservers# chxid -URx -c vserver3 /vservers/vserver3
```

Después se crearon los archivos de configuración de límite de disco, en este caso para el servidor virtual vserver3. Los límites se configuraron de la siguiente forma:

Límite de disco: 2 GB
Directorio que se limita: /vservers/vserver3 (directorio root del servidor virtual vserver3)
Límite de inodos: 1356466
Porcentaje reservado para el usuario root: 5%

Para esto se creó la carpeta y los archivos de configuración de límite de disco como se muestra a continuación:

```
root@Sandbox1:/vservers# cd /usr/local/etc/vservers/vserver3/
root@Sandbox1:/usr/local/etc/vservers/vserver3# mkdir dlimits
root@Sandbox1:/usr/local/etc/vservers/vserver3# cd dlimits/
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits# mkdir root
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits# ls
root
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits#cd root/
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# vi directory
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# cat directory
/vservers/vserver3
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# vi space_total
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# cat space_total
2097152
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# vi reserved
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# cat reserved
5
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root#
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# vi inodes_total
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# cat inodes_total
1356466
```

La configuración quedó como se muestra en la siguiente imagen:

```
root@sandbox1:/# cd /usr/local/etc/vservers/vserver3/dlimits/root/
root@sandbox1: /usr/local/etc/vservers/vserver3/dlimits/root/# ls
directory       inodes_total   reserved  space total
root@sandbox1: /usr/local/etc/vservers/vserver3/dlimits/root# cat *
/vservers/vserver3
1356466
5
2097152
root@sandbox1: /usr/local/etc/vservers/vserver3/dlimits/root#
```

Se comprobó dentro del *vserver* donde se pudo observar la nueva configuración del límite del disco con 2GB:

```
root@vserver3:/# df –h
Filesystem                    Size    Used    Avail    Use%    Mounted on
/dev/hdv1           2.0G    615M    1.3G     32%    /
None                          16M          0    16M       0%    /tmp
root@vserver3:/#
```

Desde el host se confirmó el espacio utilizado por el *vserver*:

```
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# du –sh /vservers/vserver3
615M       /vservers/vserver3
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root#
```

También desde el host se confirmó el tamaño de la partición que contiene los *vservers* (*/vservers*):

```
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root# df –h /vserv ers/
S.ficheros                  Size    Used    Avail    Use%    Montado en
/dev/sda4                   14G     5,0G    8,0G      39%    /vservers
root@Sandbox1:/usr/local/etc/vservers/vserver3/dlimits/root#
```

8.8 Administración y límites de red

Se configuraron los parámetros de red con el uso del comando *tc* como se describe en las siguientes sub-secciones.

8.8.1 Configuración Inicial de la red

Se decidió aplicar *classful queueing* para implementar límites de red.

Se añadió la "*bcapabilitie*" (capacidad tipo POSIX) NET_ADMIN en el vserver3 con el fin de posibilitar la configuración las interfaces de red, como se muestra a continuación:

```
root@Sandbox1:/usr/local/etc/vservers/vserver3# echo NET_ADMIN >> bcapabilities
root@Sandbox1:/usr/local/etc/vservers/vserver3# ls
apps           cache  context dlimits  fstab          name    run  vdir
bcapabilities  cgroup  cpuset  flags       interfaces  rlimits  uts
root@Sandbox1:/usr/local/etc/vservers/vserver3# cat bcapabilities
NET_ADMIN
```

Se verificó la configuración original (por omisión) de las *iptables*.

```
root@Sandbox1:/home/upb1# iptables -nL
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Se observó que no se tenía ninguna configuración inicialmente.

Se realizó la carga de los módulos '*dummy*' del *Kernel*, los cuales son drivers que permiten asociar los *VServers* con interfaces de red virtuales, en lugar de asociarlos con las interfaces Físicas del host, y lo cual funciona para ocultar los contadores de los paquetes desde el *VServer*. Los contadores de paquetes pueden ser utilizados por atacantes que tomen control de un *VServer* y quiera hacer ataques de tipo *side-channel* durante un criptoanálisis o durante un análisis de tráfico contra el host del *VServer* controlado u otro *VServe*r.

```
root@Sandbox1:/home/upb1# modprobe dummy
```

Se confirmó la carga del módulo *dummy* del *Kernel*, como se muestra a continuación:

```
root@Sandbox1:/# cd /boot/
root@Sandbox1:/boot# cat config-2.6.32-5-vserver-amd64 | grep dummy
root@Sandbox1:/boot# cat config-2.6.32-5-vserver-amd64 | grep DUMMY
CONFIG_DUMMY=m
CONFIG_ATM_DUMMY=m
CONFIG_DUMMY_CONSOLE=y
CONFIG_SND_SEQ_DUMMY=m
CONFIG_SND_DUMMY=m
CONFIG_SPEAKUP_SYNTH_DUMMY=m
root@Sandbox1:/boot#

root@Sandbox1:/home/upb1# lsmod | grep dummy
dummy                  1584  0
root@Sandbox1:/home/upb1#

root@Sandbox1:/home/upb1# modprobe -c | grep dummy
alias dummy0 dummy
alias dummy1 dummy
alias rtnl_link_dummy dummy
alias snd_seq_client_14 snd_seq_dummy
alias symbol:ata_dummy_port_info libata
alias symbol:ata_dummy_port_ops libata
alias symbol:i2c_new_dummy i2c_core
```

Se configuró la Interfaz virtual dummy0 en el archivo interfaces del host:

```
root@Sandbox1:/home/upb1# vi /etc/network/interfaces
```

Allí se agregaron las siguientes líneas:

auto dummy0
iface dummy0 inet static
address 192.168.1.1
netmask 255.255.255.0

Si se requiere aumentar el número de Interfaces virtuales se debe ejecutar:

```
root@Sandbox1:/home/upb1# modprobe dummy numdummies=10
```

y así es posible asociar cada *VServer* a una Interfaz Virtual única.

Después de esto se asoció la interfaz *dummy0* al *VServer3*:

```
root@Sandbox1:/home/upb1# cd /usr/local/etc/vservers/vserver3/interfaces/0/
root@Sandbox1:/usr/local/etc/vservers/vserver3/interfaces/0# echo dummy0 > dev
root@Sandbox1:/usr/local/etc/vservers/vserver3/interfaces/0# ls
dev  ip  prefix
root@Sandbox1:/usr/local/etc/vservers/vserver3/interfaces/0# echo 192.168.1.100 > ip
root@Sandbox1:/usr/local/etc/vservers/vserver3/interfaces/0# echo 100 > name
root@Sandbox1:/usr/local/etc/vservers/vserver3/interfaces/0# cat *
dummy0
192.168.1.100
100
24
```

Se verificó que la configuración ya apareciera correctamente dentro del *vserver3*:

```
root@vserver3:/# ifconfig
dummy0    Link encap:Ethernet  HWaddr de:5a:34:5a:63:dc
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

dummy0:100 Link encap:Ethernet  HWaddr de:5a:34:5a:63:dc
          inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1008 (1008.0 B)  TX bytes:1008 (1008.0 B)
```

Se realizó la configuración de las *iptables* para que el host actúe como *router* hacia la *vlan* de los *vservers*:

Se verificó la configuración de red del host:

```
root@Sandbox1:/# /etc/init.d/networking restart
Running /etc/init.d/networking restart is deprecated because it may not enable again some interfaces ... (warning).
Reconfiguring network interfaces...done.
root@Sandbox1:/# ifconfig
```

```
dummy0    Link encap:Ethernet  HWaddr de:5a:34:5a:63:dc
          inet addr:192.168.0.100  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::dc5a:34ff:fe5a:63dc/64 Scope:Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

dummy0:100 Link encap:Ethernet  HWaddr de:5a:34:5a:63:dc
          inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1

eth0      Link encap:Ethernet  HWaddr 20:6a:8a:4a:6a:2d
          inet addr:172.16.0.10  Bcast:172.16.0.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:19

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1008 (1008.0 B)  TX bytes:1008 (1008.0 B)

wlan0     Link encap:Ethernet  HWaddr 20:7c:8f:6c:2f:35
          inet addr:192.168.1.202  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::227c:8fff:fe6c:2f35/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:883 errors:0 dropped:0 overruns:0 frame:0
          TX packets:47 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:51356 (50.1 KiB)  TX bytes:9768 (9.5 KiB)
```

Dentro del host se ejecutó lo siguiente:

```
root@Sandbox1:/# iptables -t nat -A PREROUTING ! -s 192.168.1.0/24 -d 192.168.1.0/24 -j DNAT --to-destination
192.168.1.100
```

El anterior comando significa:

(todos los paquetes que van a entrar con origen (*source*) diferente (! -s) a 192.168.1.0/24 y que tengan destino (-d) 192.168.1.0/24 cambiar el destino (-j DNAT) por 192.168.1.10)

```
root@Sandbox1:/# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 ! -d 192.168.1.0/24 -j SNAT --to-source
192.168.1.202
```

El anterior comando significa:

(todos los paquetes que van hacia el origen (*source*) 192.168.1.0/24 y que tengan destino diferente a 192.168.1.0/24 (! -d) cambiar el origen (-j SNAT) por 192.168.1.202)

Se reinició la red y se verificaron las configuraciones realizadas:

```
root@Sandbox1:/# etc/init.d/networking restart
Running etc/init.d/networking restart is deprecated because it may not enable again some interfaces ... (warning).
Reconfiguring network interfaces...RTNETLINK answers: No such process
done.
root@Sandbox1:/# iptables -nL -t nat
Chain PREROUTING (policy ACCEPT)
```

```
target     prot opt source               destination
DNAT       all  -- !192.168.1.0/24      192.168.1.0/24      to:192.168.1.100

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
SNAT       all  --  192.168.1.0/24     !192.168.1.0/24      to:192.168.1.202

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

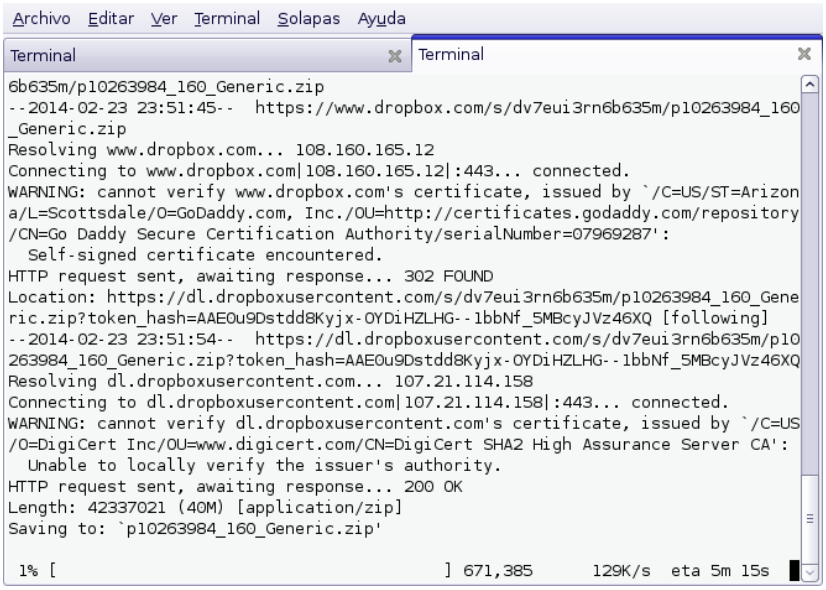
Posteriormente, se probó la conexión desde el host hacia el *VServer* y desde el *Vserver* hacia el host satisfactoriamente:

```
root@Sandbox1:/# ping 192.168.1.100
PING 192.168.1.100 (192.168.1.100) 56(84) bytes of data.
64 bytes from 192.168.1.100: icmp_req=1 ttl=64 time=0.076 ms
64 bytes from 192.168.1.100: icmp_req=2 ttl=64 time=0.065 ms
64 bytes from 192.168.1.100: icmp_req=3 ttl=64 time=0.068 ms
64 bytes from 192.168.1.100: icmp_req=4 ttl=64 time=0.071 ms
64 bytes from 192.168.1.100: icmp_req=5 ttl=64 time=0.072 ms
64 bytes from 192.168.1.100: icmp_req=6 ttl=64 time=0.071 ms

root@vserver3:/# ping 192.168.1.202
PING 192.168.1.202 (192.168.1.202) 56(84) bytes of data.
64 bytes from 192.168.1.202: icmp_req=72 ttl=64 time=0.095 ms
64 bytes from 192.168.1.202: icmp_req=73 ttl=64 time=0.095 ms
64 bytes from 192.168.1.202: icmp_req=74 ttl=64 time=0.068 ms
64 bytes from 192.168.1.202: icmp_req=75 ttl=64 time=0.071 ms
64 bytes from 192.168.1.202: icmp_req=76 ttl=64 time=0.064 ms
64 bytes from 192.168.1.202: icmp_req=77 ttl=64 time=0.070 ms
64 bytes from 192.168.1.202: icmp_req=78 ttl=64 time=0.074 ms
```

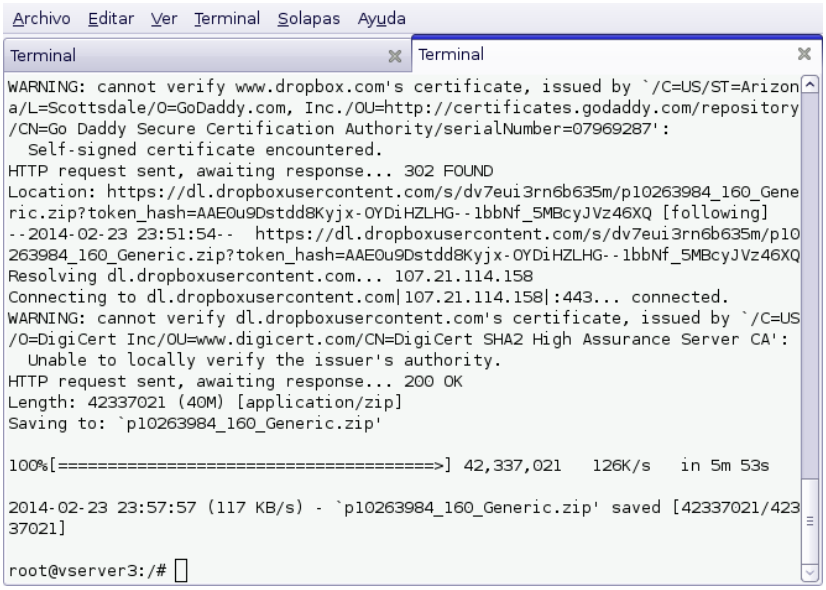
Se probó la velocidad de descarga desde el *VServer3* antes de configurar los límites de red y Se observó que la descarga se hace a 126 K/s en promedio como se ve en las siguientes imágenes:

Figura 9. Prueba Velocidad de Descarga sin restricción



Fuente. Autores

Figura 10. Prueba Velocidad de Descarga sin restricción 2



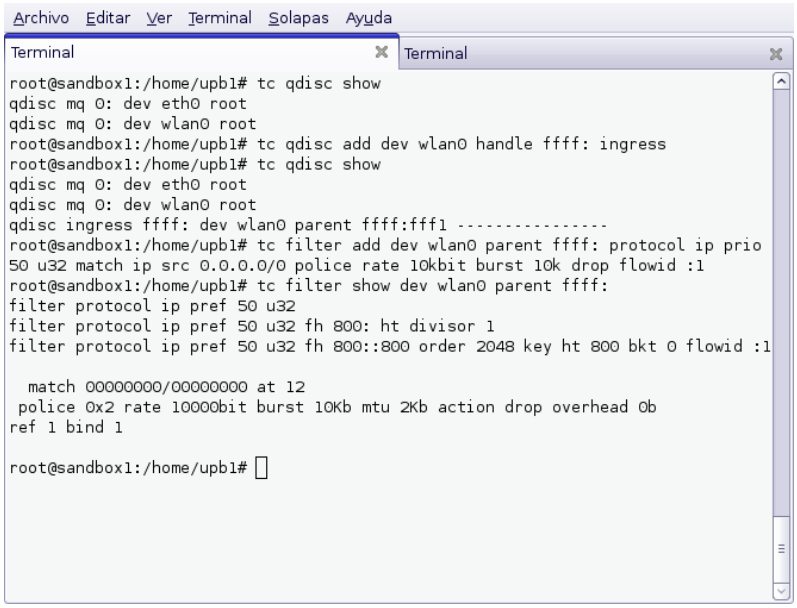
Fuente. Autores

8.8.2 Aplicando Límites de Red

Se realizó la restricción de red limitando la velocidad de bajada desde el host. Para esto se aplicaron límites a la Interfaz de red inalámbrica del host, y con ayuda del comando *tc* de la siguiente forma:

- ✓ Primero se aplicó límite ajustando el *rate* en 10 kb.

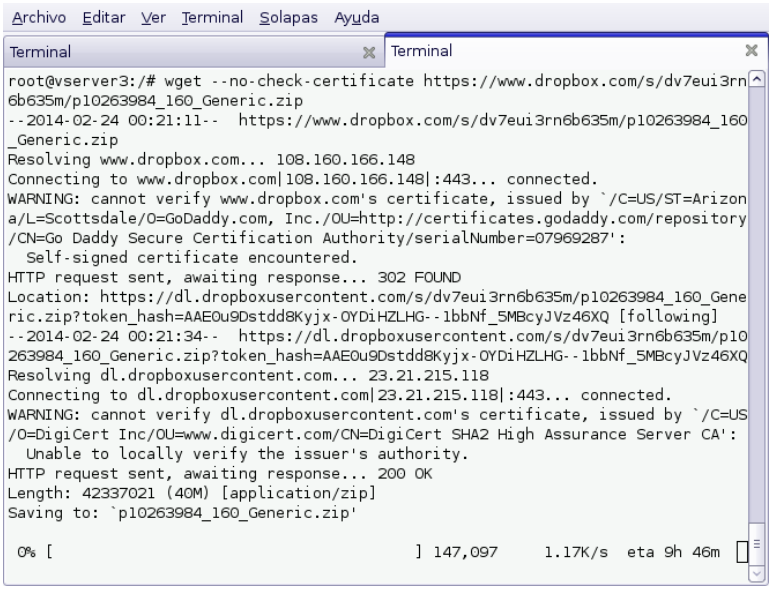
Figura 11. Restricción de velocidad de descarga a 10 kbit con filtro TC



Fuente. Autores

✓ Se obtuvo el siguiente resultado:

Figura 12. Prueba Velocidad de Descarga con restricción de 10kbit/s



Fuente. Autores

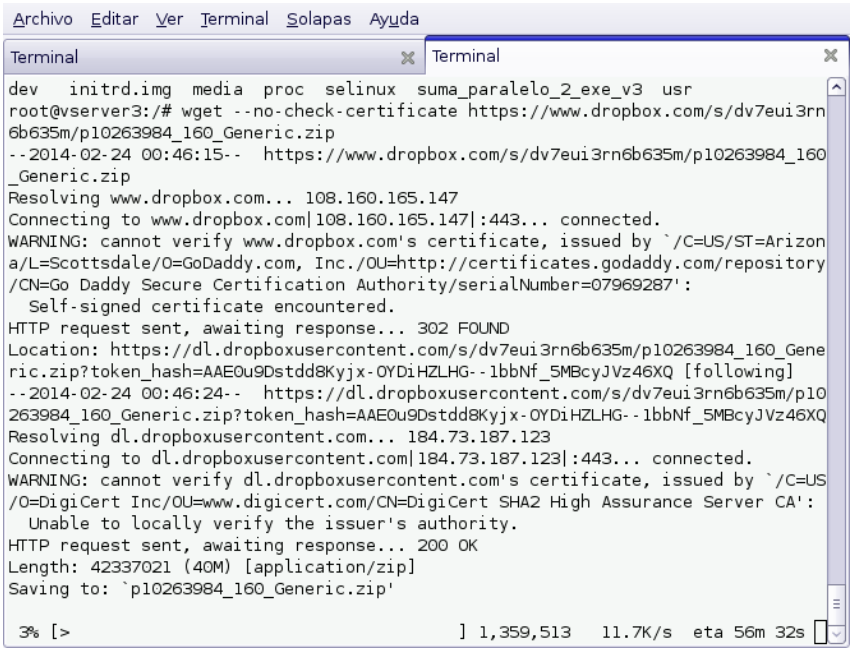
Luego se aplicó límite ajustando el *rate* en 100 kb:

Figura 13. Restricción de velocidad de descarga a 100 kbit con filtro TC

Fuente. Autores

Y se obtuvo el siguiente Resultado:

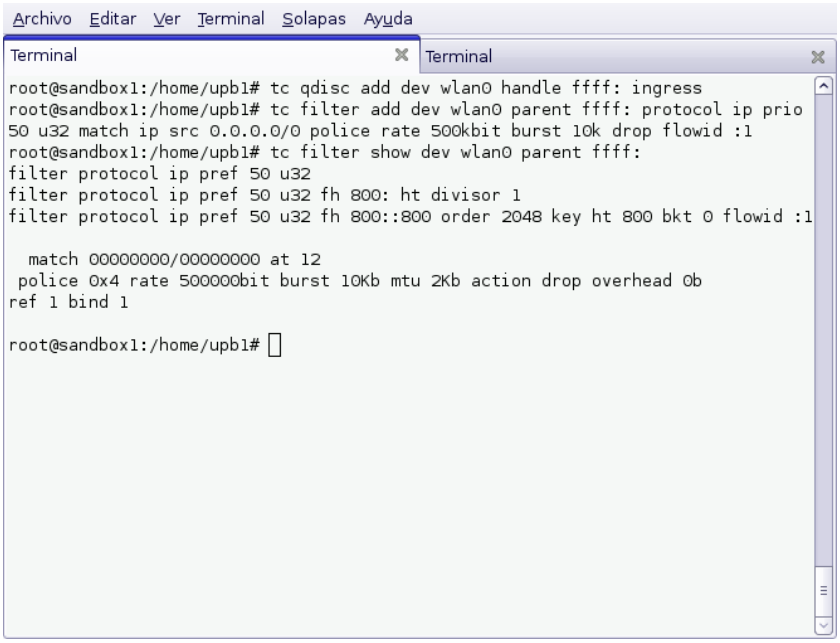
Figura 14. Prueba Velocidad de Descarga con restricción de 100 kbit/s



Fuente. Autores

Por último se probó ajustando el *rate* en 500kbit:

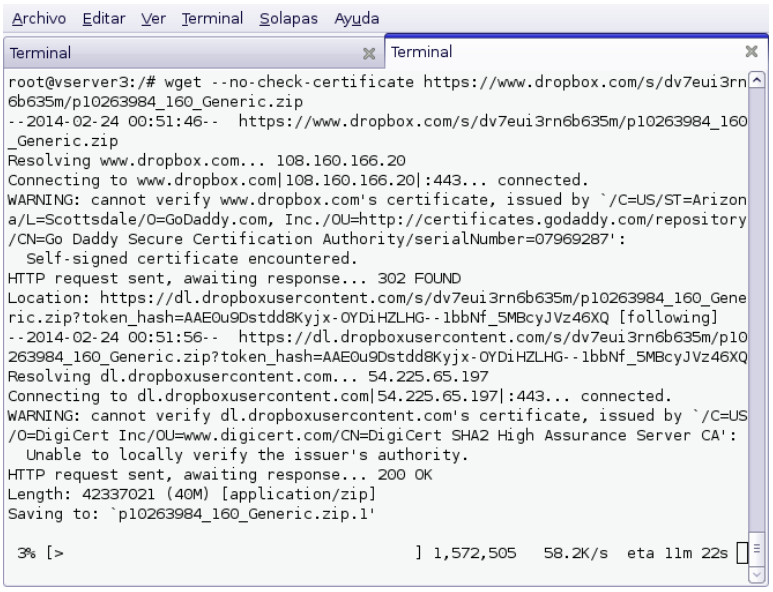
Figura 15. Restricción de velocidad de descarga a 500 kbit con filtro TC



Fuente. Autores

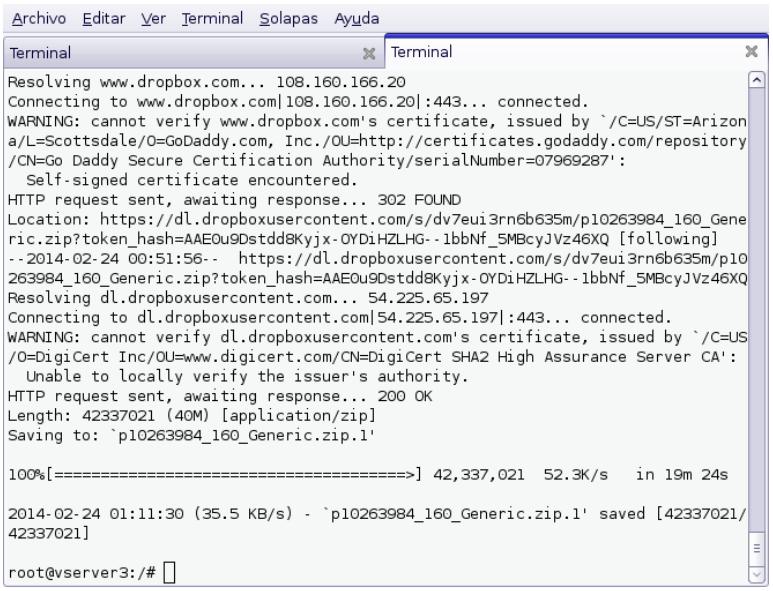
Y se obtuvieron los siguientes resultados:

Figura 16. Prueba Velocidad de Descarga con restricción de 500kbit/s



Fuente. Autores

Figura 17. Prueba Velocidad de Descarga con restricción de 500kbit/s 2



Fuente. Autores

Se observa que al aplicar los *rates* en determinado valor, se obtiene una velocidad limitada a aproximadamente el 10% de este *rate* configurado.

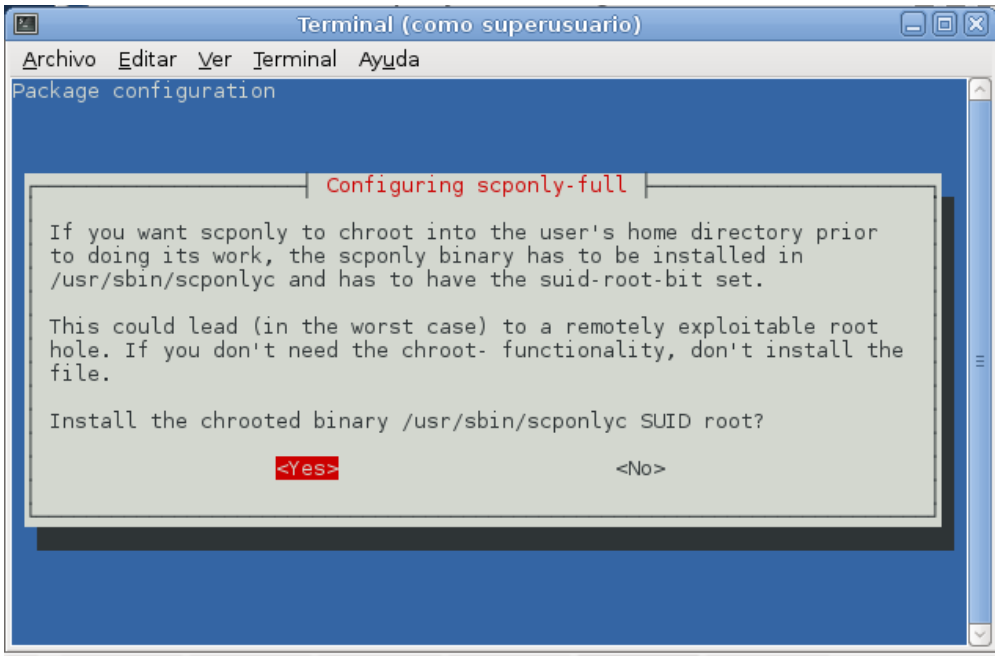
8.9 Enjaulado de la página Web

Además de las restricciones realizadas en cuanto a recursos hardware y el aislamiento del *VServer*, se requería el aislamiento de la página Web que contiene el Servidor enjaulado.

Para esto se utilizó la herramienta *scponly*, la cual actúa como una Shell que permite el acceso de los archivos en el servidor con privilegios de lectura y escritura para los usuarios remotos, sin proveer ningún otro tipo de operaciones o privilegios de ejecución.
Se realizó la instalación de *scponly-full* como se muestra a continuación:

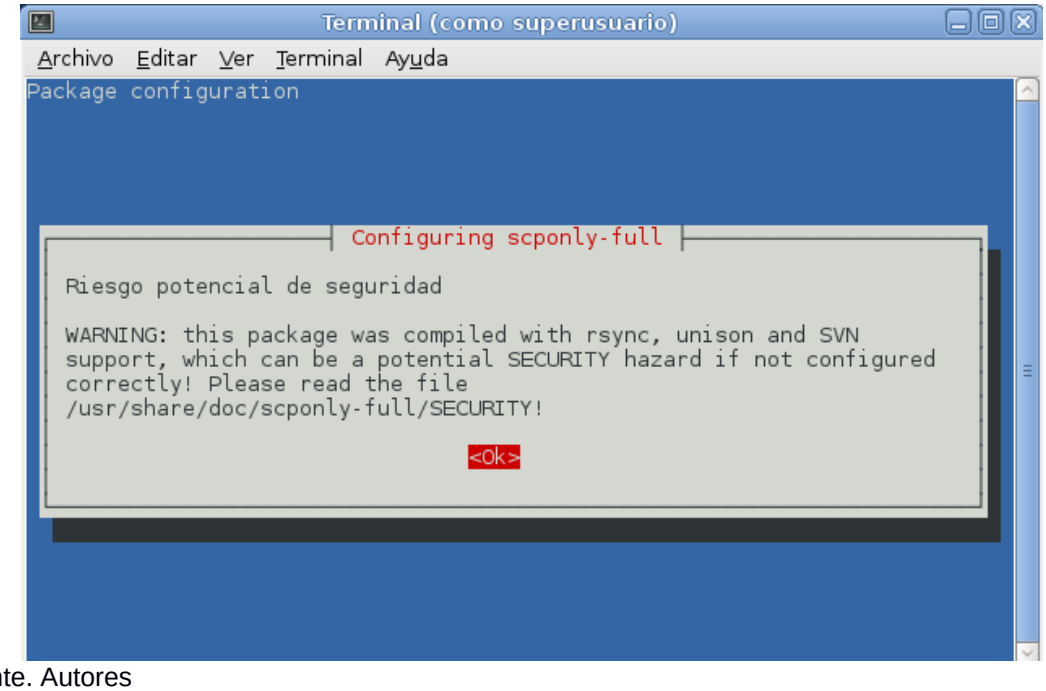
aptitude install scponly-full

Figura 18. Instalación SCPOnly 1



Fuente. Autores

Figura 19. Instalación SCPOnly 2



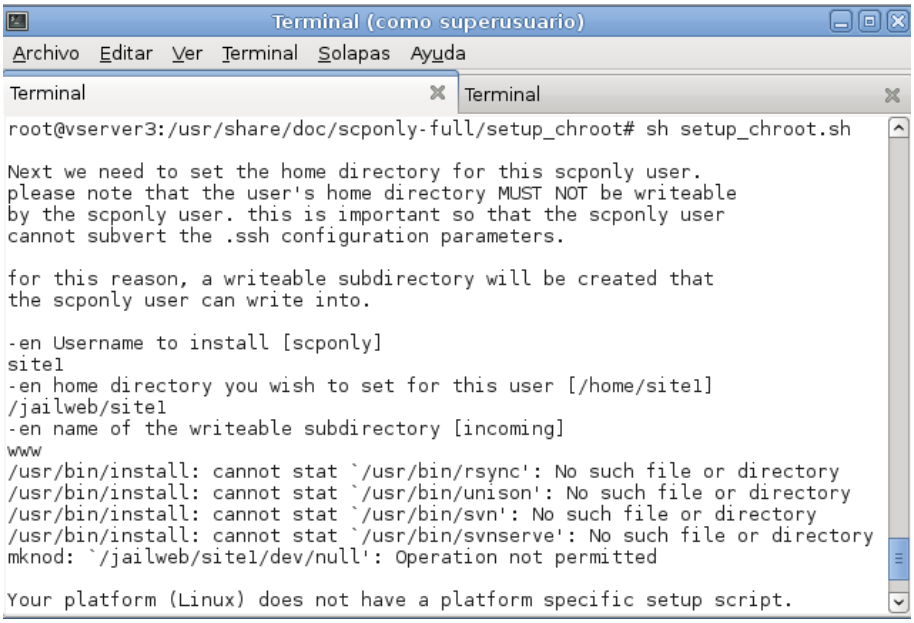
Fuente. Autores

Se realizó la configuración del ambiente web enjaulado con *scponly* de la siguiente manera: Primero se descomprimió el paquete de *chroot*:

```
root@vserver3:/# cd /usr/share/doc/scponly-full/setup_chroot/
root@vserver3:/usr/share/doc/scponly-full/setup_chroot# ls
config.h  setup_chroot.sh.gz
root@vserver3:/usr/share/doc/scponly-full/setup_chroot# gunzip setup_chroot.sh.gz
```

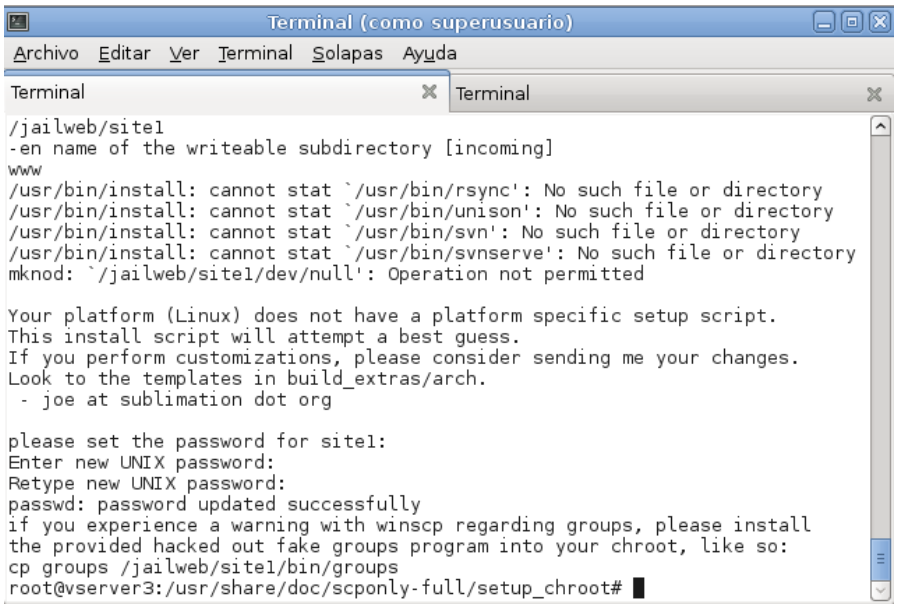
Después de esto se ejecutó el script para la creación de la jaula con los parámetros requeridos:

Figura 20. Configuración *SCPOnly 1*



Fuente. Autores

Figura 21. Configuración *SCPOnly 2*



Fuente. Autores

Al ejecutar el script solicitó los parámetros de configuración de la jaula, los cuales se insertaron de acuerdo a las necesidades:

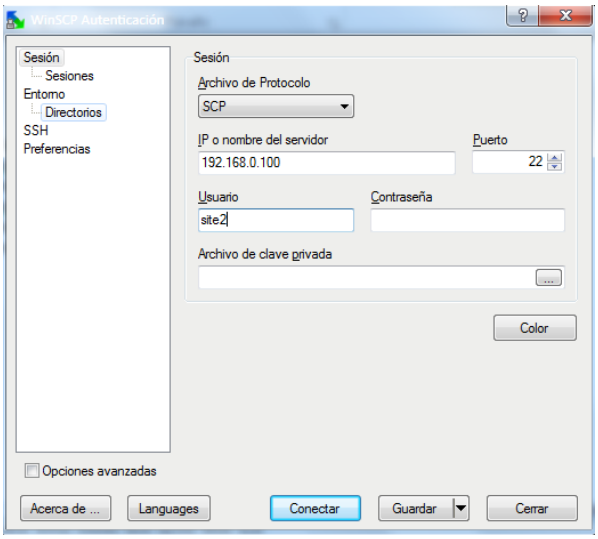
```
root@vserver3:/usr/share/doc/scponly-full/setup_chroot# sh setup_chroot.sh
Next we need to set the home directory for this scponly user.
-en Username to install [scponly]
site1
-en home directory you wish to set for this user [/home/site1]
/jailweb/site1
-en name of the writeable subdirectory [incoming]
www
/usr/bin/install: cannot stat `/usr/bin/rsync': No such file or directory
..
please set the password for site1:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
if you experience a warning with winscp regarding groups, please install
the provided hacked out fake groups program into your chroot, like so:
cp groups /jailweb/site1/bin/groups
root@vserver3:/usr/share/doc/scponly-full/setup_chroot#
```

Fue necesaria la copia de algunas librerías necesarias dentro de la jaula para permitir la conexión remota mediante *scp*, como se muestra a continuación:

```
root@vserver3:/# cp /lib/libnss_* -av /jailweb/site1/lib/
`/lib/libnss_compat-2.11.3.so' -> `/jailweb/site1/lib/libnss_compat-2.11.3.so'
removed `/jailweb/site1/lib/libnss_compat.so.2'
`/lib/libnss_compat.so.2' -> `/jailweb/site1/lib/libnss_compat.so.2'
`/lib/libnss_dns-2.11.3.so' -> `/jailweb/site1/lib/libnss_dns-2.11.3.so'
`/lib/libnss_dns.so.2' -> `/jailweb/site1/lib/libnss_dns.so.2'
`/lib/libnss_files-2.11.3.so' -> `/jailweb/site1/lib/libnss_files-2.11.3.so'
`/lib/libnss_files.so.2' -> `/jailweb/site1/lib/libnss_files.so.2'
`/lib/libnss_hesiod-2.11.3.so' -> `/jailweb/site1/lib/libnss_hesiod-2.11.3.so'
`/lib/libnss_hesiod.so.2' -> `/jailweb/site1/lib/libnss_hesiod.so.2'
`/lib/libnss_nis-2.11.3.so' -> `/jailweb/site1/lib/libnss_nis-2.11.3.so'
`/lib/libnss_nis.so.2' -> `/jailweb/site1/lib/libnss_nis.so.2'
`/lib/libnss_nisplus-2.11.3.so' -> `/jailweb/site1/lib/libnss_nisplus-2.11.3.so'
`/lib/libnss_nisplus.so.2' -> `/jailweb/site1/lib/libnss_nisplus.so.2'
root@vserver3:/#
```

Finalmente se comprobó la conexión remota por *scp* a la jaula creada dentro del *VServer*:

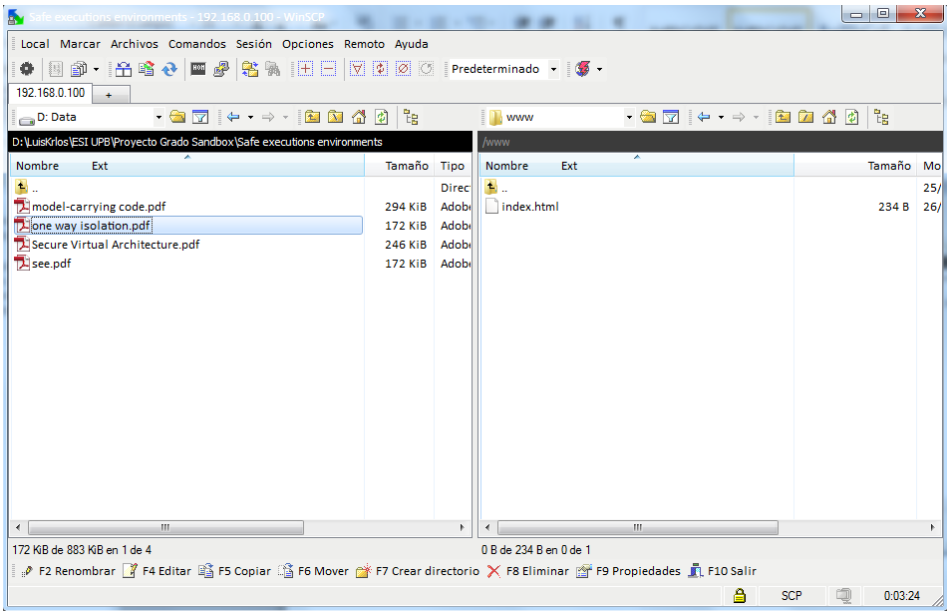
Figura 22. Acceso a la Jaula con *WinSCP* 1



Fuente. Autores

Se asignaron las credenciales de usuario para tener acceso únicamente a la jaula, y así poder modificar, cargar o eliminar los archivos de la aplicación web necesarios.

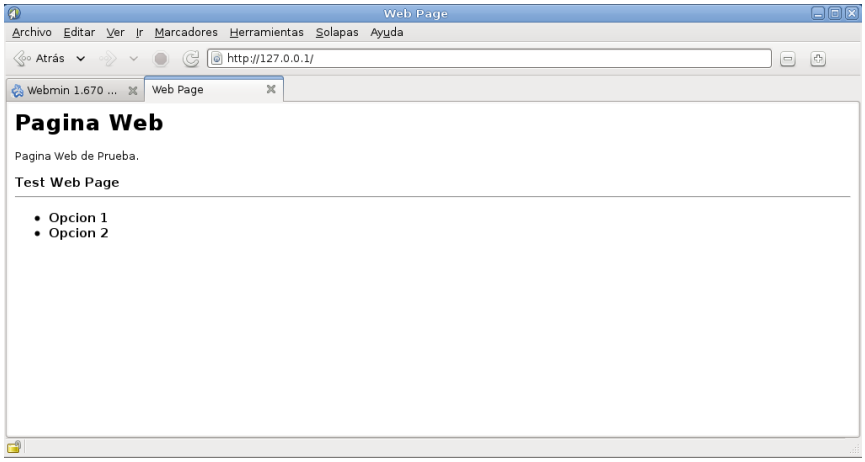
Figura 23. Acceso a la Jaula con *WinSCP* 2



Fuente. Autores

De esta forma se creó una aplicación web enjaulada, con acceso a gestión de la aplicación solamente a usuarios autorizados, y con restricciones personalizadas de recursos de espacio en disco duro, CPU, memoria y red. Con esto se logra que si la aplicación llega a contener un virus o un comportamiento inestable, no perjudique el host en el que se encuentra virtualizada.

Figura 24. Página Web Enjaulada



Fuente. Autores

9. CONCLUSIONES

Los ambientes de desarrollo de software son entornos informáticos vulnerables en gran parte debido a que en estos se encuentra software desde sus etapas iniciales de desarrollo, cuando todavía no son lo suficientemente estables y confiables para garantizar que su ejecución no afecte el sistema anfitrión ni a la infraestructura a la que puede tener acceso, por este motivo, se hace necesario implementar medidas de seguridad que permitan mitigar los riesgos asociados a ejecutar este software poco confiable.

Dependiendo de cómo esté implementada la infraestructura de red a la que pertenece el entorno de desarrollo, el software que es ejecutado allí puede tener alcance hacia otros sistemas que pueden ser críticos en los procesos de la organización, y que puede representar un gran impacto a la continuidad del negocio, por lo tanto de acuerdo a la importancia de estos sistemas y procesos, se deben implementar más controles que eviten que las vulnerabilidades se materialicen. En el DISEÑO DE UNA SOLUCIÓN SANDBOX del documento se pueden ver los controles que fueron implementados para el prototipo propuesto. En las RECOMENDACIONES del documento se describen otros posibles controles que pueden agregarse para mejorar el prototipo.

El principio de defensa en profundidad en la seguridad informática proporciona una excelente estrategia de defensa, ya que entre más medidas o “líneas” de seguridad se incorporen, mayor es el nivel de seguridad. Este proyecto busca mejorar los niveles de seguridad para ambientes de desarrollo de software, para lo cual se tienen en cuenta el principio de defensa en profundidad así como otros de los principios fundamentales y principios básicos en la seguridad de la Información. El prototipo creado genera algunos controles que mejoran el nivel de seguridad en el ambiente de intrusiones no autorizadas, disminuyen el impacto en caso de que el sistema se vea afectado en cuanto a la seguridad, y también facilita la recuperación para mantener la disponibilidad en caso de que alguna amenaza se materialice y afecte el sistema.

Los conceptos de las *Sandbox* y la virtualización que fueron aplicados en este proyecto, están relacionados debido a que los dos pueden ser utilizados para crear un marco que limita el alcance del sistema ya sea en cuanto a uso de recursos o a entorno de acción del software y los usuarios. Se implementó un modelo de seguridad por capas en donde con ayuda de la virtualización se logró obtener la primera capa de aislamiento del entorno de desarrollo y se realizó restricción de recursos hardware CPU, memoria, disco y red. Con la utilidad *scponly* se logró una segunda capa de aislamiento en donde se confina el entorno de desarrollo por medio de una jaula permitiendo el acceso solamente desde un punto raíz determinado, y se restringió el acceso solo a usuarios autorizados a los que se les ha asignado los privilegios de acceso estrictamente necesarios a esta jaula. De esta forma se creó un ambiente suficientemente confinado y funcional.

Existen diferentes herramientas que permiten verificar el uso de recursos hardware, varias de estas vienen incluidas en el sistema operativo anfitrión. Después de implementar el prototipo, utilizando herramientas de medición de recursos, y también, verificando el comportamiento y las posibilidades de acceso del ambiente confinado, se realizaron las pruebas funcionamiento, en donde se ve según los resultados, que las restricciones realizadas en cuanto a consumo de recursos, aislamiento del entorno, control de accesos, privilegios y alcance de acción tanto de usuarios como de aplicaciones, fueron efectivas.

10. RECOMENDACIONES

El sistema *Sandbox* implementado permitió conocer algunas de las medidas de seguridad que se pueden establecer en los equipos clientes y sus recursos, pero solo es una parte que pretende ser el inicio de un modelo más robusto de seguridad en donde se apliquen otras de las tantas medidas disponibles que sean aplicables al objetivo del proyecto.

Se recomienda agregar los controles de seguridad y las diferentes medidas que sean necesarias dependiendo de las necesidades específicas, con ánimo de aumentar el nivel de seguridad y en búsqueda de blindar el ambiente y perfeccionar el prototipo. Por ejemplo:

1. Se pueden aplicar medidas de control perimetrales las cuales no se tratan en este proyecto tales como:

- Sistemas de detección de intrusos (IDS) o sistemas de prevención de Intrusiones (IPS).
- Sensores de calor o de movimiento para controlar accesos a los servidores.
- Seguridad en las puertas de acceso a los servidores como lectores de tarjetas con chip, lectores de huellas digitales o de retina.
- Cámaras y/o guardias de Seguridad en las instalaciones donde se encuentran los servidores.

2. Se pueden implantar medidas de seguridad física y controles en la red dependiendo de los objetivos de la organización en donde se aplique el modelo y la criticidad del ambiente de desarrollo para el que se implemente este esquema, por ejemplo:

- Implementación de sistemas de redundancia y alta disponibilidad para la infraestructura como *mirroring* de discos, redundancia en canales de red, servidores de respaldo, centros de datos alternos, etc.
- Implementación de políticas y procedimientos para la continuidad del negocio.
- Controles de humedad, temperatura y detección de humo.
- Configuración de firewalls o políticas de *routing* en los elementos de red.
- Implementación de *Honeypots.*

A parte de las herramientas utilizadas en este trabajo, existen muchas otras herramientas útiles para el mismo fin del proyecto, tanto para plataformas Linux como en este caso, como para diferentes plataformas incluyendo plataformas Linux, Solaris, y Windows, y con las cuales se puede crear un modelo parecido al ilustrado en el documento, y que también puede ser adaptado a las organizaciones que utilicen estas diferentes plataformas.

También se podría integrar alguna herramienta, como las hay en el mercado, que permita administrar las máquinas virtuales con interfaces gráficas más ergonómicas y de manera más eficiente y cómoda para el administrador del servidor.

11. REFERENCIAS

AMC SYMPOSIUM ON OPERATING SYSTEM PRINCIPLES. (Octubre, 1973: Charlottesville, U.S.A.). The Protection of Information in Computer Systems. Charlottesville: Virginia University, 1973. 30 p. (s.f.).

(s.f.). BISHOP, Matt. Computer Security: Art and Science. U.S.A., Addison-Wesley Professional, 2002. ISBN: 0201440997.

BULAJIC, Aleksandar. SAMBASIVAM, Samuel. y STOJIC, Radoslav. An Effective Development Environment Setup for System and Application Software. En: Issues in Informing Science and Information Technology (IISIT), 2013 Volume 10. Online ISSN: 1547-5867, Print . (s.f.).

IEEE SYMPOSIUM ON SECURITY AND PRIVACY. (2009: U.S.A.). Native Cient A Sandbox for portable Untrusted X86 Native Code. U.S.A., 2009. 15 p. (s.f.).

JAKUBCO, Peter. y SLAVOMÍR, Simonák. Utilizing GPGPU in Computer Emulation: Introduction. En : Journal of Information and Organizational Sciences: JIOS. VOL 36, NO. 1 (2012); p 39-53. ISSN 1846-9418 (online) - ISSN 1846-3312 (print). (s.f.).

Microsoft. Guía de operaciones de seguridad para Windows 2000 Server. [En línea] [Citado el: 5 de Agosto de 2015.] http://www.microsoft.com/spain/technet/seguridad/2000server/chapters/ch02secops.aspx. (s.f.).

Sandboxie. Sandboxie trust no program. [En línea] [Citado el: 28 de Mayo de 2015.] http://www.Sandboxie.com. (s.f.).

Scarfone Karen, Souppaya Murugiah, Hoffman Paul. Guide to security for full virtualization technologies: Recommendations of the National Institute of Standars and Technology. En : National Institute of Standards and Technology NIST. Special Publication 80. (s.f.).

(s.f.). SCHILLER, Craig A. BINKLEY, Jim. HARLEY, David. EVRON, Gadi. BRADLEY, Tony. WILLEMS, Carsten. y CROSS, Michael. Botnets: The Killer Web App. U.S.A., Syngress Publishing, Inc., 2007. ISBN-10: 1-59749-135-7. p. 346.

(s.f.). SCHILLER, Craig A. BINKLEY, Jim. HARLEY, David. EVRON, Gadi. BRADLEY, Tony. WILLEMS, Carsten. y CROSS, Michael. Botnets: The Killer Web App. U.S.A., Syngress Publishing, Inc., 2007. ISBN-10: 1-59749-135-7. p. 346.

USENIX WINDOWS SYSTEMS SYMPOSIUM. (Agosto, 2000: New York, U.S.A.). User-level Resource-constrained Sandboxing. New York: New York University, 2000. 11 p. (s.f.).

Vmware. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. [En línea] [Citado el: 21 de Julio de 2015.] http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf. (s.f.).

Wikipedia. Operating System Level Virtualization. [Citado el: 21 de Julio de 2015.] *http://en.wikipedia.org/wiki/Operating_system%E2%80%93level_virtualization*. (s.f.).

Wikipedia. Sandbox. [En línea] [Citado el: 28 de Mayo de 2015.] *http://es.wikipedia.org/wiki/Sandbox.* (s.f.).

(s.f.). *Yu Yang. OS-level Virtualization and Its Applications. Dissertation Presented to The Graduate School in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science. Stony Brook University U.S.A. 2007. P 8-9.*

12. ANEXOS

12.1 ANEXO 1. Formato 01 – Relación básica de requerimientos

FORMATO 01 Código: 01	**RELACIÓN BÁSICA DE REQUERIMIENTOS**	Versión 1
Fecha de creación del formato: Abril 2015	Formato solicitud de creación/modificación de requerimientos	Página 1 de 1
Nombre de la dependencia		
Responsable de la dependencia		
Cargo		Ext.
E-mail del responsable de la dependencia		Firma :
Nombre del recolector		
E-mail del recolector		

Número	**Fecha recolección**	**Fuente**	**Descripción**	**Usuarios involucrados**	**Observación (Prioridad/Estado/Tipo)**

Observaciones generales:

12.2 ANEXO 2. Formato 02 – Lista de Actores

FORMATO 02 Código:02	**LISTA DE ACTORES**		Versión 1
Fecha de creación del formato: Abril 2015			Página 1 de 1
Caso de prueba especifico		Código	
Nombre del recolector			
E-mail del recolector		Fecha	

Número	Actor (Rol)	Nombre Usuario	Observación/Permisos

Observaciones generales:

12.3 ANEXO 3. Relación Básica de requerimientos del Caso Específico No. 1

FORMATO 01 Código: 01	RELACIÓN BÁSICA DE REQUERIMIENTOS	Versión 1
Fecha de creación del formato: Abril 2015	Formato solicitud de creación/modificación de requerimientos	Página 1 de 1
Nombre de la dependencia	Administración de servidores	
Responsable de la dependencia		
Cargo	Director del Proyecto - Profesor	Ext.
E-mail del responsable de la dependencia		Firma :
Nombre del recolector		
E-mail del recolector		

Número	Fecha recolección	Fuente	Descripción	Usuarios involucrados	Observación (Prioridad/Estado/Tipo)
1	22/08/2015	Director	Computador configurado como servidor anfitrión con acceso a internet	Director del proyecto	Prioridad 1
2	22/08/2015	Director	La solución informática a diseñar debe ser para ejecutarla en un Sistema Operativo GNU/Linux.	Director del proyecto	Prioridad 1
3	22/08/2015	Director	Directorio confinado, es decir, restringido solo a los usuarios autorizados (estudiantes desarrolladores de los sitios web, administrador del servidor y director del proyecto) para que estos puedan copiar sus archivos y scripts, leerlos y editarlos solo en ese directorio.	Usuario desarrollador, Administrador, director del proyecto	Prioridad 1
4	22/08/2015	Director y administrador	Que los estudiantes no puedan subir en algún nivel en jerarquía, del directorio asignado para sus sitios web, es decir, no tengan ingreso a otros directorios o espacios en el servidor, que no sean específicamente autorizados por el director del proyecto y luego modificados sus permisos por el administrador.	Usuario desarrollador, Administrador, director del proyecto	Prioridad 1
5	22/08/2015	Director y administrador	Que en caso que se requiera que ellos ejecuten algún script autorizado, se les restrinja con una Shell específica para el usuario o de alguna manera su ejecución no afecte al sistema operativo anfitrión, el administrador también puede ejecutarles los scripts, pero que no afecten o impacten en el sistema operativo anfitrión.	Usuario desarrollador, Administrador, director del proyecto	Prioridad 2
6	22/08/2015	Director	Restringir el uso de recursos de hardware (CPU, memoria RAM, espacio en Disco Duro, velocidad de descarga de datos por la red) a los desarrolladores, de acuerdo a las políticas de la administración y bajo la autorización del director del proyecto, para utilizar más eficientemente los recursos del servidor.	Director del proyecto, Administrador	Prioridad 2

Observaciones generales: Nivel de prioridad (Escala de 1 a 3, donde 1 es lo más importante y tres es de menor importancia)

12.4 ANEXO 4. Lista de Actores del Caso Específico No. 1

FORMATO 02 Código:02 Fecha de creación del formato: Abril 2015	LISTA DE ACTORES		Versión 1 Página 1 de 1
Caso de prueba especifico	No. 1 Ambiente académico	Código	01
Nombre del recolector			
E-mail del recolector		Fecha	22/08/2015

Número	Actor (Rol)	Nombre Usuario	Observación/Permisos
1	Director del Proyecto de Servidores Web	root	Todos los permisos en el servidor
2	Administrador del servidor	root	Todos los permisos en el servidor
3	Estudiante desarrollador	site1	Permisos de lectura, escritura y ejecución en el directorio ubicado en la ruta /jailweb/site1/www

Observaciones generales:
La administración de servidor es compartida entre el director del Proyecto de Servidores y el Administrador encargado.
En el directorio /www se suben y alojan los sitios web.

12.5 ANEXO 5. Relación Básica de requerimientos del Caso Específico No. 2

FORMATO 01 Código: 03	RELACION BASICA DE REQUERIMIENTOS	Versión 1
Fecha de creacion del formato: Abril 2015	Formato solicitud de creacion/modificacion de requerimientos	Página 1 de 1
Nombre de la dependencia	Administracion de servidores	
Responsable de la dependencia		
Cargo	Administrador del servidor	Ext.
E-mail del responsable de la dependencia		Firma :
Nombre del recolector		
E-mail del recolector		

Número	Fecha recolección	Fuente	Descripción	Usuarios involucrados	Observación (Prioridad/Estado/Tipo)
1	29/08/2015	Administrador	Computador configurado como servidor anfitrión con acceso a internet	Administrador	Prioridad 1
2	29/08/2015	Administrador	La solucion informatica a disenar debe ser para ejecutarla en un Sistema Operativo GNU/Linux Debian 6.	Administrador	Prioridad 1
3	29/08/2015	Administrador	Configurar un espacio en el servidor anfitrión de manera que los desarrolladores puedan ejecutar scripts o software autorizado, para probarlo y este no afecte al sistema operativo del servidor anfitrión.	Usuario desarrollador, Administrador, Jefe del departamento	Prioridad 1
4	29/08/2015	Administrador	Que los desarrolladores no puedan leer o modificar otros archivos fuera del espacio asignado para pruebas.	Usuario desarrollador, Administrador, Jefe del departamento	Prioridad 1
5	29/08/2015	Administrador	Restringir el uso de recursos de hardware (CPU, memoria RAM, espacio en Disco Duro, velocidad de descarga de datos por la red) a los desarrolladores, de acuerdo a las políticas de la administración y bajo la autorización del director del proyecto, para utilizar más eficientemente los recursos del servidor.	Administrador, Jefe del departamento	Prioridad 2

Observaciones generales: Nivel de prioridad (Escala de 1 a 3, donde 1 es lo más importante y tres es de menor importancia)

12.6 ANEXO 6. Lista de Actores del Caso Específico No. 2

FORMATO 02 Código:04 Fecha de creación del formato: Abril 2015	LISTA DE ACTORES		Versión 1 Página 1 de 1
Caso de prueba especifico	No. 2 Ambiente Empresarial	Código	02
Nombre del recolector			
E-mail del recolector		Fecha	29/08/2015

Número	Actor (Rol)	Nombre Usuario	Observación/Permisos
1	Jefe del Departamento	boss	Acceso con usuario boss: Usuario normal en el servidor Autoriza administrativamente el uso de recursos en el servidor por parte de los desarrolladores.
2	Administrador del servidor	root debian	Acceso como root: Todos los permisos en el servidor. Acceso con usuario debian: Usuario normal en el servidor Realiza la parte operativa de las configuraciones en el servidor y creación de las máquinas virtuales para pruebas.
3	Desarrolladores	nombreusuariodesarrollador	Permisos de lectura, escritura y ejecución en el directorio confinado que se le asigno /../vserverX/ Desarrollan scripts y prueban software en las máquinas virtuales.

Observaciones generales: