CIS0930SD01

APLICACIONES ANTI-FORENSES EN EL SISTEMA DE ARCHIVOS ZFS

JONATHAN CIFUENTES ARIAS

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
BOGOTÁ, D.C.
2010

CIS0930SD01

APLICACIONES ANTI-FORENSES EN EL SISTEMA DE ARCHIVOS ZFS

Autor:

JONATHAN CIFUENTES ARIAS

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TITULO DE INGENIERO DE SISTEMAS

Director

Jeimy José Cano Martínez

Jurados del Trabajo de Grado

Nelson Gomez de la Peña

José Luis Lara

Página web del Trabajo de Grado

http://pegasus.javeriana.edu.co/~CIS0930SD01/

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERIA

CARRERA DE INGENIERIA DE SISTEMAS

BOGOTÁ, D.C.

Diciembre, 2010

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS

Rector Magnífico

Joaquín Emilio Sánchez García S.J.

Decano Académico Facultad de Ingeniería

Ingeniero Francisco Javier Rebolledo Muñoz

Decano del Medio Universitario Facultad de Ingeniería

Padre Sergio Bernal Restrepo S.J.

Directora de la Carrera de Ingeniería de Sistemas

Ingeniero Luis Carlos Díaz Chaparro

Director Departamento de Ingeniería de Sistemas

Ingeniero César Julio Bustacara Medina

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTOS

La presente tesis fue posible, gracias a muchas personas que me apoyaron y me acompañaron en todo el proceso contagiándome de alegrías y ánimo. Agradezco especialmente al profesor Jeimy Cano por la dirección de este trabajo, por la confianza y el apoyo que me brindó y sobre todo por sus grandiosos consejos. A la confianza y comprensión de mis Padres y hermanos a quien se las dedico con mucho cariño y esfuerzo, por que se que cuento con ellos siempre. Agradezco también la lealtad y la paciencia de mis amigos y amigas que siempre me apoyaron y me dieron ánimos desde el comienzo del proceso. Agradezco a Dios por todas estas personas maravillosas que ha puesto en mi vida.

CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ABSTRACT

ZFS is a new file system developed by Sun Microsystems that brings benefits and functionality based on ease administration and robust design. The forensic research in this file system has already started and in this work we apply the anti-forensic methods, thinking that the solution to these problems lies in understanding the own details to this file system.

RESUMEN

ZFS es un nuevo sistema de archivos desarrollado por Sun Microsystems que trae funcionalidades y beneficios basados en una fácil administración y en un diseño robusto. La investigación forense en este sistema de archivos ya ha comenzando y en este trabajo vamos a aplicar los métodos anti-forenses, pensando que la solución a estos problemas se encuentra en la comprensión de los detalles propios de este sistema de archivos.

RESUMEN EJECUTIVO

Los sistemas de archivos proporcionan un mecanismo para que los usuarios almacenen sus datos a largo plazo. Razón por la cual los investigadores o profesionales de la informática forense estudian detalladamente los sistemas de archivos con el fin de recolectar y analizar evidencia digital que se pueda aportar a un determinado proceso. Sin embargo y a medida que pasa el tiempo, cualquier persona decidida a comprometer la disponibilidad o utilidad de la evidencia digital, podría invalidar con herramientas especializadas que pueden alterar, destruir, ocultar o inhabilitar la generación de los rastros en la máquina.

Solaris 10 es la versión más reciente del sistema operativo desarrollado por Sun Microsystems, el cual incluye nativamente el nuevo sistema de archivos ZFS. Este sistema de archivos fue pensado y diseñado para cambiar la manera en que se administraban los sistemas de archivos. La investigación forense en este sistema de archivos ya ha comenzando y en este trabajo vamos a aplicar los métodos anti-forenses, pensando que la solución a estos problemas se encuentra en la comprensión de los detalles propios de este sistema de archivos.

Las técnicas anti-forenses entre sistemas de archivos suelen diferir en herramientas y métodos, pero siempre conservan los mismos objetivos y principios. Si lo anterior es correcto, intentar aplicar estos objetivos y principios será un nuevo desafío para la investigación forense en ZFS.

Luego, esta investigación se organiza primero en estudiar el sistema de archivos ZFS teniendo en cuanta las limitadas fuentes de información, conociendo las estructuras de datos y los métodos de almacenamiento para poder identificar donde se aloja la evidencia, para luego analizar como manipular, confundir o eliminar estas evidencias concibiendo un modelo de técnicas anti-forenses para ZFS, detallando el contexto general de las pruebas a realizar y cómo se hicieron en la práctica sobre el sistema de archivos.

INTRODUCCIÓN

En tiempos donde tantas cosas parecen disminuir en respuesta a la crisis económica mundial, el "universo digital" continúa con su desbordante crecimiento. La gente continúa tomando fotos, enviando e-mail, escribiendo en blogs, posteando videos, etc. Una investigación de IDC patrocinada por EMC, denominada: "The Diverse and Exploding Digital Universe: An Updated Forecast of Worldwide Information Growth Through 2011" [1], revela que el universo digital en el año 2007 ha sido de 281 .000 millones de gigabytes (281 exabytes) o lo que es equivalente a casi 45 gigabytes de información por cada persona del planeta, con una tasa de crecimiento anual del 60%, es decir que el universo digital alcanzaría los 1.8 zettabytes (1.800 exabytes) para el año 2011.

En resumen, el crecimiento del universo digital se ha multiplicado por 10 en cinco años. Por estas razones Joe Tucci, Presidente Ejecutivo y CEO de EMC comentaba que: "Las organizaciones necesitan planificar, en función de las oportunidades limitadas, nuevas formas de utilizar la información y cómo encarar los desafíos de administración de la información".

Es decir que la innovación en las tecnologías de la información dentro de las organizaciones les permita adquirir ventajas competitivas notables en la manera como utilizan y administran la información. Ahora bien, revisemos la definición de tecnología de la información por Bakopoulos (1985): "El conjunto de recursos materiales que se utilizan para el almacenamiento, procesamiento y comunicación de la información, y la manera en la cual estos recursos son organizados, dentro de un sistema, para desarrollar eficientemente el conjunto de tareas asignadas." Podemos apreciar claramente que las TI encierran también los componentes físicos que a su vez son controlados por el software.

Por ejemplo podemos encontrar tecnologías en el contexto del almacenamiento de la información como los LVM[1], herramienta que ofrece flexibilidad en la gestión y permite atender a las necesidades cambiantes, solucionando problemas en los servidores como por ejemplo al instalar nuevos discos, realizar backups, agrupación del almacenamiento físico, redimensionamiento de volúmenes lógicos, la migración de datos o la restauración de datos.

Sin embargo, los LVM pueden ser difíciles de administrar y no satisfacen las necesidades cambiantes en el contexto de la escalabilidad; adicionar o remover dispositivos al sistema no es tan sencillo y el límite

[1] Significa gestor de volúmenes lógicos (Logical Volume Manager). Es un método de localización del espacio disco duro en volúmenes lógicos que pueden ser fácilmente redimensionados en vez de particiones [53].

finito en la capacidad de almacenamiento en los sistemas de archivos de 64 bits será también un factor límite algún día [2].

Otras tecnologías en el contexto de la seguridad de la información, específicamente en la integridad de los datos como los RAID[2], que también permiten atender las necesidades cambiantes, ofreciéndonos mayor integridad, mayor tolerancia a fallos, mayor rendimiento y mayor capacidad; sin embargo las implementaciones RAID no previenen la corrupción silenciosa de los datos ni las vulnerabilidades "write-hole", los RAID no calculan ni verifican la suma de verificación (checksums) de un archivo, por lo que no pueden detectar errores cuando un archivo es leído o escrito y mucho menos subsanarlos. Estos mecanismos son necesarios para validar la integridad de los datos [2].

Por estas razones y otras, observamos que existen retos que enfrentar, nuevas formas de utilizar la información y cómo encarar los desafíos de administración de la información, que llevará a las organizaciones por el camino de la innovación para repensar sus ventajas competitivas.

Hoy por hoy Sun Microsystems se está preparando para el futuro, innovando con el nuevo sistema operativo Solaris 10 que incluye nativamente el sistema de archivos ZFS. Sistema de archivo que será objeto de estudio en este trabajo. ZFS es un sistema de archivos que fundamentalmente cambia la manera en que se administraban los sistemas de archivos, con funcionalidades y beneficios que no se pueden encontrar en otros sistemas en la actualidad. ZFS nos proporcionará básicamente una administración simple, semántica transaccional, integridad de los datos, y escalabilidad. Además ZFS no es una mejora incremental de la tecnología existente; es fundamentalmente un nuevo acercamiento a la administración y gestión de datos [3] .

Una vez entendidas las implicaciones del sistema de archivos ZFS en el universo digital y las TI dentro de las organizaciones; retomamos las palabras de Joe Tucci: "A medida que las huellas digitales de las personas continúen creciendo, también se incrementará la responsabilidad de las organizaciones respecto a la privacidad, disponibilidad y confiabilidad de esa información. La carga recae sobre los departamentos de IT dentro de las organizaciones que deberán manejar los riesgos asociados y el cumplimiento de regulaciones en lo referente al mal uso de la información, fuga de datos y protección contra fallas de seguridad." [4]

[2] Significa matriz redundante de discos independientes (Redundant Array of Independent Disks). Es un método de combinación de varios discos duros para formar una única unidad lógica en la que se almacenan los datos de forma redundante. Ofrece mayor tolerancia a fallos y altos niveles de rendimiento que un sólo disco duro o un grupo de discos duros independientes [54].

Cabe complementar que los individuos participan activamente en la generación del universo digital dejando una "huella digital" (información que usted mismo crea: como usuarios de Internet y de redes sociales) y una "sombra digital" (información acerca de usted: como nombres en registros financieros, búsquedas en la Web o imágenes suyas obtenidas por cámaras de seguridad) un nuevo fenómeno, que según la investigación, por primera vez la "sombra digital" es más grande que su "huella digital" [1].

En otras palabras, debido al inminente crecimiento de la información, la responsabilidad de las organizaciones respecto a la seguridad de la información también debería aumentar; y de acuerdo con Joe Tucci, se deberían analizar y administrar los riesgos, cumpliendo con mejores medidas de control para minimizar lo referente al mal uso de la información, fuga de datos y protección contra fallas de seguridad.

Además, con el incremento de los delitos informáticos presentados en todo el mundo, gran cantidad de países se han visto obligados a tener en cuenta este concepto en sus legislaciones y a reglamentar la admisión de la evidencia digital en una corte [5]. Lo que hace evidente una prevención y detección de los delitos informáticos. No obstante hay que tener en cuenta que la seguridad total no es posible, pues no existe ningún elemento que no esté expuesto a situaciones no controladas o inesperadas; por lo que estrategias como el análisis y administración de riesgos nos permitirá reconocer la presencia de situaciones no previstas e identificar los posibles controles que mitiguen los mismos [6].

Luego, aquellos que pretendan investigar incidentes por medio de la evidencia respondiendo preguntas como: qué, quién, porqué, dónde, cuándo y cómo [6], deberían tener en cuenta las situaciones no previstas, como la habilidad de los intrusos en distorsionar la evidencia mediante lo que llamamos: técnicas anti-forenses.

1 DESCRIPCIÓN GENERAL DEL TRABAJO DE GRADO

1.1 FORMULACIÓN

Cualquier ordenador debe tener instalado un sistema operativo, permitiendo así las diferentes interacciones entre el usuario y la máquina. Este sistema operativo es instalado en un dispositivo de almacenamiento secundario, el cual tiene como función principal registrar los datos e información. Para lograr este objetivo es necesario concebir una estructura que nos permita representar los datos. Esta estructura se denomina sistema de archivo y varía dependiendo del sistema operativo, por ejemplo: Windows usa el sistema de archivo NTFS, Linux usa EXT3, MacOS usa HFS, Unix usa UFS y Solaris recientemente usa ZFS [7].

Estos sistemas de archivo son de gran importancia en un proceso forense ya que la mayoría de la evidencia digital se encuentra registrada en estos. En este sentido, los investigadores o profesionales de la informática forense deben conocer las estructuras de datos y los métodos de almacenamiento que utilizan los diferentes sistemas de archivos para identificar los procedimientos y técnicas [8] necesarias para extraer datos y obtener evidencia válida para una determinada investigación forense.

Parte de este trabajo se enfocará en estudiar el nuevo sistema de archivos ZFS (Zettabyte File System) desarrollado por Sun Microsystems, que representa una revolución en el diseño de sistemas de archivos, con el ánimo de conocer las estructuras de datos y los métodos de almacenamiento para lograr extraer datos y obtener evidencia.

Sin embargo, la constante habilidad de los intrusos por manipular, confundir o eliminar las evidencias relacionadas con un incidente; desconcierta a los profesionales de la seguridad al ver como estos son capaces de alterar o desafiar el funcionamiento de sus herramientas, dejando entre dicho las respuestas a los interrogantes: qué, quién, porqué, dónde, cuándo y cómo. Esta realidad nos muestra que la inseguridad de la información se matiza en habilidades que llamaremos anti-forenses [6].

El propósito de este trabajo no estará en identificar los procedimientos y técnicas necesarias para obtener evidencia digital, sino en los métodos anti-forenses, lo que nos permitirá darle respuesta a la pregunta de investigación: *¿Cómo se materializan las técnicas anti-forenses en el nuevo sistema de archivos ZFS?*.

1.2 JUSTIFICACIÓN

Lo anti-forense es más que tecnología, es un enfoque al hacking criminal que puede resumirse así*: "Haz difícil que te encuentren e imposible de probar que te encontraron"* [9].

Luego la seguridad total no es posible, pues no existe ningún elemento que no esté expuesto a situaciones no controladas o inesperadas, por lo que descubrir aquellas vulnerabilidades nos proveerán elementos tangibles y reales que nos dicen que la seguridad de la información, concepto intangible, requiere de la inseguridad para tener sentido y desarrollar mejores estrategias y prácticas para incrementar los niveles de seguridad [6].

Por lo tanto la solución a estos problemas se encuentra en la comprensión de los detalles propios del sistema de archivos y en el análisis de la realidad anti-forense, con el ánimo de determinar estrategias mitigadoras [10], mediante el análisis y administración de riesgos, proceso que reconoce la presencia de situaciones no previstas y trata de identificar los posibles controles que mitiguen los mismos [6].

A lo mejor se está persiguiendo el camino equivocado, estamos colocando mucho énfasis en las tecnologías forenses e ignorando la necesidad de capacitación de las personas y el desarrollo de procesos. Tal vez necesitemos tomar tiempo para replantear nuestra visión forense y crear nuevas formas de atacar el problema de raíz [10].

Razón por la cual el inicio del análisis de los métodos anti-forense en un sistema de archivo relativamente nuevo y diferente a los tradicionales, es un cambio en la visión forense.

Sin embargo y desafortunadamente no podemos conseguir el completo control del problema anti-forense y nunca podremos evitar la corrupción de la evidencia [11]. En este sentido, si atacamos los problemas uno por uno, seremos capaces de minimizar la susceptibilidad de lo anti-forense [10] es decir:

> *"Reconocer que tenemos que aprender todos los días; es abrir la mente a nuevas posibilidades; es mirar en la inseguridad de la información la fuente de las propuestas creativas para mejorar la gestión de la seguridad"; "Renunciar a aprender de la inseguridad, es "enterrar" el futuro de la seguridad"* [6].

Por estas razones, este trabajo desea aprender de la inseguridad utilizando a la computación anti-forense como estrategia técnica para avanzar en la generación de investigaciones, estrategias y procedimientos

forenses más confiables en el sistema de archivos ZFS, que de no ser acogido en el mercado en los próximos años, se preverá que la próxima generación de sistemas de archivos, le apostarán al almacenamiento, a la seguridad y a la administración de la información de maneras similares como el caso de Btrfs[3]. Cabe mencionar que actualmente se desconoce el impacto que pueda tener la adquisición de Sun Microsystems por parte de Oracle sobre ZFS.

Finalizando, los responsables de la seguridad no solo deberían conocer el sistema, sino también su inseguridad; descubriendo fallas y reconociendo vulnerabilidades, para que a largo plazo se puedan aumentar los niveles de seguridad de las organizaciones.

1.3 OBJETIVO GENERAL

Aplicar los métodos anti-forenses sobre el sistema de archivos ZFS desarrollado por Sun Microsystems, que comprometan la disponibilidad o la utilidad de la evidencia digital sobre éste.

1.4 OBJETIVOS ESPECÍFICOS

- Estudiar y analizar el estado del arte del sistema de archivos ZFS, las consideraciones forenses actuales sobre éste y la realidad anti-forense existente.
- Plantear un modelo de aplicación de técnicas anti-forenses para ZFS, detallando el tipo de pruebas y donde se aplican en dicho sistema de archivos.
- Aplicar el modelo propuesto sobre el sistema de archivos ZFS.

[3] Sistema de archivos con un modelo COW (copia por escritura), anunciado por Oracle Corporation para Linux.

2 REVISIÓN DE LITERATURA

2.1 SISTEMA DE ARCHIVOS ZFS

ZFS (Zettabyte File System) ha sido diseñado e integrado desde cero para satisfacer las necesidades que no han podido suplir los sistemas de archivos tradicionales. Cualquier persona ha sufrido problemas como: borrado de archivos importantes, quedarse sin espacio en una partición, semanas agregando nuevo almacenamiento a servidores, intentando aumentar o reducir el tamaño de un sistema de archivos y la corrupción de datos; por lo tanto, existe mucho campo a la hora de mejorar las funcionalidades y administración de un sistema de archivos [12].

ZFS es un avance en la administración de datos enfocándose en la integridad de estos, en la fácil administración y en la integración del sistema de archivos con la capacidad de administrar volúmenes. El concepto central de la arquitectura, es el grupo de almacenamiento virtual ("*virtual storage pool*"), que permitiría desacoplar el sistema de archivos del almacenamiento físico, de la misma manera que lo hace la memoria virtual con la memoria física, permitiendo así mejor eficiencia en los dispositivos de almacenamiento. Es decir que ZFS logra compartir espacio dinámicamente entre múltiples sistemas de archivos mediante un grupo de almacenamiento ("*storage pool*"). Adicionalmente se permite agregar o remover dinámicamente almacenamiento físico a los grupos de almacenamiento sin necesidad de interrumpir servicios, proveyendo nuevos niveles de flexibilidad, disponibilidad y desempeño [12].

2.1.1 FUNCIONALIDADES

ZFS es un sistema de archivos novedoso con funcionalidades y benéficos que no encontrará en otros sistemas de archivos, con énfasis en su fácil administración y diseño robusto.

2.1.1.1 GRUPOS DE ALMACENAMIENTO (STORAGE POOLS)

Para manejar el almacenamiento físico, ZFS utiliza el concepto de grupos de almacenamiento (storage pool), donde podemos tener múltiples dispositivos, a diferencia de los sistemas de archivos comunes que se enfocaban sobre un simple dispositivo (ver Ilustración 12), por lo que necesitaban un gestor de volúmenes (por ejemplo: LVM) que permite tener una vista de alto nivel de los discos de almacenamiento y así poder sacar provecho a múltiples dispositivos. ZFS elimina la necesidad de un gestor de volúmenes

ya que primero agrega los dispositivos al espacio de almacenamiento y después se crean volúmenes virtuales [13].

El sistema de archivos no se enfoca en dispositivos individuales, permite que estos compartan espacio con todos los sistemas de archivos en el espacio de almacenamiento. Por lo que ya no es necesario predeterminar el tamaño de un sistema de archivos ya que el espacio de almacenamiento de ZFS actúa como una memoria virtual, es decir que cuando se agrega nuevo almacenamiento a ZFS, todos los sistemas de archivos en el espacio de almacenamiento pueden usar el espacio adicional sin mayor trabajo [13]. En otras palabras, si necesitamos más espacio, adicionamos más discos y automáticamente en tiempo de ejecución el espacio es disponible para todos los sistemas de archivos sin necesidad de aumentar o disminuir manualmente estos. Ya no es necesario crear particiones, lo podemos abstraer como una sola partición con cientos de discos, por lo que herramientas como *fdisk, newfs, tunefs, fsck* se pueden olvidar [14].

2.1.1.2 SEMÁNTICA TRANSACCIONAL

Para garantizar que el sistema de archivos está funcionando de manera estable y fiable, se debe revisar que se encuentre en un estado coherente. Desafortunadamente no es fácil garantizar la coherencia en el caso de una falla de energía, porque la mayoría de la operaciones de los sistemas archivos no son atómicas [14], es decir que las transacciones finalicen de forma correcta o incorrecta, como unidad completa, pero no pueden acabar en un estado intermedio.

ZFS es un sistema de archivos que siempre es coherente en disco, sin necesidad de usar *fsck* o *journaling*, mejorando el rendimiento y garantizando que en caso de un corte de electricidad, la estructura del sistema de archivos sea siempre válida.

Para este fin, ZFS administra los datos mediante la semántica copia por escritura (Copy On Write – COW; ver **¡Error! No se encuentra el origen de la referencia.**). Lo que significa que los datos nunca se sobrescriben y ninguna secuencia de operaciones se compromete. Es decir que siempre se escribe en áreas libres lo que asegura que los datos se guarden confiablemente, estableciendo un puntero a los bloques del padre, es decir que los punteros de los bloques nunca apuntan a bloques inconsistentes [14].

2.1.1.3 SUMAS DE COMPROBACION Y AUTO- REPARACIÓN DE DATOS

En ZFS se efectúa una suma de comprobación de todos los datos y metadatos para garantizar la integridad de los datos mediante un algoritmo seleccionable por el usuario (fletcher2, fletcher4 o SHA256); los

metadatos siempre usan SHA256. Las sumas de comprobación se almacenan en los metadatos de los datos y permiten detectar la corrupción silenciosa de los datos causada por cualquier defecto en disco, en controladores, cables, drivers o firmware [14].

Además, ZFS ofrece soluciones para la auto-reparación de datos. ZFS permite distintos niveles de redundancia de datos en los grupos de almacenamiento (storage pool), incluida la duplicación (RAID-1) y una variación de RAID-5 (RAID-Z). Si se detecta un bloque de datos incorrectos, ZFS recupera los datos correctos de otra copia redundante y los repara sustituyendo los datos incorrectos (ver Ilustración 13). La recuperación de datos y las sumas de comprobación se efectúan en la capa del sistema de archivos y es transparente para las aplicaciones [13].

2.1.1.4 ESCALABILIDAD

ZFS acrónimo del término en inglés "Zettabyte File System", es el primer sistema de archivos de 128 bits y el nombre hace referencia al hecho de que puede almacenar 256 cuatrillones de zettabytes [13], un valor interesante si tenemos en cuenta que el universo digital alcanzara los 1.8 zettabytes para el año 2011 [1].

Jeff Bonwick, que conduce el equipo que desarrollo ZFS, explica la razón de manera detallada por la que 128 bits son suficientes en su blog [17]; "los sistemas de archivos han demostrado tener una duración mucho mayor que la mayoría del software tradicional, debido en parte al hecho de que es extremadamente difícil cambiar el formato en disco. Teniendo en cuenta el hecho de que UFS (Unix File System) ha perdurado en su forma actual durante casi 20 años, no es una locura esperar que ZFS dure como mínimo 30 años en el futuro. En este momento, se aplica la ley de Moore en lo que respecta al almacenamiento y empezamos a predecir que se almacenarán más de 64 bits de datos en un sistema de archivos único" [18].

2.1.1.5 INSTANTÁNEAS, CLONES Y "*DITTO BLOCKS*"

Una instantánea o "*snapshot''* es una copia de sólo lectura en un instante de tiempo de un sistema de archivos o de un volumen. Estas se crean rápido y fácilmente. Inicialmente, las instantáneas no consumen espacio adicional en el pool [13].

Las instantáneas incluyen metadatos y datos de manera automática y transparente para el usuario. Estas se logran usando el modelo COW (copia por escritura), ya que cuando nuevos datos sean almacenados, no se libera el espacio de los datos antiguos, lo que implica menos trabajo para crear las instantáneas [14].

Un clon se crea a partir de una instantánea de tipo lectura/escritura, lo que posibilita volver a un estado previo sin las modificaciones que se hicieron después de la creación de la instantánea [14].

Los "*Ditto blocks*" son copias automáticas de metadatos y de datos con el objetivo de establecer políticas de protección de datos. Por ejemplo puedo ajustar las políticas para obtener mayor protección en mis archivos personales que en archivos del sistema que puedo recuperar desde el DVD de instalación [19]. Los "*Ditto blocks*" se comprenderán mejor en la Tabla 8 con los DVAs (Data Virtual Address).

2.1.1.6 ADMINISTRACIÓN SIMPLIFICADA

Uno de los aspectos más destacados de ZFS es su modelo de administración simplificado. Mediante un sistema de archivos con distribución jerárquica, herencia de propiedades y administración automática de puntos de montaje y semántica share de NFS, ZFS facilita la creación y administración de sistemas de archivos sin tener que usar varios comandos ni editar archivos de configuración. Con un solo comando puede establecer fácilmente cuotas o reservas, activar o desactivar la compresión o administrar puntos de montaje para diversos sistemas de archivos. Los dispositivos se pueden examinar o reparar sin tener que utilizar un conjunto independiente de comandos de VolumeManager. Puede crear un número ilimitado de instantáneas de los sistemas de archivos. Puede hacer copias de seguridad y restaurar archivos de sistemas concretos [13].

Los sistemas de archivos en ZFS son muy sencillos (equivalen a un nuevo directorio), de manera que se recomienda crear un sistema de archivos para cada usuario, proyecto, espacio de trabajo, etc. Este diseño permite definir los puntos de administración de forma detallada [13].

2.1.2 ARQUITECTURA

2.1.2.1 DISPOSITIVOS VIRTUALES

Los grupos de almacenamiento poseen una colección de dispositivos virtuales (*vdevs*). Estos dispositivos virtuales pueden ser [20]:

- Dispositivos virtuales físicos: Son bloques escribibles en un dispositivo, como por ejemplo un disco.
- Dispositivos virtuales lógicos: Es un modelo para agrupar Dispositivos virtuales físicos.

Los dispositivos virtuales tienen la estructura de un árbol, donde los físicos son las hojas del árbol. La raíz del árbol es el "*root vdev*" que se encuentra por cada grupo de almacenamiento [20].

La Ilustración 13 nos muestra un árbol de dispositivos virtuales con dos espejos. El primer espejo M1 contiene el disco A y el disco B, el segundo espejo M2 contiene el disco C y el disco D. Por lo tanto los dispositivos virtuales físicos son: A, B, C y D; los dispositivos virtuales lógicos: M1 y M2, que son agrupados por el dispositivo virtual raíz, denominado: "*root vdev*" [20].

2.1.2.2 ETIQUETAS DE UN DISPOSITIVO VIRTUAL (VDEV LABELS)

Los *"vdev labels"* o etiquetas de un dispositivo virtual, son una estructura de datos de 256KB por cada uno de los dispositivos virtuales físicos en el grupo de almacenamiento. Estas etiquetas contienen información que describe a los dispositivos virtuales físicos y a sus lógicos asociados. Por ejemplo en la Ilustración 13, la estructura de la etiqueta o el "*vdev label"* del dispositivo virtual físico C, poseerá información de los siguientes dispositivos virtuales: C, D y M2 [20].

Los *"vdev labels"* cumplen con dos propósitos: proveer acceso al contenido de los grupos de almacenamiento y verificar la integridad y disponibilidad de estos. Para cumplir con esto se utiliza un modelo de redundancia y actualización por etapas, que se describirán a continuación [20].

2.1.2.2.1 REDUNDANCIA

Para proporcionar la redundancia, se realizan cuatro copias idénticas del "*vdev label*" cada una de 256KB y para cada uno de los dispositivos virtuales físicos que se encuentran en el grupo de almacenamiento. Estas cuatro copias se almacenan en su respectivo dispositivo virtual, por lo que no serán idénticas a las copias de otros dispositivos virtuales en el grupo de almacenamiento. Por ejemplo si se agrega un dispositivo de tamaño N al grupo de almacenamiento, ZFS colocará dos etiquetas (L0 y L1) al frente del dispositivo y dos etiquetas (L2 y L3) atrás del mismo (ver Ilustración 1). Esta manera de almacenar las copias, es debida a que una corrupción de datos o una sobrescritura del disco, ocurre en fragmentos contiguos del disco, por lo que la probabilidad de que alguna de estas copias sea accesible es alta [20].

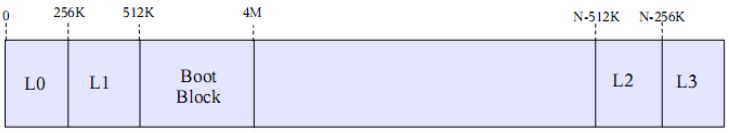

ILUSTRACIÓN 1. DISTRIBUCIÓN DE LAS ETIQUETAS Y EL BOOT BLOCK EN UN DISPOSITIVO DE TAMAÑO N. TOMADO DE [20]

Después de las etiquetas L0 y L1 se encuentra un espacio de 3.5MB denominado "*Boot Block*" que es reservado para un uso futuro [20].

2.1.2.2.2 ACTUALIZACIONES TRANSACCIONALES DE DOS ETAPAS

La localización de las etiquetas de un dispositivo virtual (*vdev labels*), se establece cada vez que un dispositivo es agregado al grupo de almacenamiento. Cuando una de las etiquetas de un dispositivo virtual es actualizada, se sobrescribe el contenido de la etiqueta, ya que las etiquetas no utilizan la semántica COW (copia por escritura). Sin embargo, para evitar errores y que ZFS siempre sea capaz de acceder a las etiquetas, es necesario usar etapas. La primera etapa de la actualización escribe en las etiquetas L0 y L2 (ver Ilustración 1). Si falla algo en esta actualización las etiquetas L1 y L3 siguen siendo validas, por lo que se actualizan y se escriben en disco. De esta manera siempre existirá una copia válida en cualquier momento [20].

2.1.2.3 DETALLES DE UN DISPOSITIVO VIRTUAL

El contenido de una etiqueta en un dispositivo virtual consta de cuatro partes [20]:

- Espacio en blanco (8KB)
- Encabezados sobre información del boot (8KB)
- Pares de Nombre-Valor (112KB)
- Arreglo de "*uberblock*" (128KB, cada uno de 1K)

Estas partes se distribuyen en la etiqueta de un dispositivo virtual como se muestra en detalle en la Ilustración 2. A continuación se revisaran cada una de las partes.

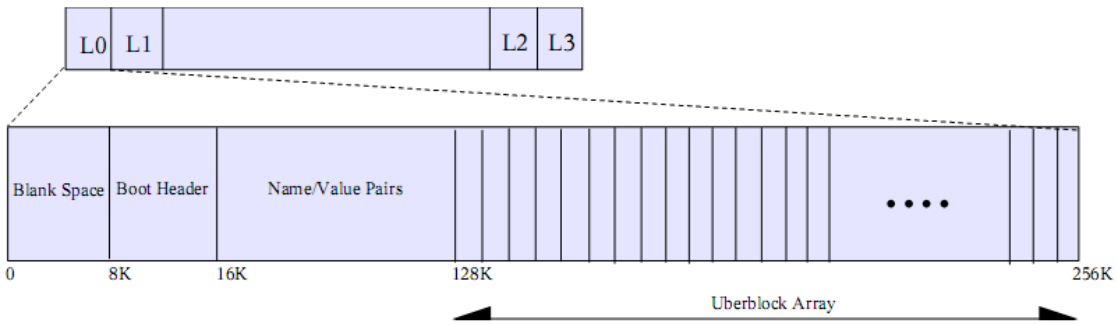

ILUSTRACIÓN 2. COMPONENTES DE LA ETIQUETA DE UN DISPOSITIVO VIRTUAL. TOMADO DE [20]

2.1.2.3.1 *"BLANK SPACE"*

ZFS soporta etiquetas de disco VTOC (tabla de contenidos del volumen) y EFI, como métodos validos para describir el diseño del disco. Las etiquetas EFI tienen su propio espacio reservado, las VTOC necesitan 8K al comienzo del disco, para soportar esto se dejan 8K vacios para prevenir una sobrescritura de la etiqueta [20].

2.1.2.3.2 ENCABEZADOS SOBRE INFORMACIÓN DEL BOOT ("*BOOT HEADER*")

Los siguientes 8K se reservan para un uso futuro [20].

2.1.2.3.3 PARES DE NOMBRE-VALOR ("*NAME/VALUE PAIRS*")

Los siguientes 112KB son una colección de pares nombre-valor, que describen el dispositivo virtual y todos sus dispositivos virtuales asociados al subárbol que los contiene menos el dispositivo virtual raíz (*root vdev*). Por ejemplo la etiqueta del dispositivo virtual "A" (ver Ilustración 13) contendrá información que describirá los dispositivos virtuales: "A", "B" y "M1" [20]. Toda esta información se almacena en nvlists[4] con codificación XDR [20].

2.1.2.3.4 ARREGLO UBERBLOCK ("*UBERBLOCK ARRAY*")

El arreglo "*uberblock"* contiene información necesaria para acceder al contenido del pool. Solo un *uberblock* es activo en cualquier instante de tiempo. Este será el que tiene el número de grupo de transacción más alto y una suma de verificación válida (SHA-256). El *uberblock* activo nunca es sobrescrito, para asegurar el constate acceso a esté. Para esto cualquier actualización de un *uberblock*, se realiza siguiendo el modelo COW (copia por escritura), escribiendo en otro *uberblock* del arreglo incrementando el número del grupo de transacción, convirtiéndolo en el *uberblock* activo [2] [20]. El contenido de un uberblock lo podemos revisar en la Ilustración 14 y sus detalles en laTabla 7.

2.1.2.4 PUNTEROS DE BLOQUE (*BLKPTR_T*)

Para transferir datos entre el disco y la memoria principal se utilizan unidades llamadas bloques. Los punteros de bloque (*Block Pointers*) son una estructura de 128 bytes que describe, localiza y verifica los

[4] Refiere a la clase "NVList", que básicamente es una lista modificable que contiene objetos "NamedValue". Cada objeto de estos tiene un nombre de tipo String, un valor de cualquier tipo y una bandera [55].

datos en disco [20]. En otras palabras son punteros a bloques de datos. El diseño de esta estructura (*blkptr_t*) la vemos en la Ilustración 16 con sus respectivos detalles en la Tabla 8.

2.1.2.5 UNIDAD GESTORA DE DATOS (DMU)

Los bloques y grupos de bloques llegan a la unidad gestora de datos (*Data Management Unit* - DMU) en unidades llamadas objetos, para que esta los organice en agrupaciones de objetos relacionados [20].

2.1.2.5.1 OBJETOS

En ZFS todo se trata como objetos. Los bloques y grupos de bloques conforman objetos. Los diferentes tipos de objetos que pueden existir son los siguientes:

Los objetos se definen con una estructura de datos (*dnode_phys_t*) de 512 bytes llamada "*dnode*". Esta se encarga de organizar y describir un grupo de bloques para conformar un objeto [20].

En otras palabras la estructura son metadatos usados para describir todos los objetos en ZFS. Esta estructura posee unos campos de longitud fija y otros campos de longitud variable ya que estos últimos son arreglos con tamaños variables. (Ilustración 17). El detalle de la estructura *dnode* la podemos apreciar en la Tabla 10.

2.1.2.5.2 GRUPOS DE OBJETOS

La unidad gestora de datos (DMU) organiza los objetos en grupos de objetos relacionados ("*object sets*") para formar cuatro tipos: las instantáneas, los clones, los volúmenes y los sistemas de archivo [20].

Para este fin se utiliza la estructura de 1 KB *objset_phys_t* como se muestra en la Ilustración 3 y se detalla en la Tabla 11.

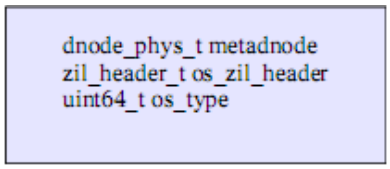

ILUSTRACIÓN 3. ESTRUCTURA DE LOS GRUPOS DE OBJETOS (*OBJSET_PHYS_T*). TOMADO DE [20]

2.1.2.6 DSL

Es la capa de las instantáneas y los grupos de datos (Dataset and Snapshot Layer) que tiene como función describir y gestionar las relaciones entre los grupos de objetos (*object sets*). Estos grupos de objetos pueden ser [20]:

- Sistemas de archivo: Almacena y organiza los objetos para su fácil acceso.
- Clones: Es idéntico a un sistema de archivos con la diferencia que se origino de una instantánea con su contenido.
- Instantáneas: Es un sistema de archivos, clon o volumen en un punto de tiempo con la característica de solo lectura.
- Volúmenes: Es un volumen lógico exportado por ZFS como un dispositivo de bloques.

Como un sistema de archivos, un clon o un volumen pueden estar asociados a una instantánea y está asociada a un clon, el propósito del DSL es básicamente gestionar esas relaciones. No se puede eliminar un sistema de archivos, un clon o un volumen a menos que la instantánea asociada también sea eliminada o en el caso que la instantánea tenga un clon asociado entonces no se puede eliminar la instantánea a menos que el clon también sea eliminado [20].

2.1.2.6.1 EL *"META OBJECT SET"* (MOS)

En el DSL los grupos de objetos se representan por medio de un "*dataset"*. El *dataset* gestiona el espacio y contiene información sobre la localización de los grupos de objetos. Los *dataset* a su vez se agrupan jerárquicamente en el "*DSL Directory"*. Este gestiona las relaciones de los grupos de *datasets*, sus propiedades y siempre existe solo un *dataset* activo, los demás *datasets* se relacionan con el activo. En la Ilustración 18 se puede ver que el *DSL Directory* tiene un solo *dataset* activo que es el sistema de archivos. A la derecha del *dataset* activo vemos una lista de instantáneas tomadas en algún punto del tiempo. A la izquierda del *DSL Directory* vemos un objeto ZAP (ver sección 2.1.2.7) con una lista de dependencias hijas [20].

Los *datasets* usan el objeto de tipo DMU_OT_DSL_DATASET con la estructura "*dataset_phys_t".* El DSL implementa un grupo de objetos del tipo DMU_OST_META (ver Tabla 11). Este grupo de objetos lo llaman el "*Meta Object Set*" (MOS). Solo puede existir un MOS por pool y es de gran importancia notar que el uberblock apunta a un MOS (ver Tabla 7 e Ilustración 19). Adicionalmente, El objeto *DSL Directory* contiene la estructura "*dsl_dir_phys_t"* en el bonus buffer.

2.1.2.6.2 EL "*OBJECT DIRECTORY*"

El MOS tiene un objeto del tipo DMU_OT_OBJECT_DIRECTORY (ver Tabla 9), llamado como el "*object directory".* Este objeto siempre está localizado en el segundo elemento (índice 1) en el bloque donde apunta el arreglo *dnode* (Ilustración 19) [20]. Luego con el *object directory* podemos localizar cualquier objeto mediante los grupos de objetos que tiene referenciados.

El *object directory* es un objeto ZAP (ver 2.1.2.7) por lo que contiene 3 atributos del tipo nombre-valor (ver Tabla 12).

2.1.2.7 ZAP

Es un modulo que opera con objetos ZAP (*ZFS Attribute Processor*), estos almacenan atributos de la forma nombre-valor. El nombre es un string de 256 bytes y el valor es un arreglo variable de enteros. Los diferentes tipos de objetos ZAP (ver Tabla 13) se usan para almacenar propiedades de los *dataset* y propiedades del pool, y para navegar por los objetos del sistema de archivos [20].

2.1.3 INFORMÁTICA FORENSE EN ZFS

Luego de conocer la arquitectura, las estructuras de datos y los métodos de almacenamiento en ZFS, nos enfocaremos en la problemática principal del artículo: las técnicas anti-forenses; finalizando con un análisis de dichas técnicas en ZFS.

Para concebir la problemática anti-forense primero hay que clarificar que la informática forense aparece para enfrentar los desafíos y técnicas de los intrusos informáticos, así como garante de la verdad alrededor de la evidencia digital que se pudiese aportar en un proceso [22].

La evidencia digital es el tipo de evidencia física que está construida de campos magnéticos y pulsos electrónicos que pueden ser recolectados y analizados con herramientas y técnicas especiales [23].

Esta evidencia digital requiere de su correspondiente identificación, preservación, extracción, análisis, interpretación, documentación y presentación para detallar, validar y sustentar las hipótesis que sobre un evento se hayan formulado [22].

La importancia de la recolección de esta evidencia digital radica en que nos permitirá lograr los tres objetivos de la informática forense [24]:

1. La compensación de los daños causados por los criminales o intrusos.
2. La persecución y procesamiento judicial de los criminales.
3. La creación y aplicación de medidas para prevenir casos similares.

Luego para recolectar y poder analizar evidencia digital, se debe primero determinar donde se encuentra localizada esta evidencia. Generalmente y en su mayoría la evidencia digital se encuentra en un dispositivo de almacenamiento digital como un disco duro.

Para lograr acceder de manera eficiente a los datos del disco, el sistema operativo utiliza el sistema de archivos que proporcionara un mecanismo para que los usuarios almacenen sus datos a largo plazo mediante una jerarquía de archivos y directorios [8].

Así pues, los investigadores o profesionales de la informática forense requieren de conocimiento detallado del sistema de archivos para poder examinarlo y conseguir una lista de archivos que se encuentran en un directorio, recuperar contenido eliminado o ver el contenido de un sector [8] y de esta manera lograr probar la existencia de los archivos así hayan sido borrados, teniendo precaución de no contaminar o modificar la evidencia durante el análisis, para que pueda ser válida en una corte [25].

2.1.3.1 CONFERENCIA: "OPEN SOLARIS FORENSICS TOOLS PROJECT"

Para lograr la validez de la evidencia digital y evitar la contaminación de la misma, son necesarias ciertas herramientas y metodologías. Por ejemplo para hacer análisis de sistemas de archivos: Ext2 y Ext3 de Linux, UFS de Unix y BSD, HFS de Apple y FAT y NTFS de Microsoft; se pueden utilizan herramientas de código abierto como "*The Sleuth Kit"* y "*The Coroner's Toolkit*". Sin embargo para el caso de ZFS de Sun, no es posible utilizar estas herramientas ya que la estructura difiere de los tradicionales sistemas de archivos mencionados [25].

En el 2004 Evtim Batchev propuso la creación del "Open Solaris Forensics Tools Project" [26] soportado por la comunidad y con una clara preocupación en la falta de herramientas específicas de análisis forense para ZFS. La idea principal detrás del proyecto es crear o adaptar herramientas y desarrollar metodologías que apoyen a la investigación forense y a la repuesta de incidentes en las plataformas Solaris [26]. La actual etapa del proyecto encierra las siguientes áreas:

- Live System Evidence Gathering Instructions and Methodologies: Metodología para recopilar datos mediante técnicas no intrusivas que implican la ejecución del Kernel sin contaminar la evidencia [27].
- ZFS forensics tool set and methodology.
- Live system monitoring and active data gathering tool sets.
- Malware detection tool sets especially for LKM rootkits: Live Kernel Data Gathering Scripts: La herramienta "findrootkit" realiza varias pruebas para detectar actividades sospechosas en el Kernel y rastrea módulos falsos para facilitar la detección de rootkits [27].
- Open Solaris Forensics toolkit compilation in a bootable DVD/CD/PenDrive ISO: Herramienta que incluya un conjunto de scripts para una automática recolección de datos [27].
- Solaris fingerprint database tools:
    - Solaris Fingerprint Database Sidekick: Herramienta usada para ayudar a los administradores recoger los "file fingerprints" de una variedad de tipos de archivos y clases [27].
    - Solaris Fingerprint Database Companion
      Herramienta usada para enviar automáticamente "fingerprints" a la base de datos "Solaris Fingerprint Database", retornando una respuesta de tipo texto para ser procesada posteriormente de ser necesario [27].

2.1.3.2 IMPLICACIONES FORENSES EN ZFS

Para el desarrollo de las herramientas y metodologías en el sistema de archivos ZFS, primero se debe conocer a fondo la arquitectura y diseño del mismo, para luego analizar las implicaciones forenses que este sistema de archivos trae consigo.

A continuación se revisaran los aspectos teóricos sobre las repercusiones de las nuevas funcionalidades y paradigmas de ZFS en la forensia digital.

2.1.3.2.1 COPIAS EN ESPACIOS NO ASIGNADOS

Con el nuevo paradigma y diseño del sistema de archivos ZFS, la localización y el método para llegar al contenido de los sectores de datos es una nueva noción. El manejo de la información ha cambiado debido al modelo de transacciones COW (copia por escritura), que evita la corrupción y la sobrescritura de los datos. Es decir que al modificar un bloque el cambio no se realizará directamente sobre éste, sino en

cualquier lugar no asignado en el disco y los metadatos asociados son actualizados de la misma manera con el modelo COW [2].

El impacto de ésto radicará en que el examinador forense conseguirá detectar y recuperar numerosas copias de metadatos y bloques de datos [2]. Esto quiere decir que va a aumentar la probabilidad de encontrar evidencia, porque hay mayor seguimiento de los archivos por parte del sistema de archivos.

Adicionalmente ZFS utiliza un objeto ZIL (ver Tabla 11) por sistema de archivos, que funciona como un log que registra las transacciones y acciones del usuario que posteriormente permitirá replicar información o regresar a estados previos [2]. Para los examinadores forenses será de gran utilidad ya que, es como un historial de sucesos del sistema de archivos, lo que les permitirá tener un idea más clara del estado del sistema, por ejemplo en el momento de la detención de una máquina.

2.1.3.2.2 COPIAS EN ESPACIOS ASIGNADOS

ZFS además de tener múltiples copias de metadatos y datos en espacios no asignados mediante el uso del modelo COW, también tiene la funcionalidad de tener hasta 3 copias de metadatos y datos mediante los "*Ditto blocks"* (ver Tabla 8 - DVA Data Virtual Address).

Los "*Ditto blocks*" permiten establecer políticas para la protección de los datos mediante el número de DVAs (1-sencillo, 2-doble o 3-triple) que contiene el puntero del bloque [20] [21].

La política para la protección de los datos por defecto es [2] [21]:

- Sencillo para los datos de usuario.
- Doble al nivel de los *dataset*, como objetos del sistema de archivos (archivos, directorios, metadatos).
- Triple para metadatos globales al nivel del pool.

El usuario puede definir el numero de "*Ditto blocks"* localizando los DVAs en el puntero de bloque (ver Tabla 8). Los DVAs se encargan de identificar la dirección del bloque donde apuntan los datos mediante la combinación del vdev y el offset (ver Ilustración 16).

La implicación de los "*Ditto blocks"* radicará entonces en que existirán copias de metadatos y datos en espacios asignados; lo que significa que las diferentes estructuras de datos implicadas con el

funcionamiento de los "*Ditto blocks"* como los punteros de bloque (*blkptr_t*) deberán ser bien conocidas por el examinador forense [2].

2.1.3.2.3 INSTANTÁNEAS Y CLONES

Los examinadores muchas veces se ven limitados con la información obtenida de archivos temporales, archivos corruptos, archivos log y registro de transacciones[5]; por lo que los sistemas de archivos con tecnologías de backup son de gran ayuda a la hora de esclarecer sucesos ya que pueden realizar instantáneas (ver 2.1.1.5) en varios instantes de tiempo, lo que permite a los investigadores forenses comparar las instantáneas cronológicamente y analizar los cambios que se efectuaron en los datos, por ejemplo en el caso de una intrusión [2].

2.1.3.2.4 COMPRESIÓN

La compresión siempre ha sido un desafío en la indexación de datos que sirve para su posterior análisis por parte del investigador forense. El desafío radica en que se requiere de un preprocesamiento adicional para descomprimir los datos y en ZFS la descompresión no es tan simple, ya que este sistema de archivos implementa una compresión a gran escala; es decir, que no solo aplica a los objetos, sino también al nivel de las estructuras de datos, metadatos y *datasets*. La compresión tiene una variedad de estados como: sin compresión, comprimido y comprimido con una variedad de algoritmos [2].

Las diferentes configuraciones se establecen globalmente en el zpool y solo afecta a los nuevos datos almacenados después de la configuración. Por defecto, los metadatos tienen la compresión activa y solo se pueden desactivar en los punteros de bloques indirectos (en los punteros de bloques directos permanece activa). Además los objetos ZAP y los datos al parecer no son comprimidos, pero es sencillo de cambiar. Luego las herramientas forenses que se desarrollen para analizar la evidencia en ZFS, primero deben analizar las estructuras de datos para revisar la configuración de compresión. Así, se lograra descomprimir los objetos en cuestión, para luego poder realizar los procesos de búsquedas, extracción y análisis [2].

2.1.3.2.5 MEDIDAS DE TAMAÑO DINÁMICO

[5] También conocido como *"journaling",* que tiene por objetivo almacenar todas las operaciones del sistema de archivos como los cambios en los metadatos.

En comparación con los sistemas de archivos tradicionales que almacenan el contenido de los archivos en unidades lógicas de asignación[6], ZFS usa lo que llama FSB ("*File System Blocks*"). La función de estas unidades o bloques es que cuando un archivo incremente su tamaño se le asignen los bloques necesarios para su correcto almacenamiento. El tamaño de estos bloques usualmente se definía en la instalación y creación del sistema (512KB, 1024KB…) y no variaba después de la instalación. En ZFS este tamaño sí puede ser ajustado después de la instalación, ya que se puede cambiar dinámicamente dependiendo de la necesidad del archivo [2].

El tamaño del registro del FSB tiene un tamaño máximo de 128 KB (por defecto) y dinámicamente se ajusta al tamaño de los datos asignados. El lugar donde se almacena el tamaño del FSB de un objeto especifico como un archivo, es en el "*dnode*" en el campo "*dn_datablkszsec"* [2] (ver Tabla 10).

Las unidades lógicas de asignación tradicionales dieron origen al concepto del slack space (ver 2.2.2.2.2), donde muchos investigadores se beneficiaron encontrando viejos contenidos de archivos [2], sin embargo también benefició a atacantes para esconder contenido.

En el caso de los tamaños dinámicos, se encontrara menos slack space. Los archivos más pequeños que el tamaño del registro del FSB poseen un slack space casi inútil, además de suponer que está ocupado de ceros. Para los archivos más grandes en muchos de los casos, presentaran una cantidad significativa de slack space que usualmente no será tan amplia como en otros sistemas de archivos [2].

2.1.3.3 CONFERENCIA: "ZFS ON-DISK DATA WALK (OR: WHERE'S MY DATA)"

En junio del 2008 en Praga, Max Bruning [28] [26] realizó una conferencia que explicaba como es el método para llegar a los datos en los sectores del disco y de esta manera comprender como recuperar un archivo borrado en ZFS (ver su weblog [29]). Para este proceso es necesario el conocimiento de las siguientes estructuras de datos [30]: (sus archivos cabecera los podemos encontrar en "*uts/common/fs/zfs/sys/*.h")*

- uberblock_t (ver Tabla 7)
- blkptr_t (ver Tabla 8)
- dnode_phys_t (ver Tabla 10)
- objset_phys_t (ver Tabla 11)
- ZAP Objects (ver Tabla 13)

[6] Sector lógico, o bloque o cluster o unidad de asignación. Es una agrupación de sectores contiguos y es el espacio mínimo que va a ocupar un fichero. Lo "dibuja" y maneja el S.O. Su tamaño depende del sistema de archivos [56].

Objetos en el bonus buffer:

- dsl_dir_phys_t
- dsl_dataset_phys_t
- znode_phys_t

A continuación el paso a paso del camino para llegar a los datos en los sectores del disco [31] [30] (revisar la Ilustración 19 y la Ilustración 20):

1. Encontrar el *uberblock* activo y su respectivo puntero de bloque.
2. Encontrar el dnode del "*Meta Object Set*" (MOS).
3. Encontrar el dnode "*object directory"* y el objeto ZAP respectivo.
4. Encontrar el objeto DSL Directory.
5. Encontrar el objeto DSL dataset.
6. Usando el DSL dataset, encontrar el grupo de objetos "*ZFS file system*".
7. Usando el objeto "*ZFS file system*" obtener el "*Master Node*" y su objeto ZAP.
8. Del objeto ZAP del "M*aster Node*", obtener el directorio raíz de tipo dnode.
9. Del puntero de bloque del directorio raíz, encontrar el id del objeto del archivo objetivo
10. Con la dirección almacenada en el objeto, recuperar el bloque de datos del disco.

2.2 TÉCNICAS ANTI-FORENSES

Aunque es un tema relativamente nuevo, existen bases que nos ayudan a entender y estudiar la materia con un poco más de rigor científico. Para poder explicar el impacto de éste tema en la actualidad, inicialmente definiremos algunos de los principios básicos que nos serán de gran ayuda para entender holísticamente el tema.

2.2.1 INTRODUCCION Y DEFINICIÓN

Lo anti-forense es más que tecnología, es un enfoque al hacking criminal que puede resumirse así*: "Haz difícil que te encuentren e imposible de probar que te encontraron"* [9].

¿Qué son las técnicas anti-forenses?, la palabra anti se encuentra definida en el diccionario de la Real Academia Española como "opuesto" o "con propiedades contrarias" y forense significa "Perteneciente o relativo al foro" (en la antigua Roma el acusado y el demandante se presentaban ante un foro que se encontraba precedido por sujetos notables de la comunidad y en el cual el demandante exponía las

pruebas que demostraban que el acusado era culpable de crimen), por lo tanto éste término fue acuñado para identificar a las personas que se dedican a buscar la verdad por medio de las pruebas que fueron dejadas en la escena del crimen; por ende la ciencia forense es aquella ciencia que intenta develar la verdad por medio de métodos probatorios científicos.

Teniendo en cuenta las anteriores definiciones podemos afirmar que son todas aquellas técnicas que comprometen la integridad y veracidad de las pruebas que serán presentadas para la reconstrucción de los hechos y el grado de participación de los actores.

Ahora realizaremos una aproximación un poco más acertada utilizando como fuente la definición que nos proporciona Ryan Harris [10]. El define las técnicas anti-forenses como:

> *"Cualquier intento para comprometer la disponibilidad o utilidad de la evidencia en un proceso forense"*

Modificando esta definición se podría expandir de tal forma que oriente a los investigadores forenses en la práctica como lo plantea el profesor Cano [32]:

> *"Cualquier intento exitoso efectuado por un individuo o proceso que impacte de manera negativa la identificación, la disponibilidad, la confiabilidad y la relevancia de la evidencia digital en un proceso forense"*

Ahora bien, revisadas las definiciones y contrastando con la realidad vemos que las estrategias anti-forenses se han desarrollo justo al lado de la informática forense. Algunos dicen que la anti-forensia se está desarrollando más rápido porque lo que antes era solo posible para la élite ahora se configura en herramientas automatizadas. Luego, cualquier persona decidida podría generar dolores de cabeza en el camino de las investigaciones forenses, ahora que las herramientas están allí para hacer todo ésto posible [33].

2.2.2 CLASIFICACIÓN DE LOS MÉTODOS ANTI-FORENSES

Para lograr materializar las estrategias anti-forenses es necesario profundizar en los métodos propuestos por los investigadores en la materia. Ryan Harris [10] propone la siguiente clasificación:

2.2.2.1 DESTRUCCIÓN DE LA EVIDENCIA:

Este método pretende evitar que la evidencia sea encontrada por los examinadores forenses destruyéndola de tal manera que sea inútil para el proceso de investigación. Además no busca que la evidencia sea inaccesible si no que sea irrecuperable [10].

Por ejemplo, en el mundo físico, verter cloro en la sangre para destruir el ADN o en el caso del mundo digital, eliminar o sobrescribir un archivo [10].

Este es el método más fácil de ejecutar por su simplicidad [35]. Sin embargo cuando se ejecutan estos procesos de destrucción, se puede crear nueva evidencia como huellas dactilares en la botella que contenía el cloro o el software utilizado para eliminar el archivo puede dejar trazas que sirven como pruebas [10].

Luego, para comprender como eliminar los archivos sin dejar trazas, revisaremos el proceso de eliminación, para posteriormente saber cuáles son las trazas y como se limpian éstas.

2.2.2.1.1 ELIMINAR

Cuando se elimina un archivo, lo que el sistema de archivos hace en realidad es remover la referencia a los datos, es decir marcar ese elemento como eliminado, pero no lo borra físicamente del disco magnético. De esta manera, cuando el sistema de archivos requiere espacio para grabar nuevos archivos, va a la marca y sabe que puede reutilizar ese espacio.

En los sistema de archivos UNIX se libera el Inode y se establece como “unallocated” mediante la función unlink() [35].

En el sistema de archivos NTFS se realiza de manera similar, se desasigna la entrada en la Master File Table (MFT), eliminando la bandera de indicación: “en uso” [8] [36].

Sin embargo los datos persisten en el disco como bloques sin asignar (“dirty” data blocks). Lo que significa que un investigador está en capacidad de encontrar estos datos, sin embargo pudieron ser sobrescritos; por lo tanto la siguiente estrategia tiene el propósito de no dejar residuos.

2.2.2.1.2 LIMPIAR (“WIPING”)

Los investigadores cuando están en busca de archivos eliminados en sistemas de archivos Unix se centran en buscar [35] [37]:

- Residuos o huellas de archivos borrados

    - Entradas de directorios
    - “Dirty” Inodes
    - “Dirty” data blocks
- Actividad del sistema de archivos
    - Marcas de tiempo en el Inode
- Conocimiento previo de las herramientas anti-forenses
- Cadenas de caracteres

Luego, para que esta estrategia pueda lograr eliminar los archivos completamente sin dejar rastros, deberá tener en cuenta dos cosas:

2.2.2.1.2.1 SOBRESCRIBIR

El objetivo es sobrescribir los datos con fines destructivos, reemplazándolos con ceros, números, caracteres aleatorios, etc; logrando de esta manera que ningún software pueda lograr recuperar la información en cuestión. Sin embargo y teniendo en cuenta que la información es digital pero el almacenamiento es análogo, es posible recuperar información mediante análisis magnético ya que los datos persisten como pequeñas modulaciones en los datos nuevos [35], por lo que entre mayor sea el número de pasadas o fases, mayor será la dificultad del análisis.

Algunos de los algoritmos y estándares que existen son [34] [35] [38]:

- ✓ Zeroes
- ✓ Pseudo-random numbers
- ✓ Pseudo-random & Zeroes
- ✓ DoD 5220.22-M (3 Passes)
- ✓ DoD 5200.28-STD (7 Passes)
- ✓ NAVSO P5239-26
- ✓ AFSSI-5020
- ✓ AR380-19
- ✓ Russian Standard – GOST
- ✓ B.Schneier’s algorithm (7 passes)
- ✓ German Standard, VSITR(7 passes)
- ✓ Peter Gutmann(35 passes)
- ✓ US Army AR 380-19 (3 passes)
- ✓ North Atlantic Treaty Organization – NATO Standard
- ✓ US Air Force, AFSSI 5020 (método de borrado que usa la fuerza aérea de estados unidos)

Algunas de las herramientas que utilizan estos algoritmos y métodos [34] [35] [38]:

- shred (Unix)
- wipe (Unix)
- overwrite (Unix)
- Srm (Unix, Mac OS X)
- PGP secure delete (Windows)
- Eraser (Windows)
- Sdelete (Windows)
- Darik's Boot and Nuke (DBAN: borrado de todos los discos duros)

2.2.2.1.2.2 BORRAR HUELLAS

Las herramientas de borrado seguro en sistemas de archivos Unix solo remueven el contenido de los data blocks, dejando los Inodes y las entradas de directorios intactas.

Luego, para una eliminación segura de todo rastro, que implique la existencia de un archivo, encontramos: "The Defiler's Toolkit", que provee de dos herramientas para este fin [35] [37]:

- Necrofile: Localiza y erradica los "dirty" Inodes.
- Klismafile: Localiza directorios que posean entradas borradas para erradicarlas.

En los sistemas de archivo NTFS, herramientas como *SDelete* no solamente sobrescriben los archivos, sino también llenan cualquier parte libre existente de la MFT con archivos que se ajusten dentro de un registro MFT [39].

2.2.2.2 OCULTAR LA EVIDENCIA:

El objetivo es lograr hacer la evidencia lo menos visible para el examinador, para que sea menos probable que se incorporen en el proceso forense [10].

Por ejemplo: en el mundo físico, esconder un arma o enterrar un cuerpo; en el mundo digital, los archivos pueden ser renombrados para confundir o se pueden colocar en lugares inusuales para evadir el software forense [10].

Sin embargo, al ocultar la evidencia no garantiza que tenga éxito, ya que se basa en buscar puntos ciegos del investigador [10].

Lo primero que se debe estudiar para comprender esta técnica, son las diferentes áreas en la que está organizado el disco, ya que los atacantes las aprovechan para ocultar material [34] [38] [40]:

ILUSTRACIÓN 4. HOST PROTECTED AREA & DEVICE CONFIGURATION OVERLAY. TOMADO Y ADAPTADO DE [40]

- Host Protected Area (HPA)

  Área de discos ATA que son inaccesibles para el sistema operativo, usadas con frecuencia para software del fabricante. No es visible para la BIOS y se puede usar en abuso para ocultar datos.

- Device Configuration Overlay (DCO)

  Usada para almacenar metadatos del disco y no es visible para el sistema operativo. También se usa en abuso para ocultar datos.

- Espacio no asignado

  Espacios no asignados actualmente, que sirven para almacenar archivos.

- Slack space

  Espacio no utilizado ubicado al final de la mayoría de los archivos.

- Sectores buenos que son marcados como malos.

Sumado a estas áreas en que está organizado el disco, encontramos diferentes metodologías para ocultar datos:

2.2.2.2.1 FIST (FILESYSTEM INSERTION & SUBVERSION TECHNIQUE)

Esta técnica inserta datos donde no pertenecen, más exactamente en áreas que no son usadas que contengan estructuras de datos, como por ejemplo en archivos metadatos, journals, entradas de directorios, etc. Existen varias implementaciones de esta técnica en sistemas de archivos Unix [35] [37]:

- Rune FS

  Almacena más de 4GB de datos en los "*bad blocks*".

- Waffen FS

  Almacena 32MB de datos en el archivo *journal* de ext3.

- KY FS

  Almacena los datos en archivos de directorios que posean entradas nulas.

- Data Mule FS

Almacena los datos en el espacio reservado del *inode*.

2.2.2.2.2 SLACK SPACE

Imaginemos que tenemos como unidad de asignación[7] a ocho bloques de 512 bytes cada uno, es decir un total de 4096 bytes de espacio no asignado listo para almacenar cualquier cosa. Un nuevo archivo es escrito usando todo el espacio (4096 bytes). Luego este archivo es eliminado y el espacio es ahora no asignado. Un nuevo archivo es creado pero no usa los 4096 bytes de espacio, solo 3380 bytes son escritos sobrescribiendo los primeros 3380 bytes del archivo borrado. El *slack space* contiene 716 bytes de información del archivo original que fue borrado, por lo que esta área puede ser usada para almacenar información oculta (ver Ilustración 5) [34] [38].

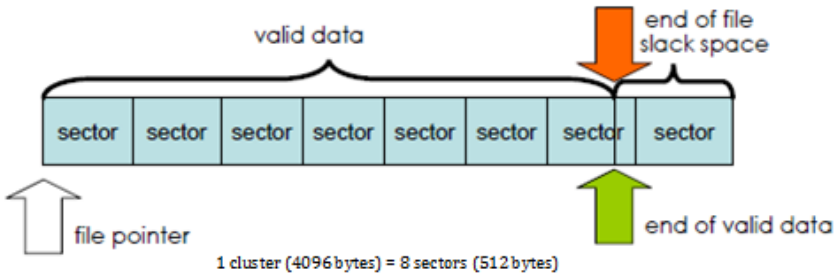

ILUSTRACIÓN 5. SLACK SPACE. TOMADO Y ADAPTADO DE [35]

En este espacio se pueden tener archivos de todo tipo, como por ejemplo archivos comprimidos y cifrados. Por otro lado, hay que seleccionar adecuadamente cuales archivos se van a usar para ejecutar esta técnica: podría ser los primeros N archivos, archivos aleatorios o los últimos archivos modificados [41].

Las herramientas que se usan para lograr este cometido son:

- MAFIA -> Slacker
  Esconder archivos en el *slack space* en NTFS
- Bmap
  Analizador de *slack space* para Unix

2.2.2.2.3 ESTEGANOGRAFÍA

[7] Sector lógico, o bloque o cluster o unidad de asignación. Es una agrupación de sectores contiguos y es el espacio mínimo que va a ocupar un fichero. Lo "dibuja" y maneja el S.O. Su tamaño depende del sistema de archivos [56].

Arte y ciencia de escribir mensajes ocultos, es decir esconder información en un archivo sin cambiar su comportamiento. Existen diferentes herramientas para ocultar datos en archivos de imágenes o en archivos de audio. Algunas de estas herramientas son [40] [42]:

- Hermetic Stego
- S-Tools
- 4t HIT Mail Privacy Lite
- Camouflage
- Steghide
- Hydan
- Anti-técnica (para detectar Esteganografía):
    - Stegsecret y SegDetect
    - Gargoyle

2.2.2.2.4 CIFRADOS Y COMPRIMIDOS

El cifrado es el proceso de transformación de la información para que sea ilegible con excepción de la persona o personas que posean la clave. El cifrado normalmente también se combina con la compresión de los datos, lo que significa que hay un proceso adicional para descomprimirlos.

Muchos autores incluyen el cifrado de datos como un método aparte, sin embargo teniendo en cuenta los métodos propuestos por Harris, el cifrado clasificaría aquí por ser una técnica que se puede usar con el propósito de esconder información.

El cifrado regularmente es un reto para el analista forense y puede aplicarse de varias maneras [34]: a un disco entero, a archivos específicos o al material antes de ocultarlo.

Algunas de las herramientas que se usan para lograr este cometido:

- GnuPG - OpenPGP (estándar RFC2440) –
  Para cifrar archivos.
- TrueCrypt
  Para cifrar discos.

2.2.2.2.5 ADS (ALTERNATE DATA STREAMS)

Es una característica en NTFS, que permite mantener información asociada del fichero (meta información) sin requerir de espacio adicional. Esta característica fue creada para brindar compatibilidad con archivos jerárquicos en HFS.

Luego, podemos incluir archivos a un archivo sin que se sospeche de éstos. Esta característica no es vista por los antivirus ni por anti-troyanos por lo que es ideal para esconder logfiles, keyloggers o denegación de servicios del sistema [35].

Por otro lado la herramienta "Streams" nos permite encontrar archivos y directorios que tengan "alternate data streams". Para ver su funcionamiento en detalle revisar [34].

2.2.2.3 ELIMINACIÓN DE LAS FUENTES DE LA EVIDENCIA:

Es la desactivación de los mecanismos de registro de evidencia existentes para evitar tener que destruirla. Por ejemplo: en el mundo físico un criminal puede usar guantes de goma para utilizar un arma con el fin de no dejar huellas dactilares, neutralizando la evidencia [10]; en el mundo digital esta neutralización de las fuentes de la evidencia aplica, ejecutando binarios en un sistema remoto sin crear archivos en el disco [43].

La técnica que utiliza el investigador para hallar evidencia radica en capturar y analizar toda información del disco, la lista de procesos activos, archivos abiertos, puertos abiertos, etc. Luego, la anti-técnica deberá tener en cuenta nunca tocar el disco, nunca abrir un puerto, nunca abrir un archivo, nunca crear un proceso nuevo y demás [41].

Sin embargo, este método al igual que la destrucción de la evidencia, pueden crear pruebas por iniciativa propia, como el no encontrar huellas dactilares en un arma puede llevar a un buen investigador a pensar que el criminal uso guantes de goma y que el asesinato fue cuidadosamente planeado [10].

Con respecto al método, siempre se debe tener en cuenta dos principios básicos [43]:

1. Prevenir que los datos se almacenen en el disco.
2. El uso de herramientas comunes y no personalizadas, siempre que sea posible.

Teniendo en cuenta estos principios, el objetivo es lograr interactuar con el sistema operativo de la siguiente manera [43]:

1. Operar únicamente en memoria:

   Para operar en memoria, se requiere que un programa en la máquina objetivo actué como un servidor para de esta manera lograr interactuar con el sistema operativo.

2. Usar herramientas comunes antes que las personalizadas:

   Esto significa que no existe nada de valor que el analista pueda recuperar, reduciendo la efectividad de la investigación forense. Por ejemplo escribir un backdoor usando gawk[8].

Si lo anterior se logra con éxito, se procede en tiempo de ejecución y manipulando la memoria; lograr que un proceso existente realice acciones como acceder a recursos del sistema o robar datos [35].

Entre las diferentes técnicas y herramientas que tienen presente no escribir en el disco encontramos:

2.2.2.3.1 SAM JUICER

Sam Juicer es una herramienta del proyecto MAFIA, que permite volcar los hashes de las contraseñas del SAM en Windows. La conexión se realiza mediante una consola meterpreter[9] que permite interactuar sin necesidad de tocar el disco ni el registro y sin iniciar un servicio como si lo hacían las viejas técnicas como el "pwdump".

El funcionamiento de la técnica es el siguiente [41]:

1. Usar un canal meterpreter.
2. Usar memory injection directamente.
3. El flujo de datos pasa sobre la conexión existente.
4. Una falla no deja evidencia.

2.2.2.3.2 SYSCALL PROXYING

Es una de las nuevas técnicas para realizar pruebas de penetración, que tiene por objetivo simplificar la escalada de privilegios proporcionando una interfaz directa con el sistema operativo que permite automáticamente al código del atacante y a las herramientas controlar los recursos remotos. También usa

[8] Herramienta del sistema Unix utilizada para manipular ficheros de texto.
[9] Plugin avanzado para utilizar sobre sistemas Windows para cargar todo en memoria sin crear ningún proceso adicional.

los procesos como un proxy y el sistema llama a un servidor remoto, simulando efectivamente una ejecución remota [44].

2.2.2.3.3 REXEC

El propósito de esta técnica (Unix) es ejecutar binarios en un sistema remoto desde la memoria. Básicamente el proceso es [43]:

1. Usar un IUD para ganar acceso a un espacio de dirección.
   The GNU debugger (gdb)
2. Subir los binarios a ejecutar en memoria.
3. Cargar los binarios en el espacio de dirección.
   Librería: ul_exec
4. Control de transferencia de la ejecución del binario.
   Librería: gdbprpc

2.2.2.3.4 XSH (THE "EXPLOIT SHELL")

Acceder a rexec requiere de un cliente complejo que pueda tener acceso a una Shell. La herramienta XSH encapsula el protocolo rexec funcionando básicamente como un exploit que sirve para acceder a una Shell aplicando con éxito la anticoncepción de los datos en el momento de la penetración al sistema [37] [43].

2.2.2.3.5 FTRANS

Es una herramienta puramente anti-forense diseñada para que no haya binarios que capturar (evidencia), operando en un ambiente forense extremadamente hostil de un honeypot. El programa usa un servidor personalizado (sistema hackeado) que usa SSL para copiar los binarios del cliente en su propio espacio de dirección para posteriormente ejecutar los binarios utilizando la librería ul_exec (ver 2.2.2.3.3) [35] [43].

2.2.2.3.6 BOOTABLE LIVE CD & USB

Aunque la mayoría de las técnicas y herramientas revisadas anteriormente están encaminadas a interactuar remotamente, también es posible utilizar directamente en la máquina objetivo un bootable live cd o usb con las siguientes condiciones [45]:

1. La distribución que se use no debe auto montar los medios físicos (discos, etc).

2. Se pueden montar las unidades pero solo en modo lectura, para que no haya cambios en los datos ni en las marcas de tiempo.

El sistema operativo del CD o de la USB usa la RAM para almacenar el sistema de archivos, lo que significa que una vez apagado el computador no existirá evidencia en el disco y en caso de la RAM, los datos no se retienen durante más de unos segundos dependiendo de la temperatura, sin embargo es bueno saber que hay métodos avanzados de recuperación de datos en la RAM [45]. Luego el disco nunca se tocara cumpliendo con el principio básico del método logrando así no dejar ningún rastro de evidencia.

Algunas distribuciones Linux basadas en seguridad: BackTrack, Operato,r PHLAK, Auditor, L.A.S Linux Knoppix-STD, Helix.

2.2.2.4 FALSIFICACIÓN DE LA EVIDENCIA:

Es la modificación o edición de la evidencia existente o la creación de evidencia inválida con el fin de engañar y confundir a los investigadores [10].

Este método no destruye la evidencia sino que invalida la verdadera evidencia. Por ejemplo: en el mundo físico, el criminal puede crear falsa evidencia para convencer al investigador que la víctima se suicido; en el mundo digital la falsificación incluiría usar cuentas de otras personas o lanzar ataques desde máquinas remotas que impliquen a otras personas [10].

Entre las técnicas más conocidas con el objetivo de engañar a los investigadores encontramos:

2.2.2.4.1 CAMBIAR MARCAS DE TIEMPO

Para cambiar las marcas de tiempo en sistemas Unix y Mac se utiliza el comando “touch” [41]. Este programa cambia la fecha y hora de los tiempos de acceso y modificación de un archivo [46]. Luego, es posible colocarle a un archivo una hora diferente a la actual irrumpiendo en el proceso forense cuando el analista examine las marcas de tiempo.

Por otro lado en Windows con sistemas de archivo NTFS encontramos la herramienta “Timestomp” del proyecto MAFIA (Metasploit Anti-Forensic Investigation Arsenal), que se especializa en falsificar la evidencia digital aprovechándose de las vulnerabilidades del sistema de archivos NTFS, atacando los atributos de tiempo MACE [34] [35] [40]:

- Modification: La última modificación de los metadatos.

- Access: El ultimo acceso al archivo.
- Create: Cuando el archivo fue creado.
- Entry Modification: Marcas de tiempo en la Master File Table.

De modo que esta herramienta logra vulnerar el software forense (EnCase) y el MS Antispyware [41].

2.2.2.4.2 FILE CARVING

Para buscar archivos de manera avanzada se utiliza la técnica "file carving", que básicamente realiza una búsqueda bit a bit de encabezados de tipos de archivos en el disco. La anti-técnica es uno de los grandes problemas del análisis forense, debido a que Windows solo usa la extensión del archivo para identificar el formato y el software forense como EnCase utiliza además los encabezados. Luego, la manera en que se logra evadir esta detección constaría de [41] [35]:

1. Cambiar el encabezado: Se modifica un byte del encabezado byte repercutiendo en la variación de la firma y que el software forense no logre determinar el tipo de archivo.
2. Cambiar el encabezado y la extensión: Si un archivo es manipulado para alterar su encabezado y su extensión para que coincidan en un formato de archivo especifico, podríamos cambiar un archivo de formato texto a formato ejecutable logrando así vencer las firmas de archivo vulnerando el software forense que lo reconocerá equívocamente:

   Esta técnica puede ser útil por ejemplo para cambiar los logs de modo texto a ejecutables o DLLs para vulnerar el software forense. La herramienta "Transmogrify" del proyecto MAFIA, nos permite automatizar el cambio de encabezados y extensiones para enmascarar y desenmascarar archivos de cualquier tipo.

2.2.2.4.3 COLISIONES HASH

Los atacantes modifican y recompilan las herramientas usadas o realizan cambios binarios para permanecer ocultos bajo el radar. Los investigadores por su lado comparan la función hash de los archivos, para revisar inconsistencias y así detectar si el archivo sufrió alguna variación [35]. Por ejemplo la modificación de un byte repercutirá en la variación total de la función hash md5.

La anti-técnica de esto se conoce como "*hash collisions*", que tiene por objetivo lograr que dos archivos diferentes posean la misma función hash. Luego, se le podría dar un hash correcto a un archivo malicioso.

En el 2004 se publicó como obtener colisiones hash con MD5 [47], sin embargo uno de los archivos carecía de significado. Desde entonces se han ido mejorado estos métodos logrado generar colisiones hash en MD4, MD5 y en SHA1 [41].

2.2.2.4.4 ROOTED BOX

La mayoría de los buenos hackers no solo se centralizan en realizar ataques a la máquina objetivo y en su defecto en la preocupación por no dejar evidencia, sino también a realizarlos desde máquinas controladas remotamente. Estos hackers pueden tener en el peor de los casos, una serie de máquinas infectadas y preparadas para realizar nuevos ataques.

Para este tipo de ataques, se suben las herramientas necesarias a la máquina remota para luego usarlas como ataques a otros sistemas. Por ejemplo se puede usar metasploit framework para ingresar a otra máquina y en esta usar el mismo metasploit como un payload[10] para ejecutarlo y posteriormente lanzar exploits remotamente. Así pues cuando el investigador intente analizar de dónde provino el ataque será engañado o confundido sobre el real culpable de los hechos.

2.2.3 ANÁLISIS DE TÉCNICAS ANTI-FORENSES EN ZFS

ZFS es un sistema de archivos que no tiene mucho en común con los sistemas de archivos tradicionales. Su diseño mediante objetos, el pool de almacenamiento común, la localización y método para llegar al contenido del disco son nuevas nociones que lo convierte en un nuevo paradigma para el diseño de sistemas de archivos.

Las técnicas anti-forenses entre sistemas de archivos suelen diferir en herramientas y métodos, pero siempre conservan los mismos objetivos y principios. Si lo anterior es correcto, intentar aplicar estos objetivos y principios será un nuevo reto para la investigación forense en ZFS.

A continuación comenzaremos analizando como podrían concebirse las técnicas anti-forenses en ZFS, relacionándolas con el nuevo paradigma, diseño e implicaciones forenses ya estudiadas en este trabajo.

Iniciamos tal vez con la característica más interesante para el investigador que es paradigma COW. Este paradigma permite a ZFS lograr las instantáneas, los clones y los ditto blocks y en general tener copias de metadatos y datos en espacios asignados y no asignados. Lo que implicará que el atacante que desee

[10] Acción a ejecutarse después de explotar el servicio vulnerable.

eliminar y sobrescribir archivos sin dejar trazas o residuos, tendrá que realizar un trabajo de mayor cautela contratando con el que se realizaba en los sistemas de archivos tradicionales.

Las sumas de comprobación (fletcher2, fletcher4 o SHA256) que se efectúan para garantizar la integridad de los datos y metadatos, lo que se traduce en que cualquier cambio ejecutado directamente en el disco será identificado automáticamente por ZFS. Lo que significa que ésta funcionalidad probablemente estará presente en la mente de los atacantes a la hora de modificar metadatos y datos.

Otra funcionalidad que probablemente estará presente en la mente de los atacantes es el ZIL (*"ZFS Intent Log"*) ya que éste registra todas las transacciones del sistema de archivos, lo que debe ser muy útil para el investigador, por lo que el atacante pensaría en la posibilidad de desactivarlo o en general alguna técnica que le permita vulnerar el examen forense.

Otra característica y novedad es la compresión transparente al usuario, que tiene como propósito maximizar el espacio disponible en disco, lo que se traduce en un desafío en la indexación de datos para su posterior análisis por parte de la investigación forense; de igual manera si ZFS comienza a soportar cifrado transparente de datos, se convertiría en otro desafío más para los investigadores, ya que la compresión y el cifrado son técnicas utilizadas con propósitos de ocultación de la información.

Adicionalmente y con el fin de ocultar información, encontramos el slack space (ver 2.2.2.2.2), que en ZFS se ve reflejado en los FSB ("*File System Blocks*") (ver 2.1.3.2.5) que proporcionan mediante los tamaños dinámicos una menor posibilidad de utilizar el slack space como se hacía por ejemplo en NTFS. Los archivos más pequeños que el tamaño del registro del FSB (128 KB por defecto) poseen un slack space casi inútil además de suponer que está ocupado de ceros; para los archivos más grandes en muchos de los casos, presentaran una cantidad significativa de slack space que usualmente no será tan amplia como en otros sistemas de archivos [2].

Luego, existe la posibilidad de ocultar información en el slack space mediante archivos muy grandes. Además de la posibilidad de ocultar información en áreas y espacios no usados o reservados como podrían ser el "*Boot Block*" (ver Ilustración 1) o el "*padding*" en el puntero de bloque (ver Ilustración 16).

Finalmente puede que la gran capacidad de almacenamiento de ZFS pueda repercutir en la generación de grandes cantidades de material que dificulte el examen forense.

3 MODELO PROPUESTO DE LA INVESTIGACIÓN

3.1 INTRODUCCIÓN

Este capítulo trata de un modelo de aplicación de técnicas anti-forenses en general, para luego especificar para cada uno de los métodos anti-forenses, dónde se puede aplicar en el sistema de archivos ZFS y por último el planteamiento de una metodología que nos indique cómo y qué tipo de pruebas se implementaran posteriormente.

3.2 MODELO DE APLICACIÓN DE TÉCNICAS ANTI-FORENSES

Un atacante que no quiera dejar rastros o pruebas en disco, tiene claro cuál es el objetivo, como por ejemplo: cómo hago para eliminar del todo un archivo; o cómo hago para ocultar la evidencia o mejor aún cómo hacer para no crearla. Un investigador por el contrario, no tiene unos objetivos muy claros, como por ejemplo: cómo hago para encontrar evidencia en el disco, cómo se que no me engañaron, cómo pruebo que me engañaron o cómo se que no existe evidencia.

Es por esto que el investigador tiene un arduo trabajo, descubriendo en el camino nuevas y diferentes técnicas usadas por los maliciosos que lo desconciertan y lo desmotivan. Pero ésta no es razón para renunciar, al contrario son acontecimientos para abrir los ojos y como lo plantea Harris: tomar tiempo para replantear nuestra visión forense y crear nuevas formas de atacar el problema de raíz.

Para realizar un estudio de las técnicas anti-forenses en un sistema de archivos, vemos que lo primordial es organizar y categorizar todas las técnicas conocidas para que de alguna manera sean tangibles. Luego, la investigación se enfocará en plantear un modelo de aplicación de técnicas anti-forenses en general. Para este fin analizaremos cada uno de los métodos:

3.2.1 DESTRUCCIÓN DE LA EVIDENCIA

Para materializar este método, primero debemos analizar dónde se encuentra la evidencia en el disco y si es posible accederla sin usar la capa del sistema de archivos, para luego eliminarla completamente de tal manera que el investigador y los programas forenses no encuentren rastro alguno.

Esto es básicamente conocer el mecanismo que utiliza el sistema de archivos para borrar los datos y luego buscar la manera de borrar los datos de manera irrecuperable y que no dejen residuos que pueda evidenciar que fueron borrados.

1. Identificar dónde se encuentra el archivo objetivo en el disco.
2. Sobrescribir el archivo con algún algoritmo destructor.
3. Identificar cuáles son las trazas que deja el archivo.
4. Eliminar las trazas que deja el archivo.

3.2.2 OCULTAR LA EVIDENCIA

Para este método lo que el atacante busca, son los puntos ciegos del investigador sin comprometer el sistema de archivos para lograr camuflar información. Estos puntos ciegos se reflejan básicamente en las áreas del disco como el slack space, los espacios no asignados y en general las áreas que no son usadas y que contengan estructuras de datos, como archivos metadatos, journals, entradas de directorios, espacios reservados, etc.

Para materializar este tipo de técnicas hay que tener en cuenta lo siguiente:

- Analizar áreas en que está organizado el disco que puedan alojar datos.
- Analizar áreas y espacios no usados o reservados que contengan estructuras de datos.
- Encontrar vulnerabilidades en el sistema de archivos o en archivos que permitan esconder información sin alterar el comportamiento de estos.
- Cifrar y comprimir datos.

3.2.3 ELIMINACIÓN DE LAS FUENTES DE LA EVIDENCIA

Este método pretende no crear evidencia, es decir evitar que se escriba en el disco. Esto significa que los procesos, programas y datos que se requieran para el ataque deben ejecutarse y operarse desde la memoria. Esto se puede lograr de dos maneras:

1. Desde un bootable live CD o USB sin permisos de escritura sobre el disco.
   El sistema operativo del CD o USB usa la memoria para almacenar el sistema de archivos con el propósito de no tocar el disco.

2. Desde una conexión remota.

Se utiliza una conexión remota que permita ejecutar binarios, manipular procesos y recursos en la máquina objetivo. El procedimiento en general hacia la máquina objetivo es el siguiente:

1. Establecer una conexión.
2. Ganar acceso a un espacio de dirección en memoria.
3. Subir los binarios
4. Cargar los binarios
5. Controlar la ejecución de los binarios remotamente.

Cabe mencionar que estas técnicas primordialmente vulneran el análisis de disco, pero la dificultad aumenta cundo se realizan adicionalmente análisis de memoria y de red o de dispositivos externos.

3.2.4 FALSIFICACIÓN DE LA EVIDENCIA

Las formas en las que se puede engañar y confundir a los investigadores se refleja en el ingenio y la creatividad de los atacantes. Este método es el más astuto pero el menos ético ya que se puede llegar a inculpar a personas inocentes. El método va desde cambiar la fecha y hora de un archivo hasta lanzar ataques desde varios equipos remotos.

Los atacantes pueden tener diferentes propósitos para ejercer este método:

1. Tratar de desviar la dirección de las pruebas.
   Para lograr este propósito la mayoría de las veces los atacantes se valen de la edición y modificación de las pruebas para invalidar la verdadera evidencia. Las técnicas en general se pueden traducir en:
   - La modificación de metadatos.
   - La modificación de la extensión y encabezado de los archivos.
   - Colisiones Hash.
   - Ataques desde una máquina controlada remotamente.

2. Incriminar a una persona especifica.
   El atacante tiene la firme intención de culpar a una persona, simplemente infectándolo de evidencia culposa como por ejemplo:
   - Pistas falsas
   - Sembrar firmas de virus

- Sospechosas palabras clave
- Usar cuentas de otras personas
- Ataques desde una máquina controlada remotamente.

3. Atacar al analista

   Este enfoque intenta dificultar el trabajo del investigador de tal manera que se tome mucho más del tiempo reglamentario para realizar un análisis. Como por ejemplo:

   - Grandes cantidades de material y archivos dispersos.

3.3 ANÁLISIS DE TÉCNICAS ANTI-FORENSES EN ZFS

Las técnicas anti-forenses entre sistemas de archivos suelen diferir en herramientas y métodos, pero siempre conservan los mismos objetivos y principios. Si lo anterior es correcto, intentar aplicar estos objetivos y principios será un nuevo desafío para la investigación forense en ZFS.

A continuación comenzaremos analizando como podrían concebirse el modelo de técnicas anti-forenses en general, en relación con el nuevo paradigma, diseño y arquitectura de ZFS.

3.3.1 DESTRUCCIÓN DE LA EVIDENCIA

En este método lo primero que se debe analizar, es dónde se encuentra la evidencia en el disco, por lo que hay que comprender como el sistema de archivos accede al contenido de los datos en el disco para lograr accederlos sin usar la capa del sistema de archivos. Finalmente, se procede a eliminar completamente la evidencia de tal manera que el investigador y los programas forenses no encuentren rastro alguno de esta evidencia.

A continuación identificamos las consideraciones y herramientas a tener en cuenta para aplicar este método a ZFS:

1. Identificar dónde se encuentra el archivo objetivo en el disco.

El ZFS debugger (ZDB) nos permitirá examinar estructuras como el *uberblock*, el puntero de bloque, objetos, grupos de objetos y objetos ZAP, entre otros; que nos proporcionarán la información necesaria para determinar la dirección física (*offset*) en términos de sectores (bloques de 512 bytes) de los contenidos de un archivo en el disco [28]. Con esta herramienta también podemos obtener fragmentos de datos del disco con contenido de un archivo, sin modificar datos y metadatos de este. Lo cual es muy útil

para el investigador que desee examinar el disco evitando que el sistema de archivos modifique los metadatos del archivo [48]. Para el caso del atacante, esta herramienta será útil para llegar a los datos en los sectores del disco.

Por otro lado, el paradigma COW es el responsable de las instantáneas, los clones, los "Ditto blocks" y en general de crear copias de metadatos y datos en espacios asignados y no asignados, lo que significa que al identificar dónde se encuentra el archivo objetivo, también será necesario identificar si existen más copias de este en diferentes instantes de tiempo.

2. Sobrescribir el archivo con algún algoritmo destructor.

Esto se realiza accediendo directamente al disco mediante la dirección física (*offset*) donde se encuentra el archivo, reemplazando los datos con ceros, números, caracteres aleatorios, etc. Es decir, empleando algún método o estándar para la destrucción de datos como lo son: *Quick Erase, Peter Gutmann, DoD 5220.22-M* y *Pseudo Random Number Generator*, entre otros[34] [35] [38].

3. Identificar cuáles son las trazas que deja el archivo.

Revisando cual es el procedimiento para llegar a un archivo en el disco, identificamos que este es referenciado por el puntero de bloque del directorio raíz (*dnode_phys_t*), que se encuentra en el objeto ZAP del Master Node (*dnode_phys_t*) [30] [48]. Luego, que estos objetos son trazas que deja el archivo sobrescrito.

4. Eliminar las trazas que deja el archivo.

Esto se completa en la medida que las trazas identificadas (objetos de tipo *dnode_phys_t*) sean destruidas con éxito.

3.3.2 OCULTAR LA EVIDENCIA

En ZFS encontramos que para almacenar el contenido de los archivos en las unidades lógicas de asignación[11], utiliza los FSB ("*File System Blocks*") que proporcionan tamaños dinámicos. Los archivos más pequeños que el tamaño del registro del FSB (128 KB por defecto) poseen un slack space casi inútil además de suponer que está ocupado de ceros; para los archivos más grandes en muchos de los casos,

[11] Sector lógico, o bloque o cluster o unidad de asignación. Es una agrupación de sectores contiguos y es el espacio mínimo que va a ocupar un fichero. Lo "dibuja" y maneja el S.O. Su tamaño depende del Sistema de Archivos [56].

presentaran una cantidad significativa de slack space que no será tan amplia como en otros sistemas de archivos [2].

Con respecto a las áreas en las que ZFS organiza el disco, encontramos el "*Boot Block*" y el "*Blank space"* como espacios reservados (ver sección 2.1.2.2.1 y 2.1.2.3.1) que podrían usarse para ocultar información. Asimismo, examinando áreas al estilo FIST (*Filesystem Insertion & Subversion Technique*), identificamos el campo "*padding*" que es un espacio reservado en la estructura de datos del puntero de bloque (*blkptr_t*).

Por otro lado, una característica y novedad en ZFS es la compresión transparente al usuario, que tiene como propósito maximizar el espacio disponible en disco, lo que significa un desafío en la indexación de datos para su posterior análisis por parte de la investigación forense [2]; de igual manera si ZFS llegará a soportar cifrado transparente de datos, se convertiría en otro desafío más para los investigadores, ya que la compresión y el cifrado de datos se pueden usar con propósitos para ocultar la evidencia.

3.3.3 ELIMINACIÓN DE LAS FUENTES DE LA EVIDENCIA

La eliminación de las fuentes de la evidencia lo que pretende básicamente es no crear evidencia. En ZFS el ZIL (*"ZFS Intent Log"*) registra todas las transacciones del sistema de archivos, lo que debe ser muy útil para el investigador. El atacante que pretenda no crear evidencia deberá deshabilitarlo o intentar operar en memoria ejecutando binarios remotamente sin almacenar datos en el disco.

3.3.4 FALSIFICACIÓN DE LA EVIDENCIA

Este método funciona desde modificar la fecha y hora de un archivo hasta lanzar ataques desde varios equipos remotos. Existen diferentes propósitos para efectuar este método:

1. *Tratar de desviar la dirección de las pruebas.*

Esto es la edición y modificación de las pruebas para invalidar la verdadera evidencia. En ZFS se efectúan sumas de comprobación (fletcher2, fletcher4 o SHA256) para garantizar la integridad de los datos y metadatos. Lo que significa que la modificación de metadatos y encabezados de los archivos o en general cualquier cambio efectuado directamente en el disco, será identificado automáticamente por ZFS. Luego, si se desea atacar los atributos de tiempo de un archivo por ejemplo, primero se debe encontrar la manera de vulnerar las sumas de comprobación para poder realizar modificaciones en los metadatos sin ser identificados. Por otro lado, el *uberblock* es el punto de inicio para acceder a la totalidad de los datos en el

pool y no tiene sumas de comprobación, tiene muchas copias debido al modelo COW y a las copias de las etiquetas en el dispositivo virtual (ver Ilustración 2. Componentes de la etiqueta de un dispositivo virtual. Tomado de). Luego, un análisis del *uberblock* aportaría información relevante para el investigador, que de ser modificada desviaría la dirección de las pruebas. Por ejemplo modificar el campo *"ub_timestamp"* (ver Tabla 7) que provee al investigador información acerca de la última escritura en el *uberblock*.

2. Incriminar a una persona especifica.

El atacante tiene la intención de inculpar a una persona por medio de evidencia falsa. En ZFS todos los objetos contienen la estructura "*znode_phys_t"* en el campo "*dn_bonus"* del dnode (ver Tabla 10), la cual almacena los atributos del objeto como: marcas de tiempo, tamaño, propietario, privilegios de acceso del usuario, entre otros. Para el investigador estos atributos son de utilidad en el momento de analizar la evidencia, sin embargo no es tan confiable ya que se podría falsificar por ejemplo la identidad del usuario cambiando el propietario del archivo.

3. Atacar al analista.

Este enfoque intenta dificultar el trabajo del investigador de tal manera que se tome mucho más del tiempo reglamentario para realizar un análisis. La gran capacidad de almacenamiento en ZFS, puede repercutir en la generación de grandes cantidades de material que dificulte el examen forense.

3.4 MODELO PARA IMPLEMENTAR TÉCNICAS ANTI-FORENSES EN ZFS

Para cada uno de los métodos anti-forenses, se llevaran a cabo una serie de pruebas sobre el sistema de archivos ZFS. Para esto se planea utilizar la siguiente metodología [49]:

3.4.1 CONSTRUIR UN VOLUMEN DE PRUEBA

Las pruebas se van a trabajar sobre una imagen de un disco con el sistema de archivos ZFS con la finalidad de evitar daños al sistema de archivos en el momento de realizar las pruebas.

Una vez las pruebas se han realizado sobre la imagen del disco se procede a montar la imagen para comprobar el si las pruebas fueron un éxito para posteriormente analizar los resultados.

3.4.2 VERIFICAR LA INFORMACIÓN DEL VOLUMEN

Antes de comenzar a ejecutar las pruebas, hay que verificar cuales son la característica iniciales del volumen así como del sistema de archivos ZFS, con el fin de poder utilizar esta información en las pruebas y lograr verificar y analizar los resultados.

3.4.3 MODIFICAR EL VOLUMEN

Esto es la ejecución de las pruebas, donde se utilizaran diferentes herramientas y técnicas para realizar modificaciones en el volumen. Las pruebas que se identificaron para realizar son las siguientes:

3.4.3.1 DESTRUCCIÓN DE LA EVIDENCIA

En este método se pretende realizar un borrado seguro de un archivo mediante el proceso de limpieza denominado “wiping” (ver sección 2.2.2.1.2), el cual tiene en cuenta dos aspectos: sobrescribir los datos y borrar las huellas que deje el archivo. Luego, para desarrollar esta técnica en ZFS se van a tener las siguientes consideraciones:

1. Sobrescribir los datos reemplazándolos con ceros (0x0).
2. Borrar el puntero de bloque del directorio raíz (*dnode_phys_t*), que se encuentra en el objeto ZAP del Master Node (*dnode_phys_t*) (ver sección 2.1.3.3).

3.4.3.2 OCULTAR LA EVIDENCIA

Para este método nos enfocaremos en encontrar áreas en donde sea posible ocultar información en ZFS. Las áreas que identificamos en ZFS para realizar las pruebas son:

1. Slack space: Se utilizara un archivo de gran tamaño para tener una cantidad significativa de slack space para posteriormente almacenar cualquier información en este espacio.
2. Espacios reservados:
    a. *Boot Block* (ver sección 2.1.2.2.1).
    b. *Blank space* (ver sección 2.1.2.3.1).

3.4.3.3 ELIMINACIÓN DE LAS FUENTES DE LA EVIDENCIA

El ZIL (*"ZFS Intent Log"*) registra todas las transacciones del sistema de archivos. Luego, las pruebas que se realicen en este método estarán encaminas a desactivar esta funcionalidad.

3.4.3.4 FALSIFICACIÓN DE LA EVIDENCIA

Para este método se pretende invalidar la verdadera evidencia cambiando las marcas de tiempo con la finalidad de confundir al investigador. En ZFS el *uberblock* es como el puntero maestro, el punto de inicio para acceder a la totalidad de los datos en el pool; en otras palabras sin *uberblock* no hay acceso a los datos, razón por la cual su análisis aportara información relevante al investigador. Las pruebas que se realizaran estarán encaminadas a modificar el campo "*ub_timestamp"* (ver Tabla 7) que le provee al investigador información acerca de la ultima escritura en el *uberblock*.

3.4.4 VERIFICAR Y ANALIZAR EL RESULTADO DE LAS PRUEBAS

Luego de ejecutar cada prueba, se procede a verificar y analizar los resultados con ayuda de algunas herramientas y procedimientos para establecer si las pruebas fueron exitosas o si se presentaron inconvenientes o limitaciones.

3.4.5 CONCLUSIONES DE LOS RESULTADOS

Finalmente se muestran las conclusiones obtenidas luego de verificar los resultados, mostrando las implicaciones en la computación forense y que procedimientos podrían detectar las técnicas anti-forenses.

4 APLICACIÓN DEL MODELO PROPUESTO

En este capítulo se va a detallar el contexto general de las pruebas a realizar, siguiendo el modelo planteado anteriormente. Adicionalmente para cada una de las pruebas que se plantearon, se detallará cómo se hicieron en la práctica sobre el sistema de archivos ZFS. En resumen este capítulo pretende mostrar cómo fue la aplicación del modelo.

4.1 IMPLEMENTANDO TÉCNICAS ANTI-FORENSES EN ZFS

A continuación se detallará y se aplicará el modelo propuesto:

4.1.1 CONSTRUIR UN VOLUMEN DE PRUEBA

Para comenzar, se debe crear una imagen de un disco con el sistema de archivos ZFS. Para ésto se utiliza el software de vitalización:

1. VirtualBox Graphical User Interface Version 3.1.8_OSE

Esta herramienta nos permitirá crear un disco duro virtual con el sistema de archivos ZFS instalando el sistema operativo:

2. OpenSolaris 2009.06 x64

Una vez instalado el sistema operativo en la imagen de un disco duro virtual de tamaño fijo y no dinámico[12] con extensión *"Virtual Disk Image"* (VDI), se procede a crear una copia de esta imagen, con la finalidad de tener una copia donde podamos realizar cómodamente las pruebas sin alterar la imagen original. Para este fin, no es suficiente copiarla ya que VirtualBox asigna un número de identificación único (UUID) para cada imagen de un disco [50]. Luego, utilizamos el comando vboxmanage con el parámetro *clonehd* para crear una copia con otro número de identificación:

```
vboxmanage clonehd /directorio/original.vdi /directorio/copia.vdi
```

A continuación se convierte la imagen del disco duro virtual con extensión VDI a una imagen del tipo RAW (dd) mediante el siguiente comando:

[12] Las imágenes de tamaño dinámico a diferencia de las imágenes de tamaño fijo, eliminan los bloques no usados, es decir los bloques que contengan íntegramente ceros.

```
vboxmanage internalcommands converttoraw file.vdi file.raw
```

También es posible crear la imagen directamente del disco duro virtual original sin necesidad de crear la copia.

Las imágenes RAW se usaran para restaurar los discos duros virtuales con la finalidad de no tener que volver a instalar todo de nuevo y que la información inicial del volumen siempre sea consistente. Para montar las imágenes, primero se debe convertir la imagen RAW a VDI con tamaño fijo y no dinámico mediante el siguiente comando:

```
vboxmanage convertdd --variant fixed file.raw file.vdi
```

Los discos virtuales VDI se usaran para ejecutar las pruebas directamente. Es importante tener presente que la única deferencia entre la imagen y el disco duro virtual es que de un encabezado asociado al VirtualBox en este último. El resultado que se espera al final es:

Nombre	Tipo	Tamaño	Descripción
SolarisV	VDI	3,794,959 KB	Imagen del disco duro original.
SolarisR	RAW	3,794,944 KB	Copia de la imagen original sin formato.

Adicionalmente, para las diferentes pruebas se van a crear *pools* con diferentes tamaños. El procedimiento para crearlos y destruirlos es el siguiente:

1. Crear un disco duro virtual de un tamaño específico mediante VirtualBox.
2. Iniciar el sistema operativo y en la línea de comandos examinar cual es el identificador del el dispositivo virtual (*vdev*) que vamos a agregar en el *pool*:

```
# format
Searching for disks...done

AVAILABLE DISK SELECTIONS:
```

```
       0. c7d0 <DEFAULT cyl 3324 alt 2 hd 128 sec 32>
          /pci@0,0/pci-ide@1,1/ide@0/cmdk@0,0
```

```
1. c7d1 <DEFAULT cyl 1021 alt 2 hd 64 sec 32>
   /pci@0,0/pci-ide@1,1/ide@0/cmdk@1,0
```

3. Crear un *pool* con el nombre de este y con el dispositivo virtual elegido:

```
# zpool create nombrepool c7d1
```

4. Verificar el estado del nuevo *pool:*

```
# zpool status

  pool: nombrepool
 state: ONLINE
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        nombrepool  ONLINE       0     0     0
          c7d1      ONLINE       0     0     0

errors: No known data errors

  pool: rpool
 state: ONLINE
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        rpool       ONLINE       0     0     0
          c7d0s0    ONLINE       0     0     0

errors: No known data errors
```

5. Destruir el *pool:*

```
# zpool destroy nombrepool c7d1
```

4.1.2 VERIFICAR LA INFORMACIÓN DEL VOLUMEN

Antes de comenzar a ejecutar las pruebas, se verificaran las características iniciales del sistema de archivos ZFS, con el fin de poder utilizar esta información posteriormente en los análisis y demás.

Una vez iniciado el sistema operativo OpenSolaris procedemos con el siguiente comando a mostrar la lista de *pools* configurados en el sistema:

```
# zpool list
```

Este nos muestra decir que tenemos dos *pools* en el sistema, el *pool* raíz y el *pool* creado con el nombre "antizfs".

Para dar una clara información del volumen, se pretenden revisar las siguientes estructuras y atributos de los *pools*, escribiendo en la línea de comandos como administrador:

1. Para ver la colección de pares nombre-valor de la etiqueta del dispositivo virtual (ver secci
2. ón 2.1.2.3.3):

```
# zdb <pool>
```

3. Para ver la estructura "*dsl_dir_phys_t"* de el objeto *DSL Directory* (ver sección 2.1.2.6):

```
# zfs get all <pool>
```

4. Para ver la información del ZIL asociado a cada *dataset*:

```
# zdb -iiii <pool>
```

5. Para ver un reporte de *datasets*, objetos y estadísticas de entrada y salida del zdb.

```
# zdb -v <pool>
```

6. Para recorrer todos los bloques y verificar las sumas de comprobación.

```
# zdb -c <pool>
```

Estos comandos seran utilizados para verificaran las características iniciales del sistema de archivos dependiendo de la tecnica a implementar.

4.1.3 MODIFICAR EL VOLUMEN

Esto es la ejecución de las pruebas, donde se utilizaran diferentes herramientas y técnicas para realizar modificaciones en el volumen. Las pruebas que se identificaron para realizar son las siguientes:

4.1.3.1 ELIMINACIÓN DE LAS FUENTES DE LA EVIDENCIA

El ZIL (*"ZFS Intent Log"*) registra todas las transacciones del sistema de archivos, lo que debe ser muy útil para recopilar evidencias por parte del investigador, por ejemplo en el análisis de sistemas muertos. Luego, los atacantes cautelosos que pretendan no crear evidencia intentaran deshabilitarlo.

4.1.3.1.1 DESHABILITAR EL ZIL

Para deshabilitar el ZIL encontramos dos posibilidades usadas con el objetivo de optimizar el rendimiento de NFS[13] sobre ZFS. La primera opción es entrar a línea de comandos como administrador y escribir [51]:

```
# echo zil_disable/W0t1 | mdb –kw
```

Esto solo deshabilita temporalmente el ZIL hasta el reinicio. Para que esto tenga efecto es necesario desmontar (zfs umount) y montar (zfs mount) el sistema de archivos.

La segunda opción, es modificar el archivo "/etc/system" añadiendo la línea:

[13] Permite a los anfitriones montar particiones en un sistema remoto para usarlas como si fueran sistemas de archivos locales [58].

```
set zfs:zil_disable=1
```

El ZIL quedara deshabilitado luego de reiniciar. Luego la diferencia de cada opción es que la primera es temporal, es decir que el ZIL solo se deshabilita para una sesión y al reiniciar el sistema este queda habilitado; la segunda opción en cambio deshabilita el ZIL por tiempo indefinido hasta que el archivo "/etc/system" sea modificado para activarlo.

4.1.3.1.1.1 VERIFICAR Y ANALIZAR EL RESULTADO DE LAS PRUEBAS

Para verificar que el ZIL realmente se encuentra deshabilitado, tenemos que asegurar que no hay actividad en este. Para esto utilizamos un script que nos resume el tamaño de los datos que son enviados al ZIL en un intervalo [52]. Una vez descargado el script lo ejecutamos en la línea de comandos como administrador con los siguientes parámetros para que imprima con un intervalo de 60 segundos:

```
# ./zilstat.ksh -t 60
```

Cuando el ZIL está habilitado encontramos actividad en algunos instantes de tiempo:

TIME	N-Bytes	N-Bytes/s	N-Max-Rate	B-Bytes	B-Bytes/s	B-Max-Rate	ops	<=4kB	4-32kB	>=32kB
2010 Aug 26 10:38:16	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 10:39:16	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 10:40:16	9880	164	9880	147456	2457	147456	3	0	2	1
2010 Aug 26 10:41:16	103624	1727	93192	495616	8260	262144	7	0	1	6
2010 Aug 26 10:42:16	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 10:43:16	120224	2003	91096	929792	15496	700416	9	0	0	9
2010 Aug 26 10:44:16	156344	2605	120200	835584	13926	327680	9	0	0	9
2010 Aug 26 10:45:16	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 10:46:16	0	0	0	0	0	0	0	0	0	0

TABLA 1. ZIL HABILITADO.

Cuando el ZIL está deshabilitado no encontramos actividad alguna:

TIME	N-Bytes	N-Bytes/s	N-Max-Rate	B-Bytes	B-Bytes/s	B-Max-Rate	ops	<=4kB	4-32kB	>=32kB
2010 Aug 26 12:45:01	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 12:46:01	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 12:47:01	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 12:48:01	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 12:49:01	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 12:50:01	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 13:02:01	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 13:03:01	0	0	0	0	0	0	0	0	0	0
2010 Aug 26 13:04:01	0	0	0	0	0	0	0	0	0	0

TABLA 2. ZIL DESHABILITADO.

Después de asegurarnos que el ZIL está deshabilitado, procedemos a verificar los cambios ocurridos en ZFS y el volumen. Para esto utilizamos el ZFS debugger (ZDB)[14] que nos permite acceder a información del dispositivo virtual como metadatos.

Se revisara información del sistema y de estructuras de datos en el dispositivo virtual en los siguientes estados del ZIL:

1. Antes de deshabilitarlo o información del volumen (ver 9.3).
2. Después de deshabilitarlo (ver 9.4).
3. Después de habilitarlo (ver 9.5).

Para cada uno de estos estados se pretenden utilizar los comandos 1, 2 y 3 detallados en la sección 4.1.2, con el fin de examinar las estructuras y atributos del volumen. Los resultados de cada sesión se encuentran en los anexos (ver sección 9). Luego de obtener la información de cada estado, analizamos con ayuda de la herramienta *WinMerge* 2.12.4.0[15], que nos permitirá realizar comparaciones en cada estado.

Comenzamos analizando el antes y después de deshabilitar el ZIL (ver Ilustración 21), encontrando que después de deshabilitarlo los *datasets* ya no poseen información relacionada con el ZIL, es decir, no hay encabezado ZIL ni bloques. Sin embargo encontramos que persiste el encabezado ZIL y bloques para el *dataset: "*rpool/swap".

Cuando el ZIL vuelve a quedar activo, comparamos la información antes de habilitarlo (o lo que es igual, después de desahabilitarlo) con el después de habilitarlo (ver Ilustración 22). En este caso observamos que los *datasets* "rpool/ROOT/opensolaris" y "rpool/export/home/ferca" vuelven a tener encabezados ZIL y bloques, sin embargo para el dataset "rpool" ya no existe la información del encabezado ZIL ni de los bloques; información que si existía antes de deshabilitar el ZIL (ver

Ilustración 21).

4.1.3.1.1.2 CONCLUSIONES DE LOS RESULTADOS

El ZIL registra transacciones del sistema de archivos como crear, remover o renombrar un archivo o directorio, escrituras de archivo, ajustes en los atributos de archivo, etc. Luego, es muy útil para recopilar

[14] Permite diagnosticar fallas y obtener estadísticas de ZFS.
[15] Software de código abierto para Windows, que compara dos archivos presentando las diferencias en un formato de texto visual fácil de entender y manejar [59].

evidencias por parte del investigador como por ejemplo para determinar información eliminada recientemente.

Cuando los atacantes decidan deshabilitar el ZIL con la finalidad de evadir la generación de rastros en la máquina, el investigador podría determinar cuáles son los detalles propios del sistema de archivos en el momento de deshabilitar, como por ejemplo los *datasets* que no posean información de sus respectivos encabezados ZIL y bloques, pueden conducir a determinar si el ZIL está deshabilitado o fue deshabilitado por un periodo de tiempo.

4.1.3.2 OCULTAR LA EVIDENCIA

Las pruebas que se realizaran para ocultar información en ZFS se basaran en las siguientes técnicas:

4.1.3.2.1 OCULTAR EN ESPACIO NO ASIGNADO

Los espacios en el disco donde el sistema de archivos no ha asignado información, pueden ser usados para almacenar archivos de manera que el sistema de archivos n tenga noción de estos. Las siguientes pruebas estarán encaminadas en almacenar algún tipo de información en el espacio no asignado, para luego examinar que sucede en el sistema de archivos.

Para esta prueba crearemos un disco duro virtual de 1GB y lo agregamos a un nuevo *pool* llamado "antizfs" siguiendo el procedimiento explicado en la sección 4.1.1.

Para implementar esta técnica primero se identifica un espacio no asignado para después escribir algo que represente la información que podría almacenar un atacante. Esto nos conduce a examinar detalladamente la organización del disco duro e identificar donde están ubicadas exactamente las partes de un dispositivo virtual (ver sección 2.1.2.3) en el disco duro.

Para ubicar exactamente el *vdev* en el disco duro virtual de tipo VDI, vamos a ubicar la dirección física en el disco donde comienza el espacio de almacenamiento asignable, este es el que se encuentra justamente después de las etiquetas L1, L2 y el boot block (ver sección 2.1.2.2.1). Para esto utilizamos el ZFS debugger (ZDB)[16] que nos permite acceder a información del dispositivo virtual usando la opción R para extraer una representación hexadecimal de datos sin formato como texto ASCII en un archivo mediante el comando [30]:

[16] Permite diagnosticar fallas y obtener estadísticas de ZFS.

- zdb -R pool:vdev_specfier:offset:size> /archivo

En nuestro caso ponemos en *pool* el nombre del *pool* (antizfs), en vdev_specfier el identificador del vdev (c7d1, o puede ser cero si no hay mas *vdevs* en el *pool*), en *offset* la dirección física en términos de sectores, es decir en bloques de 512 bytes (ponemos cero porque queremos saber donde comienza el espacio de almacenamiento asignable) y en size el tamaño físico en bytes de los datos a extraer y en archivo donde queremos guardar los datos.

```
# zdb -R antizfs:0:0:2000> /tmp/rawdata
```

Ahora que sabemos cuáles datos están al comienzo del espacio de almacenamiento asignable (ver Ilustración 23), podemos encontrarlos en el disco duro virtual y de esta manera identificar cual es el offset exacto: 0x00421200

Es decir que vamos a ocultar información en un espacio no asignado después de 0x00421200. Por ejemplo, en 0x030D4000 vamos a escribir mediante un editor hexadecimal alguna información que represente la que podría almacenar un atacante en el disco duro virtual (ver Ilustración 24).

Para mayor detalle, la representación de los datos hexadecimales que identificamos en el dispositivo virtual los apreciamos en la Ilustración 6.

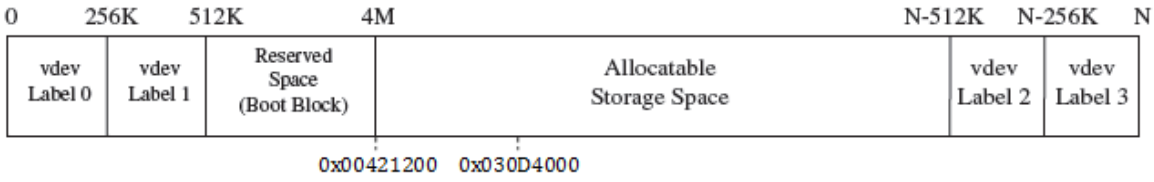

ILUSTRACIÓN 6. REPRESENTACIÓN DE LOS DATOS HEXADECIMALES EN EL DISPOSITIVO VIRTUAL.

4.1.3.2.1.1 VERIFICAR Y ANALIZAR EL RESULTADO DE LAS PRUEBAS

Lo primero que queremos verificar, es intentar extraer los datos que ocultamos utilizando el ZFS debugger usando la opción R . Para esto necesitamos saber el *offset* donde ocultamos la información, comenzando a contar después de las etiquetas L1, L2 y el boot block; es decir restando donde se encuentra la información oculta y donde comienza el espacio de almacenamiento asignable:

0x030D4000 - 0x00421200 = 0x2CB2E00

```
# zdb -R antizfs:0:2cb2e00:400> /tmp/rawdata
```

El resultado del comando es precisamente la información que habíamos ocultado inicialmente (ver Ilustración 25).

Ahora, verificaremos si existe alguna anomalía importante en el sistema de archivos, utilizando los comandos 4 y 5 detallados en la sección 4.1.2, con sus respectivos resultados los anexos (ver sección 9), que nos ayudaran a revisar los datasets, objetos, bloques y sumas de comprobación en los siguientes estados:

1. Antes de ocultar información (ver 9.6).
2. Después de ocultar información en espacios no asignados (ver 9.7).

Las diferencias de estos estados son normales y no hay anomalías (ver Ilustración 7). Son diferencias en marcas de tiempo, números de transacciones y en dos archivos de texto que creamos después en el *pool antizfs* con nombres: textfile1 y textfile2; para comprender y analizar mejor la información en los reportes.

ANTES

```
Dataset antizfs [ZPL], ID 16, cr_txg 1, 19.0K, 5 objects

    Object  lvl   iblk   dblk  lsize  asize  type
         0    7    16K    16K    16K  14.0K  DMU dnode

         1    1    16K    512    512     1K  ZFS master node
         2    1    16K    512    512     1K  ZFS delete queue
         3    1    16K    512    512     1K  ZFS directory
         4    1    16K    512    512     1K  ZFS directory
```

DESPUÉS

```
Dataset antizfs [ZPL], ID 16, cr_txg 1, 21.0K, 7 objects

    Object  lvl   iblk   dblk  lsize  asize  type
         0    7    16K    16K    16K  15.0K  DMU dnode

         1    1    16K    512    512     1K  ZFS master node
         2    1    16K    512    512     1K  ZFS delete queue
         3    1    16K    512    512     1K  ZFS directory
         4    1    16K    512    512     1K  ZFS directory
         5    1    16K    512    512    512  ZFS plain file
         6    1    16K    512    512    512  ZFS plain file
```

ILUSTRACIÓN 7. ANTES Y DESPUES DE OCULTAR INFORMACIÓN EN ESPACIOS NO ASIGNADOS.

Antes existían en el “Dataset antizfs” cinco objetos y ahora hay dos objetos adicionales del tipo “zfs plain file”, que concuerda con los textfile1 y textfile2 que creamos.

A continuación, vamos a experimentar que sucede si intentamos ocultar información en espacios asignados. Tenemos dos archivos (textfile1 y textfile2) con información cualquiera que previamente conocemos, los que nos permite buscarla con exactitud mediante el editor hexadecimal en el disco duro virtual y posteriormente modificarla. Vamos a modificar ambos archivos simplemente con la palabra “FAIL” como se muestra en la Ilustración 26 e Ilustración 27.

De igual manera, utilizamos los comandos 4 y 5 detallados en la sección 4.1.2 en el estado para verificar los resultados en el estado:

3. Después de ocultar información en espacios asignados (ver 9.8)

Analizamos las diferencias comparando con él antes de ocultar información (ver Ilustración 28).

Encontramos que el sistema de archivos identifica errores en las sumas de comprobación y además nos dice que objetos son los que poseen problemas. En este caso, los objetos número 5 y 6 poseen errores. En la Ilustración 29 podemos constatar que los objetos con este número son justamente el textfile1 y textfile2 que habíamos creado anteriormente. Estos archivos ya no se pueden acceder normalmente ya que si los intentamos abrir el sistema nos arrojara el error: "Unexpected error: I/O error". Podríamos usar el ZFS debugger usando la opción R para extraer los datos sin formato.

4.1.3.2.1.2 CONCLUSIONES DE LOS RESULTADOS

Ocultar información en espacios no asignados es posible, debido a que el sistema de archivos solo identifica errores en espacios asignados mediante las en las sumas de comprobación. Sin embargo, al ocultar información en espacios no asignados se corre el riesgo de la sobrescritura de la información debido a que el sistema de archivos no posee conocimiento de la existencia de esta y puede asignar espacios.

Los investigadores por su lado, podrán seguir usando rutinas de búsqueda indexadas con palabras claves, para lograr identificar exactamente donde se encuentra la evidencia. Sin embargo, los atacantes que quieran ir más lejos intentaran cifrar y/o comprimir los datos (ver sección 2.2.2.2.4) para evadir estas búsquedas indexadas, dificultando el trabajo del investigador.

4.1.3.2.2 OCULTAR EN ESPACIOS RESERVADOS

Estos espacios generalmente son vacios, no usados o están reservados para un uso futuro. El primero que identificamos para ocultar evidencia es un espacio de 3.5MB denominado "*Boot Block*" (ver sección 2.1.2.2.1) que es reservado para un uso futuro. El segundo que identificamos, es el "Blank Space" (ver sección 2.1.2.3.1) que son 8K vacios.

Esta prueba no contempla los archivos que se usaron anteriormente (textfile1 y textfile2), es decir que el sistema de archivos no tiene errores y todo funciona correctamente. Primero, vamos a encontrar el *offset* donde comienzan estos espacios para posteriormente ocultar información. Usaremos el *offset* 0x00421200

que es donde comienza el espacio de almacenamiento asignable para ubicarnos en el *vdev* (ver Tabla 3 e Ilustración 8).

Boot Block	Blank Space
3.5MB = 3670016 bytes = 0x00380000	4MB = 0x00400000
0x00421200 - 0x00380000 = **0x000A1200**	0x00421200 - 0x00400000 = **0x00021200**

TABLA 3. COMIENZO DE EL BOOT BLOCK, BLANK SPACE Y EL ESPACIO DE ALMACENAMIENTO ASIGNABLE.

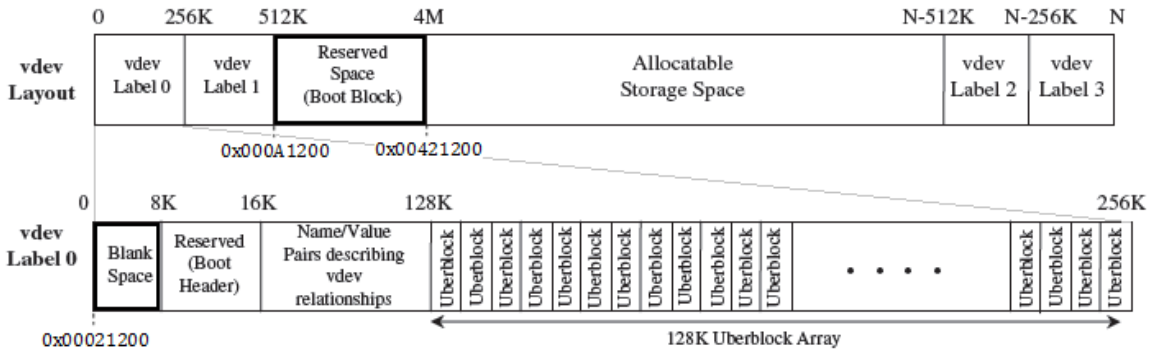

ILUSTRACIÓN 8. REPRESENTACIÓN DE LOS DATOS HEXADECIMALES EN EL DISPOSITIVO VIRTUAL.

Una vez se identificó el comienzo de cada espacio, procedemos a ocultar la información en la mitad de cada espacio, es decir:

Boot Block	Blank Space
3.5MB/2 = 1835008 bytes = 0x001C0000	8K/2 = 4096 bytes = 0x00001000
0x000A1200 + 0x001C0000 = **0x00261200**	0x00021200 + 0x00001000 = **0x00022200**

TABLA 4. DIRECCIÓN FÍSICA DONDE SE VA A OCULTAR LA INFORMACIÓN.

4.1.3.2.2.1 VERIFICAR Y ANALIZAR EL RESULTADO DE LAS PRUEBAS

Verificaremos si existe alguna anomalía importante en el sistema de archivos, utilizando los comandos 4 y 5 detallados en la sección 4.1.2, con sus respectivos resultados los anexos (ver sección 9), en los siguientes estados de igual manera que hicimos en los espacios no asignados:

1. Antes de ocultar información (ver 9.6).
2. Después de ocultar información en el *Boot Block* (ver 9.9).
3. Después de ocultar información en el *Blank Space* (ver 9.10).

Las diferencias Antes de ocultar información y después de ocultar información en el *Boot Block* y en el *Blank Space,* se encuentran en las marcas de tiempo, números de transacciones, etc; lo que significa que no hay anomalías después de ocultar la información. También encontramos que el sistema de archivos no posee errores en las sumas de comprobación como vemos en la Ilustración 32.

4.1.3.2.2.2 CONCLUSIONES DE LOS RESULTADOS

Ocultar información en espacios reservados es posible, debido a que el sistema de archivos no tiene ningún control sobre estos, es decir que nunca identificara estas anomalías. Sin embargo, al ocultar información en espacios reservados se corre el riesgo que se sobrescriba esta información, por ejemplo puede existir el caso en que el *Blank Space* tenga etiquetas de disco VTOC o EFI (ver sección 2.1.2.3.1).

Luego, los investigadores para contrarrestar, deben verificar si en estos espacios la información allí contenida es normal o hay sospechas de ocultamiento de la evidencia. Las rutinas de búsqueda indexadas con palabras claves, también serán útiles para identificar que se oculto evidencia en estos espacios. Sin embargo, los atacantes que quieran ir más lejos intentaran cifrar y/o comprimir los datos (ver sección 2.2.2.2.4) para evadir estas búsquedas indexadas, dificultando el trabajo del investigador.

4.1.3.2.3 ESCRIBIR EN EL SLACK SPACE

Para escribir en el *slack space* (ver 2.2.2.2.2) en ZFS, hay que entender que los FSB ("*File System Blocks*") (ver 2.1.3.2.5) permiten ajustar dinámicamente el tamaño de las unidades de asignación[17] o cluster, dependiendo del tamaño del archivo, a diferencia de otros sistemas de archivos. El funcionamiento de estos es de la siguiente manera:

Si el tamaño del archivo es menor que el FSB, la unidad de asignación será ajustada de la mejor manera en términos de los sectores. Por ejemplo, un archivo de 2560B será almacenado en 5 sectores (512B * 5 = 2560B) o un archivo de 2565B será almacenado en 6 sectores (512B * 6 = 3072B) con un *slack space* de 507B = 3072B - 2565B.

Si el tamaño del archivo es mayor que el FSB, la unidad de asignación será ajustada de la mejor manera en términos del FSB (por defecto esta en 128KB y es el tamaño máximo). Por ejemplo, un archivo de 640KB será almacenado en 5 *File System Blocks* (128KB * 5 = 640KB) o un archivo de 645KB será almacenado en 6 *File System Blocks* (128KB * 6 = 768KB) con un *slack space* de 123KB = 768KB - 645KB.

[17] Sector lógico, o bloque o cluster o unidad de asignación. Es una agrupación de sectores contiguos y es el espacio mínimo que va a ocupar un fichero. Lo "dibuja" y maneja el S.O. Su tamaño depende del sistema de archivos [56].

Luego, para efectos de las pruebas vamos a realizar cuatro casos que representaran las posibles formas en las que puede quedar almacenado un archivo en ZFS:

1. Archivos más pequeños que el tamaño del registro del FSB:
   Caso 1: sin *slack space*.
   Caso 2: con *slack sapce.*
2. Archivos más grandes que el tamaño del registro del FSB:
   Caso 3: sin *slack space.*
   Caso 4: con *slack sapce.*

Para esta prueba crearemos un disco duro virtual de 100MB y lo agregamos a un nuevo *pool* llamado "slack" siguiendo el procedimiento explicado en la sección 4.1.1.

Ahora, llenaremos todo el espacio disponible con un archivo de datos aleatorios para asegurarnos que todo el disco está asignado y después eliminaremos el archivo para que ahora el espacio sea no asignado.

```
# dd if=/dev/urandom of=/slack/fileAllMem bs=1M count=86

dd: writing `/slack/file86M': No space left on device
55+0 records in
54+0 records out
57409536 bytes (57 MB) copied, 13.4409 s, 4.3 MB/s

# rm /slack/fileAllMem
```

El archivo creado es de 57409536 bytes (54.75 MB), que ocupa todo el espacio disponible para almacenar archivos (nos lo asegura el mensaje "*No space left on device")*. Cuando se elimina el archivo, el disco queda de nuevo disponible con la diferencia de que los datos aún persisten en disco, es decir el espacio disponible para almacenar archivos no está ocupado de ceros sino de datos aleatorios, lo que nos ayudara a realizar posteriores análisis al crear nuevos archivos.

Ahora, crearemos un archivo para cada caso siguiendo los lineamientos de las posibilidades detalladas anteriormente para finalmente lograr ocultar información en el *slack space:*

Caso	Tamaño	Slack Space
1	2560B	0

2	2565B	507B
3	655360B (640KB)	0
4	660480B (645KB)	125952B (123KB)

TABLA 5. LOS CUATRO CASOS EN LAS SE ALMACENA UN ARCHIVO EN ZFS.

Los comandos respectivos para crear estos archivos son los siguientes:

```
# dd if=/dev/urandom of=/slackfiles/file1caso bs=512 count=5
5+0 records in
5+0 records out
2560 bytes (2.6 kB) copied, 0.000427666 s, 6.0 MB/s

# dd if=/dev/urandom of=/slackfiles/file2caso bs=513 count=5
5+0 records in
5+0 records out
2565 bytes (2.6 kB) copied, 0.00140304 s, 1.8 MB/s

# dd if=/dev/urandom of=/slackfiles/file3caso bs=128K count=5
5+0 records in
5+0 records out
655360 bytes (655 kB) copied, 0.152593 s, 4.3 MB/s

# dd if=/dev/urandom of=/slackfiles/file4caso bs=129K count=5
5+0 records in
5+0 records out
660480 bytes (660 kB) copied, 0.139692 s, 4.7 MB/s
```

Hay que notar que estos archivos aun no están almacenados en el *pool* “slack”, debido a que primero vamos a editarlos con un editor hexadecimal para ponerles una etiqueta al final del archivo con la finalidad de encontrar de manera simple donde comienza el *slack space*. Posteriormente, copiamos los archivos al *pool* “slack” y comenzamos a buscar las etiquetas al final de cada archivo e insertamos algún tipo de información en los que posean *slack space* (de antemano sabemos que son los casos 2 y 4):

4.1.3.2.3.1 VERIFICAR Y ANALIZAR EL RESULTADO DE LAS PRUEBAS

Como se había planeado en la Tabla 5, el caso 1 y el caso 3 ocupan exactamente la unidad de asignación, por lo que el *slack space* es nulo. La información que es alojada justamente después de estos archivos, son metadatos que reemplazaron los datos que pertenecían al primer archivo que habíamos almacenamos en el *pool* y que ocupaba todo el espacio disponible (54.75 MB).

En los casos 2 y 4, encontramos un *slack space* ocupado de ceros, lo que significa que los datos del archivo creado y eliminado al comienzo serán reemplazados por ceros. Respecto a la información oculta, verificaremos si existe alguna anomalía importante en el sistema de archivos, utilizando los comandos 4 y 5 detallados en la sección 4.1.2, con sus respectivos resultados en los anexos (ver sección 9).

En los resultados, encontramos que el sistema de archivos identifica errores en las sumas de comprobación y además nos dice que los objetos número 6 y 8 poseen errores. Estos objetos corresponden justamente a los archivos "file2caso" y el "file4caso" (ver Ilustración 37) que corresponden con los casos 2 y 4, que son los casos donde precisamente habíamos ocultamos información en el *slack space.*

Por otro lado, para extraer la información oculta podríamos usar el ZFS debugger usando la opción R y calculando el *offset* donde ocultamos la información de la misma manera que se realizo en la sección 4.1.3.2.1.1.

4.1.3.2.3.2 CONCLUSIONES DE LOS RESULTADOS

En ZFS la unidad de asignación puede ser ajustada después de la instalación, ya que cambia dinámicamente dependiendo de la necesidad del archivo. Luego, la cantidad de *slack space* dependerá del tamaño del archivo y del FSB; resumiéndolo en dos posibilidades:

- Si el tamaño del archivo es menor que el FSB, la unidad de asignación será ajustada de la mejor manera en términos de los sectores.
- Si el tamaño del archivo es mayor que el FSB, la unidad de asignación será ajustada de la mejor manera en términos del FSB.

El tamaño del registro del FSB está por defecto en 128 KB y es el valor máximo. Este valor se almacena en el "*dnode*" de un objeto en el campo "*dn_datablkszsec"* [2] (ver Tabla 10).

Entonces, los archivos más pequeños que el tamaño del registro del FSB poseen un *slack space* mínimo (menor que el tamaño del sector: 512B) y los archivos más grandes que el tamaño del FSB en muchos de los casos, presentaran una cantidad significativa de *slack space* (menor de 128KB). Luego, ocultar información en el *slack space* es posible, sin embargo, ZFS detecta estas anomalías mediante las sumas de comprobación e identifica los archivos defectuosos.

Es decir, que esta técnica no será efectiva en ZFS hasta que los atacantes logren vulnerar las sumas de comprobación. Por otro lado, los investigadores ya no se beneficiaran encontrando viejos contenidos de archivos debido a que el *slack space* está ocupado de ceros.

4.1.3.3 FALSIFICACIÓN DE LA EVIDENCIA

Las pruebas que se realizaran para falsificar la evidencia se enfocaran en cambiar las marcas de tiempo en el *uberblock*:

4.1.3.3.1 CAMBIANDO LAS MARCAS DE TIEMPO EN EL UBERBLOCK

El *uberblock* (ver sección 2.1.2.3.4) es el punto de inicio para acceder a la totalidad de los datos en el pool. Su análisis aportara al investigador información relevante como los instantes de tiempo en los que se escribieron las diferentes copias de *uberblocks* siguiendo el modelo COW (copia por escritura).

Para esta prueba, crearemos un disco duro virtual de 100MB y lo agregamos a un nuevo *pool* llamado "timestamp" siguiendo el procedimiento explicado en la sección 4.1.1.

Cambiar las marcas de tiempo en la estructura del *uberblock*, requiere como primer paso identificar cual es el campo en la estructura que aloja la información de la marca de tiempo: "*ub_timestamp"* (ver Tabla 7). Luego, en el *pool* "timestamp*"* creamos un archivo de texto cualquiera para efectos de las pruebas y reiniciamos el sistema varias veces hasta comprobar que si no se acceden los datos en el pool "timestamp", el *uberblock* activo no se va a actualizar y por consiguiente su estructura es la misma para cada reinicio. Para revisar la estructura del *uberblock* activo usamos el siguiente comando:

```
# zdb -v timestamp

Uberblock

        magic = 0000000000bab10c
        version = 14
        txg = 28
        guid_sum = 4466727873531178253
        timestamp = 1288576965 UTC = Sun Oct 31 21:02:45 2010
```

Ahora que sabemos que mientras no se accedan los datos del *pool* "timestamp", no se modificara el arreglo de uberblock lo que nos permitirá realizar pruebas y verificar posteriormente lo que sucede en el

sistema de archivos. Vamos a crear una copia del disco duro virtual en esta instancia con la finalidad de realizar dos pruebas:

1. Para un uberblock activo o con el número de grupo de transacción más alto.
2. Para un uberblock con un número de grupo de transacción más bajo.

Para la primera prueba y mediante búsquedas el editor hexadecimal, encontramos en el disco duro virtual el *offset* donde se encuentra el *uberblock* activo e identificamos cada uno de los campos de la estructura y nos enfocamos en el campo "*ub_timestamp"* para cambiar su valor actual de 1288576965 a la nueva marca de tiempo: 1288576964 (un segundo antes).

Deducimos que los valores hexadecimales en la estructura, se encuentran en formato *little endian* debido a que el campo *ub_magic* es igual a: 0c b1 ba 00 (ver ilustración ## y Tabla 7). Luego, vamos a cambiar las marcas de tiempo en el disco, con valores hexadecimales en formato *little endian* (ver Tabla 6) y procedemos a cambiar mediante el editor hexadecimal el campo "*ub_timestamp"* con el valor 0x c4 1f ce 4c (ver Ilustración 38).

Descripción	Decimal	Hexadecimal (*big endian*)	Hexadecimal (*little endian*)
Marca de tiempo original	**1288576965**	4c ce 1f c5	**c5 1f ce 4c**
Marca de tiempo falsificada	**1288576964**	4c ce 1f c4	**c4 1f ce 4c**

TABLA 6. MARCAS DE TIEMPO EN LITTLE ENDIAN.

Hay que notar que el *uberblock* activo tiene otras 3 copias idénticas debido a la redundancia que proporciona ZFS (ver sección 2.1.2.2.1). Estas copias también se modificaran de la misma manera para que los cambios sean consistentes.

Para la segunda prueba, usaremos la otra copia del disco duro virtual y cambiaremos la marca de tiempo de un *uberblock* con un grupo de transacción más bajo. Para esto, se abre el archivo de texto que creamos al comienzo en el pool "*timestomp*" para que el *uberblock* activo se actualice usando el modelo COW pasando de transacción 28 a 36:

```
#zdb -u uberblock

Uberblock
```

```
magic = 0000000000bab10c
version = 14
txg = 36
guid_sum = 4466727873531178253
timestamp = 1289223825 UTC = Mon Nov  8 08:43:45 2010
```

Ahora modificamos el mismo *uberblock* de la prueba anterior pero teniendo en cuanta que ahora ya no es el *uberblock* activo (ver Ilustración 39).

Hay que notar que este *uberblock* también tiene otras 3 copias idénticas debido a la redundancia que proporciona ZFS (ver sección 2.1.2.2.1). Estas copias también se modificaran de la misma manera para que los cambios sean consistentes.

4.1.3.3.1.1 VERIFICAR Y ANALIZAR EL RESULTADO DE LAS PRUEBAS

Para ambas pruebas, verificaremos si los cambios en las marcas de tiempo son persistentes y que le sucede al sistema de archivos. Para la primera prueba, iniciamos el sistema con el disco duro virtual donde modificamos el *uberblock* activo y en la línea de comandos revisamos la estructura del *uberblock* activo:

```
#zdb -u timestomp

Uberblock

   magic = 0000000000bab10c
   version = 14
   txg = 28
   guid_sum = 4466727873531178253
   timestamp = 1289139259 UTC = Sun Nov  7 09:14:19 2010
```

Vemos que el *uberblock* activo posee el mismo número de grupo de transacción, sin embargo la marca de tiempo no es la que esperábamos (0x c4 1f ce 4c), al parecer el sistema la actualizo al tiempo actual (0x 3b b4 d6 4c) para corregirla. Esto se realiza en el mismo *uberblock*, en el mismo *offset* (ver Ilustración 9); es decir, esto no se trata de una actualización del *uberblock* activo usando el modelo COW sino de una corrección del campo "*ub_timestamp"*.

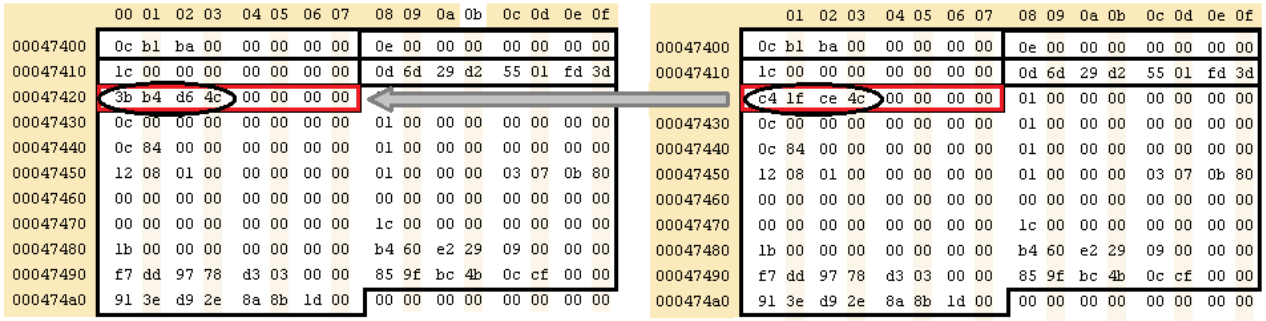

ILUSTRACIÓN 9. PERSISTENCIA DEL UBERBLOCK ACTIVO.

El *uberblock* por ser el punto de inicio, no protege la integridad de sus datos con sumas de comprobación, sino que realiza verificaciones como en el caso del campo "*ub_txg"*, con información codificada en los pares de Nombre-Valor (ver sección 2.1.2.3.3) en la etiqueta del dispositivo virtual [20]. Luego, la corrección de la marca de tiempo probablemente se realiza de la misma manera verificando con información codificada en la etiqueta del dispositivo virtual.

Ahora verificamos con el comando 4 (ver sección 4.1.2), y encontramos que no existe ninguna anomalía:

```
#zdb -v timestomp

... <-- output omitted
                    capacity   operations   bandwidth  ---- errors ----
description             used avail  read write  read write  read write cksum
timestomp               77.5K 86.9M    16     0  969K     0     0     0     0
 /dev/dsk/c7d1s0         77.5K 86.9M    16     0  969K     0     0     0     0
```

Para la segunda prueba, iniciamos el sistema con el disco duro virtual donde modificamos el *uberblock* con un número de grupo de transacción más bajo y verificamos mediante el editor hexadecimal el *offset* donde realizamos el cambio (ver Ilustración 10).

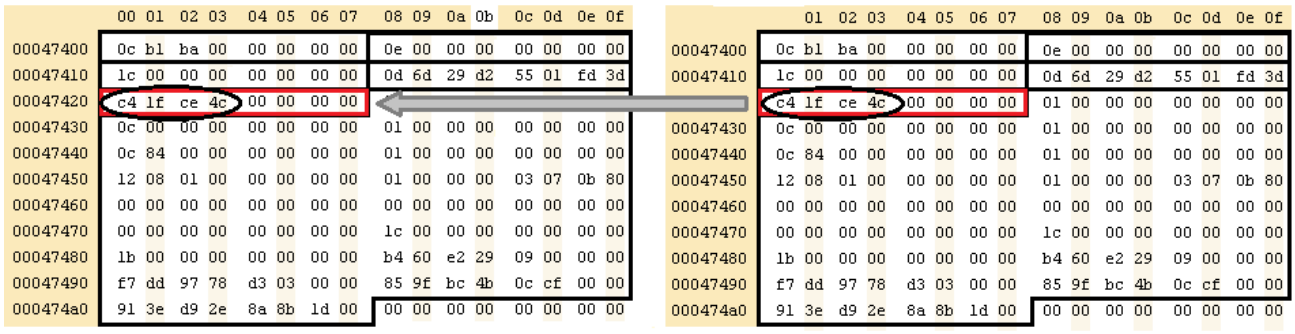

ILUSTRACIÓN 10. PERSISTENCIA DEL *UBERBLOCK* CON UN GRUPO DE TRANSACCIÓN MÁS BAJO.

Se comprueba que en este caso la marca de tiempo es persistente, debido a que solo en el *uberblock* activo se realizan verificaciones, es decir, en los demás *uberblocks* del arreglo no se realizan verificaciones por lo que es posible falsificar la información de estos. Si verificamos con el comando 4 (ver sección 4.1.2) encontramos que no existe ninguna anomalía:

```
# zdb -v timestomp

... <-- output omitted
                     capacity   operations   bandwidth  ---- errors ----
description             used avail  read write  read write  read write cksum
timestomp               77.5K 86.9M    16     0  969K     0     0     0     0
  /dev/dsk/c7d1s0         77.5K 86.9M    16     0  969K     0     0     0     0
```

4.1.3.3.1.2 CONCLUSIONES DE LOS RESULTADOS

Sin *uberblock* no hay acceso a los datos, es el puntero maestro, razón por la cual puede ser útil en el momento de invalidar evidencia cambiando las marcas de tiempo con la finalidad de confundir al investigador; la información contenida en este será analizada antes que a los datos en el pool, lo que de antemano permitirá saber cuál fue el último acceso a los datos en un pool o si los datos en un pool son consistentes.

El investigador al realizar un análisis, encontrara que tiene muchas actualizaciones en diferentes instantes de tiempo del *uberblock* debido al modelo COW y además tres copias del arreglo de *uberblock* en las etiquetas del dispositivo virtual, lo que le permitirá realizar líneas de tiempo sobre un caso específico.

El atacante que se disponga a falsificar el *uberblock* para desviar la dirección de las pruebas, podrá lograrlo en los *uberblocks* de menor número de transacción que el *uberblock* activo, debido a que el sistema realiza verificaciones únicamente en el *uberblock* activo con información codificada en los pares de Nombre-Valor (ver sección 2.1.2.3.3). Es decir que si es posible falsificar los *uberblocks* de menor número de transacción pero no el *uberblock* activo.

Sin embargo, habría una manera de falsificar el *uberblock* activo y es alterando las marcas de tiempo y no iniciar el sistema para que no se realice la verificación y de esta manera cuando el investigador analice la evidencia vea los datos alterados. Si el investigador decide iniciar el sistema y se corrigen las marcas de tiempo, este posiblemente piense que se trata de una simple actualización del *uberblock*, pero si es

cauteloso se dará cuenta que el número de transacción es el mismo y que la actualización no se realizo siguiendo el modelo COW.

Luego, es posible irrumpir en el proceso forense cuando el analista examine las marcas de tiempo para realizar líneas de tiempo coherentes. Las marcas de tiempo significan el último instante de tiempo en que el *uberblock* fue escrito, por lo que un *ubeblock* con mayor número de transacción debe tener una marca de tiempo mayor a uno con número de transacción menor.

4.1.3.4 DESTRUCCIÓN DE LA EVIDENCIA

Para este método se analizara como identificar la evidencia en el disco para posteriormente destruirla de tal manera que sea inútil para el proceso de investigación.

4.1.3.4.1 LIMPIAR ("*WIPING*")

Para esta prueba, crearemos un disco duro virtual de 100MB y lo agregamos a un nuevo *pool* llamado "wipefile" siguiendo el procedimiento explicado en la sección 4.1.1. Luego, crearemos un archivo de nombre "wipefilename" con algún contenido conocido para finalmente destruirlo.

En este pool desarrollaremos las pruebas y usaremos las consideraciones y herramientas mencionadas en la sección 3.3.1:

1. *Identificar dónde se encuentra el archivo objetivo en el disco.*

Para identificar dónde se encuentra el archivo objetivo en el disco, usaremos las versiones modificadas del zdb y el mdb por Max Bruning y el procedimiento expuesto en "ZFS On-Disk Data Walk (Or: Where's My Data)" [28] . Este procedimiento nos mostrara cual es el camino para llegar del uberblock activo del pool "wipefile" hasta el contenido del archivo "wipefilename". Los resultados los encontramos en los anexos (ver sección 9.12).

Al final del procedimiento encontramos un *dnode* que posee la referencia a el contenido del archivo mediante un puntero de bloque (blkptr_t) que se encuentra en un objeto de tipo "DMU_OT_PLAIN_FILE_CONTENTS" (dn_type = 0x13). Luego, La dirección física dónde se encuentra el contenido del archivo ("wipefiledata…") es 1de00:

```
# ./mdb /dnode1
> ::loadctf
> ::load /export/home/ferca/rawzfs.so
```

```
> a00::print -a -t zfs`dnode_phys_t
{
    a00 uint8_t dn_type = 0x13

            ... <-- output omitted

    a40 blkptr_t [1] dn_blkptr = [
        {
            a40 dva_t [3] blk_dva = [
                {
                    a40 uint64_t [2] dva_word = [ 0x1, 0xef ]
                }
                {
                    a50 uint64_t [2] dva_word = [ 0, 0 ]
                }
                {
                    a60 uint64_t [2] dva_word = [ 0, 0 ]
> a40::blkptr
DVA[0]: vdev_id 0 / 1de00
DVA[0]:       GANG: FALSE  GRID:  0000  ASIZE: 200
DVA[0]: :0:1de00:200:d
LSIZE:  200                     PSIZE: 200
ENDIAN: LITTLE                          TYPE:  ZFS plain file
BIRTH:  21              LEVEL: 0     FILL:  1
CKFUNC: fletcher2                   COMP:  uncompressed
CKSUM:  a5c5a0bdafc5a0b5:a5bb96b3a5bb96ab:1ec904c479d3842:258a381f04620741
> $q

# ./zdb -R wipefile:0:1de00:200:r
Found vdev: /dev/dsk/c7d1s0
wipefiledatawipefiledatawipefiledatawipefiledata
```

2. Sobrescribir el archivo con algún algoritmo destructor.

Esto se realiza accediendo directamente al disco mediante la dirección física (*offset*) obtenida en el paso anterior. Hay que tener en cuenta que esta dirección empieza donde comienza el espacio de almacenamiento asignable que es justamente después de las etiquetas L1, L2 y el boot block (ver sección 2.1.2.2.1). Luego, hay que sumar el *offset* 0x0001de00 y la dirección física del disco donde comienza el espacio de almacenamiento asignable 0x00420400 (este valor se obtuvo de la misma manera que se hizo en 4.1.3.2.1):

0x0001de00 + 0x00420400 = 0x0043E200

Ahora procedemos a eliminar el archivo desde el sistema operativo de la manera habitual y verificamos en la dirección física 0x0043E200 que aún persiste el contenido del archivo como era de esperarse, por lo que reemplazamos los datos con ceros (ver Ilustración 40) como representación de un algoritmo destructor (ver sección 2.2.2.1.2.1).

3. Identificar cuáles son las trazas que deja el archivo.

Para identificar cuáles son las trazas que deja el archivo usaremos las versiones modificadas del zdb y el mdb por Max Bruning siguiendo el procedimiento expuesto en "ZFS On-Disk Data Walk (Or: Where's My Data)" [28] (ver sección 9.12).

Analizando y resumiendo el procedimiento, vemos que solo hay un *Meta Object Set* (ver sección 2.1.2.6.1) por *pool* y el *uberblock* activo apunta a este directamente. En el *Meta Object Set* (MOS) se encuentra el objeto *Object Directory* (ver sección 2.1.2.6.2) que es un objeto ZAP que contiene el atributo "root_dataset" que identificara al *root DSL directory* del *pool*, que poseerá la referencia a todos los *datasets* en el pool.

Mediante los *datasets* encontraremos punteros de bloques con niveles indirectos (blkptr6, …, blkptr1) hasta llegar a un *dnode* (ver sección 2.1.2.5.1) que contiene la información del directorio raíz del archivo en el *Master Node* (dn_type = 0x15) y la referencia a el contenido del archivo mediante un puntero de bloque que se encuentra en un objeto de tipo “DMU_OT_PLAIN_FILE_CONTENTS” (dn_type = 0x13) (ver Tabla 9).

Luego, este *dnode* mencionado posee la información necesaria para que un investigador lo considere como un residuo o una huella del archivo borrado. Existen dos copias de este *dnode* debido a que el número de DVAs usados en el puntero del bloque es doble (ver Tabla 8) por políticas de ZFS para la protección de los datos. El primer *dnode* se encuentra en el *offset* 0x00020a00 y el segundo en 0x0010a060:

```
# ./zdb -R wipefile:0:21000:400:d,lzjb,4000 2>/blkptr1
Found vdev: /dev/dsk/c7d1s0
# ./mdb /blkptr1
> ::loadctf
> ::load /export/home/ferca/rawzfs.so
> 0::blkptr
DVA[0]: vdev_id 0 / 20a00
DVA[0]:       GANG: FALSE  GRID:  0000  ASIZE: 600
DVA[0]: :0:20a00:600:d
```

```
DVA[1]: vdev_id 0 / 10a0600
DVA[1]:       GANG: FALSE  GRID:  0000  ASIZE: 600
DVA[1]: :0:10a0600:600:d
LSIZE:  4000                    PSIZE: 600
ENDIAN: LITTLE                          TYPE:  DMU dnode
BIRTH:  39              LEVEL: 0     FILL:  5
CKFUNC: fletcher4                COMP:  lzjb
CKSUM:  6b32318e94:5c2daaba03bf:2c6ff4c8ee42f6:fcc053dae4ddcfe
> $q

# ./zdb -R wipefile:0:20a00:600:d,lzjb,4000 2>/dnode1
Found vdev: /dev/dsk/c7d1s0
# ./zdb -R wipefile:0:10a0600:600:d,lzjb,4000 2>/dnode2
Found vdev: /dev/dsk/c7d1s0
```

4. Eliminar las trazas que deja el archivo.

Esto se realiza accediendo directamente al disco mediante la dirección física (*offset*) obtenida en el paso anterior y sumándole la dirección física del disco donde comienza el espacio de almacenamiento asignable:

dnode1: 0x00020a00 + 0x00420400 = 0x00440E00
dnode2: 0x0010a060 + 0x00420400 = 0x014C0A00

Ahora procedemos a reemplazar los datos del *dnode* con ceros teniendo en cuenta que el *dnode* es una estructura de 512 bytes (ver Ilustración ##).

4.1.3.4.1.1 VERIFICAR Y ANALIZAR EL RESULTADO DE LAS PRUEBAS

Vamos a verificar el resultado de la prueba mediante Si verificamos el comando 4 (ver sección 4.1.2) y encontramos que no existe ninguna anomalía identificada por el sistema de archivos:

```
# zdb -v antizfs

... <-- output omitted
                        capacity   operations   bandwidth  ---- errors ----
description                used avail  read write  read write  read write cksum
wipefile                  76.0K 86.9M    13     0  840K     0     0     0     0
  /dev/dsk/c7d1s0          76.0K 86.9M    13     0  840K     0     0     0     0
```

4.1.3.4.1.2 CONCLUSIONES DE LOS RESULTADOS

Las pruebas para destruir la evidencia tuvieron resultados exitosos y sin ser identificados por las sumas de comprobación. Para esto se creó un archivo simple y se comenzó la búsqueda de la evidencia, donde los "*Ditto blocks*" entraron a jugar un papel importante mediante la políticas para la protección de los datos por defecto: sencillo para datos de usuario, doble para metadatos del sistema de archivos y triple para metadatos globales.

Luego, no había copias del contenido del archivo pero si existía una copia de los metadatos (*dnode*) por lo que se debió identificar y sobrescribir para destruirla. Sin embargo, no se tuvieron en cuenta casos donde el archivo de prueba estuviera implicado con una instantánea o con un clon donde encontraríamos más fuentes de evidencia.

Adicionalmente, si el archivo creado hubiera sido modificado, nos encontraríamos con copias del contenido del archivo debido al modelo COW (copia por escritura), lo que implicaría mas evidencia en el disco y mayor trabajo para el atacante.

Por otro lado, las sumas de comprobación no identificaron las anomalías de la sobrescritura por una sencilla razón; antes de realizar la sobrescritura se elimino el archivo desde el sistema operativo, por lo que las sumas de comprobación de este archivo ya no se tomaran en cuenta para la integridad del pool.

En resumen, en ZFS el atacante que desee eliminar un archivo sin dejar trazas o residuos, tendrá que revisar si existen más copias de los datos y de los metadatos que se pueden complicar con las instantáneas, los clones y las actualizaciones del archivo, beneficiando a los examinadores forenses.

5 RESULTADOS DE LA APLICACIÓN DEL MODELO

El modelo para implementar técnicas anti-forenses en ZFS, básicamente especificaba para cada uno de los métodos anti-forenses cómo y dónde se podían aplicar en ZFS, para poder especificar el tipo de pruebas que se iban a realizar. Cuando se empezó a aplicar el modelo, se esperaba encontrar muchos afinamientos y correcciones para que éste fuera coherente con las pruebas, sin embargo las pruebas se realizaron siguiendo los lineamientos del modelo con las pruebas establecidas allí y los resultados fueron los esperados de acuerdo a la investigación. Esto quiere decir que la profundidad y detalle de la investigación fué la correcta para obtener un modelo de aplicación afinado desde el comienzo.

Los inconvenientes más frecuentes de la aplicación del modelo, estaban relacionados con supuestos teóricos de la investigación, que no eran muy claros hasta que se comprobaban con el desarrollo de la pruebas. Por ejemplo, en el modelo se determinó que para la destrucción de la evidencia íbamos a borrar un puntero de bloque y en el momento de ejecutar la prueba se clarificó y se comprendió el verdadero objeto a borrar.

El modelo para ocultar la evidencia fue específico en las áreas vulnerables, pero no se tenía mayor detalle de la aplicación hasta que se realizaron las pruebas y se comprendieron conceptos como los FSB ("*File System Blocks*") que en la investigación previa no fue muy claro debido a la carencia de información.

Para eliminar las fuentes de evidencia y falsificar la evidencia, el modelo fue muy claro y se aplicó sin ninguna limitación al igual que su verificación y análisis.

Por otro lado, en el momento de realizar el modelo se planteaba que para cada prueba era necesario restaurar la imagen de la instalación del sistema inicial, pero en el desarrollo de las pruebas se comenzó a usar un *pool* para cada prueba con diferentes tamaños dependiendo de las necesidades de esta, lo que permitía realizar muchas pruebas con total seguridad ya que destruir un pool y crearlo nos llevaba un tiempo mínimo.

En términos generales, el modelo nos otorgo las instrucciones y metodología para la ejecución, verificación y análisis de las pruebas sin desviarse nunca del objetivo del trabajo, entregándonos unos resultados que se traducen en la correcta y total aplicación del modelo sin inconvenientes de mayor tipo.

6 RECOMENDACIONES PARA LOS INVESTIGADORES EN INFORMÁTICA FORENSE

6.1 IDENTIFICAR DONDE COMIENZA EL ESPACIO DE ALMACENAMIENTO ASIGNABLE

Para realizar cualquier análisis de un *pool* accediendo al disco directamente, la manera de ubicarse es identificando donde comienza el espacio de almacenamiento asignable, ya que el *offset* de datos y metadatos o de objetos en general es manejado por el sistema después de las etiquetas L1, L2 y el boot block.

La dirección física donde comienza el espacio de almacenamiento asignable, se puede encontrar como se realizo en la sección 4.1.3.2.1 o buscando con un editor hexadecimal la palabra clave “VERSION” desde el comienzo del disco hasta que encuentre la primera coincidencia (ver Ilustración 23).

6.2 ANALIZAR EL UBERBLOCK ACTIVO Y ENDIANESS

Antes de analizar los datos en un pool, el investigador se encontrará con el *uberblock* activo que le permitirá saber de antemano cuál fue el último acceso a los datos en un pool específico o si los datos en un pool son consistentes. Además, el investigador, encontrara muchas actualizaciones en diferentes instantes de tiempo en el arreglo de *uberblock* debido al modelo COW y además tres copias del arreglo en las etiquetas del dispositivo virtual, lo que le permitirá realizar líneas de tiempo sobre un caso específico.

Por otro lado, hay que tener cuidado con los *uberblocks* de menor número de transacción porque son vulnerables a alteraciones en las marcas de tiempo. Siempre debe haber una coherencia entre las marcas de tiempo y el número de transacción en el arreglo de *uberblock*. Un *ubeblock* con mayor número de transacción debe tener una marca de tiempo mayor a uno con número de transacción menor.

Si se desea verificar que el *uberblock* activo fue alterado en una imagen de un disco con ZFS, se debe tomar el número de transacción y marca de tiempo del *uberblock* activo desde el disco para luego verificar esta información al cargar la imagen en el sistema y si la marca de tiempo se actualizo y el número de transacción es el mismo (ver Ilustración 9), significa que hay anomalías de falsificación en la marca de tiempo del *uberblock* activo.

Para verificar el endianess o el orden en que se almacenan los bytes en la memoria, solo es necesario revisar el campo “ub_magic” del *uberblock*:

- Big Endian: 0x00bab10c
- Little Endian: 0x0cb1ba00

6.3 VERIFICAR LAS SUMAS DE COMPROBACIÓN

Es importante verificar las sumas de comprobación para identificar rápidamente anomalías en el sistema de archivos que puedan conducir a una falsificación, ocultamiento o destrucción de la evidencia. Para esto usamos los comandos 4 y 5 detallados en la sección 4.1.2. Si el sistema identifica errores en las sumas de comprobación, nos mostrara los objetos que poseen errores para el posterior análisis del investigador.

6.4 VERIFICAR SI EL ZIL SE ENCUENTRA DESHABILITADO

Para verificar si el ZIL está deshabilitado, podemos usar dos posibilidades. La primera es usar un script que nos resume el tamaño de los datos que son enviados al ZIL en un intervalo [52], la segunda posibilidad es usar el comando 3 detallado en la sección 4.1.2 para analizar los *datasets* que no posean información de sus respectivos encabezados ZIL y bloques (ver Ilustración 21).

6.5 REVISAR ANOMALÍAS EN LOS ESPACIOS RESERVADOS

ZFS no tiene ningún control sobre los espacios reservados como en el “*Boot Block*” y el “Blank Space” para identificar anomalías. Luego, es posible ocultar información en estos espacios por lo que el investigador podría sospechar y verificar si en estos espacios la información allí contenida es normal.

7 TRABAJOS FUTUROS

7.1 DESTRUCCIÓN DE LA EVIDENCIA TENIENDO EN CUENTA INSTANTÁNEAS Y CLONES

Las instantáneas y los clones implicarán nuevas copias de metadatos y datos en espacios asignados y no asignados lo que se traducirá en más fuentes de evidencia. El trabajo que se propone, es utilizar las mismas consideraciones y herramientas que se usaron para destruir la evidencia, identificar el archivo objetivo, las trazas de este y la sobrescritura de estas, pero ahora considerando pruebas que involucren el uso de instantáneas y clones.

7.2 VULNERAR SUMAS DE COMPROBACIÓN

En muchas de las pruebas realizadas, se evidenció la utilidad de las sumas de comprobación para verificar la integridad del pool, de tal manera que cualquier alteración en los espacios asignados para falsificar, ocultar o destruir la evidencia; era identificada por el sistema señalando el archivo con el problema. En el futuro se espera que los atacantes logren vulnerar las sumas de comprobación (fletcher2, fletcher4 o SHA256). Trabajo complicado cuando vemos que actualmente se están mejorado los métodos para lograr generar colisiones hash en MD4, MD5 y en SHA1.

7.3 FALSIFICAR ATRIBUTOS DE UN ARCHIVO

En este trabajo no se realizó esta prueba debido a que se identificó que el campo "dn_bonus" del *dnode* almacena los atributos del objeto como: marcas de tiempo, tamaño, propietario, privilegios de acceso del usuario; pero las sumas de comprobación identificarías los cambios en el *dnode*. Sin embargo se podrían encontrar algunas maneras de modificar estos atributos sin comprometer las sumas de comprobación como por ejemplo eliminando el archivo para que el sistema de archivos no verifique, pero si quede la evidencia falsa para el investigador.

7.4 HERRAMIENTAS Y METODOLOGÍAS FORENSES EN ZFS

Revisando las recomendaciones para los investigadores en informática forense, se puede pensar en herramientas y metodologías para realizar un análisis previo de un disco involucrado en un proceso

forense con sistema de archivos ZFS, revisando el estado del ZIL, búsquedas en espacios reservados y ocultos por el sistema de archivos, lista de los archivos anómalos por medio de las sumas de comprobación, construcción de líneas de tiempo, etc.

8 CONCLUSIONES DEL TRABAJO

La información existente para esta investigación con respecto a ZFS fue muy limitada, debido a que es un sistema de archivo relativamente nuevo y en desarrollo, lo cual implicó una gran dedicación y esfuerzo para el entendimiento de ciertos conceptos y estructuras de datos de éste sistema de archivos. De igual manera, para el estudio de las técnicas anti-forenses se requirió de gran creatividad para poder aplicar las pruebas sobre un nuevo sistema de archivos.

Para cada una de las pruebas a realizar, continuamente se hallaron complicaciones, pero con considerable paciencia y dedicación se logró resolver estos inconvenientes; como por ejemplo afinando el modelo de aplicación o buscando otro camino para ejecutar la prueba. La ejecución del modelo de aplicación, fué un proceso persistente, reiterativo y con mucha cautela para poder obtener resultados verificables y exitosos.

Mientras se realizaban las pruebas, comprobamos la utilidad de los *pools* ya que necesitábamos crear y destruir volúmenes muchas veces y rápidamente. Adicionalmente, evidenciamos que la aplicación de las técnicas anti-forense en ZFS es de mayor complejidad, debido a sus nuevas políticas de integridad, seguridad y diseño que a menudo perjudicaban a otros sistemas de archivos.

Con respecto a las pruebas, específicamente en la destrucción de la evidencia, encontramos que el sistema es vulnerable pero se requiere de un trabajo cauteloso para el atacante que desee eliminar y sobrescribir un archivo sin dejar trazas o residuos, por que tendrá que identificar si existen más copias de los datos debido al paradigma de copia por escritura (COW). El investigador por su lado, encontrará mas fuentes de evidencia recuperando numerosas copias de metadatos y bloques de datos. Lo que no sucedía en sistemas de archivos tradicionales (FAT, NTFS, EXT2/3, UFS, HFS+, etc.).

Para los archivos más pequeños que el tamaño del registro del FSB poseen un *slack space* mínimo (menor que el tamaño del sector: 512B) y los archivos más grandes que el tamaño del FSB en muchos de los casos, presentaran una cantidad significativa de *slack space* (menor de 128KB). Luego, ocultar información en el *slack space* es posible, sin embargo, ZFS detecta estas anomalías mediante las sumas de comprobación e identifica los archivos defectuosos.

Es decir, que esta técnica no será efectiva en ZFS hasta que los atacantes logren vulnerar las sumas de comprobación. Por otro lado, los investigadores ya no se beneficiaran encontrando viejos contenidos de archivos debido a que el *slack space* está ocupado de ceros.

Sin embargo, funcionalidades como la compresión transparente y posiblemente en un futuro el cifrado transparente, pueden ser utilizadas por los atacantes con el fin de ocultar información y dificultar el trabajo del investigador.

Por otro lado, cuando los atacantes decidan deshabilitar el ZIL (*"ZFS Intent Log"*) con la finalidad de evadir la generación de rastros en la máquina, el investigador podría determinar cuáles son los detalles propios del sistema de archivos en el momento de deshabilitar, como por ejemplo los *datasets* que no posean información de sus respectivos encabezados ZIL y bloques, pueden conducir a determinar si el ZIL está deshabilitado.

Con la finalidad de desviar la dirección de las pruebas, se falsificó un *uberblock* con un menor número de transacción que el *uberblock* activo. Esto debido a que el sistema realiza verificaciones únicamente en el *uberblock* activo con información codificada en los pares de Nombre-Valor (ver sección 2.1.2.3.3). Es decir que si es posible falsificar los *uberblocks* de menor número de transacción pero no el *uberblock* activo, a menos que no se inicie el sistema para que no se realice la verificación.

Adicionalmente, es posible irrumpir en el proceso forense desviando la dirección de las pruebas falsificando el *uberblocks* cuando el analista examine las marcas de tiempo para realizar líneas de tiempo coherentes. El *uberblock*, es el puntero maestro y la información contenida en éste será analizada antes que a los datos en el pool, lo que de antemano permitirá saber cuál fue el último acceso a los datos en un pool o si los datos en un pool son consistentes.

Los examinadores muchas veces se ven limitados con la información obtenida de archivos temporales, archivos corruptos, archivos log y registro de transacciones; por lo que los sistemas de archivos con tecnologías de backup son de gran ayuda a la hora de esclarecer sucesos ya que pueden realizar instantáneas en varios instantes de tiempo, lo que permite a los investigadores forenses comparar las instantáneas cronológicamente y analizar los cambios que se efectuaron en los datos, por ejemplo en el caso de una intrusión.

Recapitulando, encontramos que las sumas de comprobación y el paradigma de copia por escritura son estrategias de gran utilidad para mitigar las técnicas anti-forenses en ZFS, por lo que los investigadores deberán prestarle mucha atención; los atacantes por su parte no tendrán la misma efectividad aplicando las técnicas anti-forenses hasta que encuentren la manera de vulnerar las sumas de comprobación.

Luego, se preverá que la próxima generación de sistemas de archivos, le apostarán al almacenamiento, a la seguridad y a la administración de la información de maneras similares a ZFS. Por lo que comenzar a aprender de la inseguridad utilizando a la computación anti-forense como estrategia técnica, nos permitirá atacar el problema de raíz y avanzar en la generación de investigaciones, estrategias y procedimientos forenses más confiables.

REFERENCIAS

[1] John F Gantz et al. (2008, Marzo) The Diverse and Exploding Digital Universe: An Updated Forecast of Worldwide Information Growth Through 2011. [Online]. http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf [Ene 20, 2010]

[2] Nicole Lang Beebe, Sonia D. Stacy, and Dane Stuckey, "Digital forensic implications of ZFS," *Science Direct, journal homepage: www.elsevier.com/locate/dii n*, vol. 6, pp. S99-S107.

[3] Sun Microsystems, Inc. (2008, Sep) What is ZFS? [Online]. http://www.opensolaris.org/os/community/zfs/whatis/ [Oct 19, 2009]

[4] EMC Corporation. (2008) Crecimiento explosivo del Universo Digital. El nuevo fenómeno mundial: la "Sombra Digital". [Online]. http://argentina.emc.com/about/news/press/2008/20080311-01.htm [Sep 23, 2009]

[5] Daniel Torres, Jeimy Cano, and Sandra Rueda. (2006) Consideraciones técnicas y jurídicas para su manejo. [Online]. www.acis.org.co/index.php?id=856 [Sep 23, 2009]

[6] Jeimy Cano. (2007) Inseguridad Informática y Computación Anti-forense: Dos Conceptos Emergentes en Seguridad de la Información. [Online]. http://www.itgi.org/Template.cfm?Section=Home&CONTENTID=44702&TEMPLATE=/ContentManagement/ContentDisplay.cfm [Oct 23, 2009]

[7] ComputerForensics1. (2006) Importancia de Sistema de Archivo en Ordenador Forensics. [Online]. http://www.computerforensics1.com/spanish/Importancia-de-Sistema-de-Archivo-en-Ordenador-Forensics.html [Oct 24, 2009]

[8] Brian Carrier, *File System Forensic Analysis*, 1st ed.: Addison Wesley Professional, March 17, 2005.

[9] Scott Berinato. (2007, June 08) Data Protection: The Rise of Anti-Forensics. [Online]. http://www.csoonline.com/article/221208/The_Rise_of_Anti_Forensics [Oct 26, 2009]

[10] Ryan Harris, "Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem," *Science Direct, journal homepage: www.elsevier.com/locate/diin,* vol. 3S, pp. S44-S49, 2006.

[11] Rogers M. (2005) Anti-forensics. [Online]. http://www.cyberforensics.purdue.edu/ [Oct 28, 2009]

[12] Sun Microsystems, Inc. (2008) Solaris™ ZFS. Better, safer way to manage your data. [Online]. http://www.opensolaris.com/learn/features/solariszfs.pdf [Oct 30, 2009]

[13] Sun Microsystems, Inc, *Solaris ZFS Administration Guide*. Santa Clara, U.S.A, 2009.

[14] Pawel Jakub Dawidek. Porting the ZFS file system to the FreeBSD operating. [Online]. http://2007.asiabsdcon.org/papers/P16-paper.pdf [Oct 14, 2009]

[15] FUJITSU. (2006, Julio) fujitsu. [Online]. http://www.fujitsu.com/global/services/computing/server/unix/news/featurestory/PRMPWR_feature060714.html [Oct 30, 2009]

[16] Jeff Bonwick and Bill Moore. ZFS The Last Word In File Systems. [Online]. www.opensolaris.org/os/community/zfs [Oct 30, 2009]

[17] Jeff Bonwick. (2004, Sep) Jeff Bonwick's Blog: 128-bit storage: are you high? [Online]. http://blogs.sun.com/bonwick/entry/128_bit_storage_are_you [Nov 4, 2009]

[18] Community Group ZFS. (2008, mayo) Preguntas frecuentes de ZFS (FAQ). [Online]. http://hub.opensolaris.org/bin/view/Community+Group+zfs/faq-1 [Nov 4, 2009]

[19] Richard's Ranch. (2007, Mayo) ZFS, copies, and data protection. [Online]. http://blogs.sun.com/relling/entry/zfs_copies_and_data_protection [Dic 23, 2009]

[20] Sun Microsystems, Inc. (2006) ZFS On-Disk Specification. [Online]. http://hub.opensolaris.org/bin/download/Community+Group+zfs/docs/ondiskformat0822.pdf [Oct 29, 2009]

[21] blogs.sun.com. (2006, Mayo) Ditto Blocks - The Amazing Tape Repellent. [Online].

http://blogs.sun.com/bill/entry/ditto_blocks_the_amazing_tape [Dic 23, 2009]

[22] Jeimy Cano. (2006) Introducción a la informática forense. [Online]. http://www.acis.org.co/fileadmin/Revista_96/dos.pdf [May 16, 2010]

[23] E Casey, *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*.: Academic Press, February 2000.

[24] Óscar López, Haver Amaya, and Ricardo León. Informática forense : Generalidades, aspectos técnicos y herramientas. [Online]. http://gluc.unicauca.edu.co/wiki/images/1/1d/InfoForense.pdf [Oct 28, 2009]

[25] Andrew Li. Zettabyte File System Autopsy: Digital Crime Scene estigation for Zettabyte File System. [Online]. http://web.science.mq.edu.au/~rdale/teaching/itec810/2009H1/WorkshopPapers/Li_Andrew_FinalWorkshopPaper.pdf [Dic 18, 2009]

[26] Evtim Batchev. (2007, Nov) PROPOSAL: Open Solaris Forensics Tools Project. [Online]. http://www.opensolaris.org/jive/thread.jspa?threadID=45379 [Nov 11, 2009]

[27] Sun Microsystems, Inc. (2009) Project forensics: Forensic Tools. [Online]. http://hub.opensolaris.org/bin/view/Project+forensics/ [Dic 09, 2009]

[28] Max Bruning, "ZFS On-Disk Data Walk (Or: Where's My Data)," in *OpenSolaris Developer Conference*, Prague, 2008. [Online]. http://www.osdevcon.org/2008/files/osdevcon2008-max.pdf [Nov 11, 2009]

[29] Max Bruning. (2008, August) Max Bruning's weblog: Recovering removed file on zfs disk. [Online]. http://mbruning.blogspot.com/2008/08/recovering-removed-file-on-zfs-disk.html [Nov 11, 2009]

[30] Max Bruning, "ZFS On-Disk Data Walk (or: Where's my Data?)," 2008. [Online]. http://www.bruningsystems.com/osdevcon_draft3.pdf [Dic 9, 2009]

[31] Andrew Li and Josef Pieprzky. Digital Crime Scene Investigation for Zettabyte File System.

[Online]. http://web.science.mq.edu.au/~rdale/teaching/itec810/2009H1/WorkshopSlides/Session2RoomW6B338/05_Li_Andrew_WorkshopSlides.pdf [Sep 23, 2009]

[32] Jeimy Cano. (2007, Sept) Estrategias anti-forenses en informática: Repensando la computación forense. [Online]. http://www.alfa-redi.org/rdi-articulo.shtml?x=9608 [Ene 22, 2010]

[33] Sonny Discini. (2007, June) Antiforensics: When Tools Enable the Masses. [Online]. http://www.esecurityplanet.com/features/article.php/3685836/Antiforensics-When-Tools-Enable-the-Masses.htm [Ene 20, 2010]

[34] BJ Bellamy. (2007, Mayo) Anti-Forensics and Reasons for Optimism. [Online]. http://www.nasact.org/conferences_training/nsaa/conferences/ITWorkshopConferences/2007ITWorkshopConference/PresentationsHandouts/bellamy.ppt [Ene 20, 2010]

[35] Tom Van de Wiele. (2006, November) Uniskill – ICT Anti-Forensics. [Online]. http://www.bcie.be/Documents/BCIE_Training03_ICT_Anti-Forensics_291106_TVdW.pdf [Ene 22, 2010]

[36] Jeimy Cano. Borrando archivos. Conceptos básicos sobre la dinámica del funcionamiento de los sistemas de archivo. [Online]. http://www.virusprot.com/filesystems05.pdf [Ene 22, 2010]

[37] grugq. (2004) The art of defiling: Defeating forensic analysis on Unix file systems. [Online]. http://www.packetstormsecurity.org/hitb04/hitb04-grugq.pdf [Ene 26, 2010]

[38] Charles Williford. (2007, November) Computer Anti-Forensics. [Online]. http://www.fis.ncsu.edu/csd2007/Files/antiforensic.pdf [Ene 20, 2010]

[39] Mark Russinovich. (2006, Noviembre) SDelete v1.51. [Online]. http://technet.microsoft.com/es-es/sysinternals/bb897443.aspx [Dic 23, 2009]

[40] Vincent Liu and Francis Brown. (2006, April) Bleeding-Edge Anti-Forensics. [Online]. http://www.metasploit.com/data/antiforensics/InfoSecWorld%202006-K2-Bleeding_Edge_AntiForensics.ppt [Ene 29, 2010]

[41] Vincent Liu and Patrick Stach. (2006, May) Defeating Forensic Analysis CEIC. [Online]. http://www.metasploit.com/data/antiforensics/CEIC2006-Defeating_Forensic_Analysis.pdf [Ene 28, 2010]

[42] Paul Henry. (2007, Oct) Anti - Forensics. [Online]. http://www.thetrainingco.com/pdf/Tuesday/Tuesday%20Keynote%20-%20Anti-Forensics%20-%20Henry.pdf [Feb 2, 2010]

[43] grugq. FIST! FIST! FIST! Its all in the wrist: Remote Exec. [Online]. http://www.phrack.com/issues.html?issue=62&id=8&mode=txt [Feb 4, 2010]

[44] Maximiliano Caceres. (2002) Syscall Proxying - Simulating remote execution. [Online]. download.coresecurity.com/corporate/attachments/SyscallProxying.pdf [Feb 5, 2010]

[45] Max. (2009, February) Leave No Artifacts Behind – Linux Live CDs. [Online]. http://www.anti-forensics.com/leave-no-artifacts-behind-linux-live-cds [Feb 7, 2010]

[46] University of Edinburgh. (2008, October) touch - change file timestamps. [Online]. http://unixhelp.ed.ac.uk/CGI/man-cgi?touch [Feb 5, 2010]

[47] Xiaoyun Wang. (2007, August) Collisions for Hash Functions. [Online]. http://eprint.iacr.org/2004/199.pdf

[48] Andrew Li. Zettabyte File System Autopsy: Digital Crime Scene Investigation for Zettabyte File System. [Online]. http://web.science.mq.edu.au/~rdale/teaching/itec810/2009H1/WorkshopPapers/Li_Andrew_FinalWorkshopPaper.pdf [Dic 18, 2009]

[49] Carlos Enrique Nieto, *Consideraciones anti forenses en sistemas de archivos HFS y HFS+*. Bogotá, Colombia: Pontificia Universidad Javeriana, 2007.

[50] hswong3i. (2009) Clone VirtualBox disk image on MS Windows. [Online]. http://edin.no-ip.com/content/clone-virtualbox-disk-image-ms-windows-mini-howto [Jun 17, 2010]

[51] Anónimo. ZFS Evil Tuning Guide. [Online].

http://www.solarisinternals.com/wiki/index.php/ZFS_Evil_Tuning_Guide#Disabling_the_ZIL_.28Don.27t.29 [Ene 20, 2010]

[52] Richard Elling. How do you know if a separate ZIL log device will help your ZFS performance? [Online]. http://www.richardelling.com/Home/scripts-and-programs-1/zilstat [Ago 26, 2010]

[53] Red Hat Enterprise Linux. Manual de administración del sistema, Qué es LVM? [Online]. http://web.mit.edu/rhel-doc/3/rhel-sag-es-3/ch-lvm-intro.html [Oct 14, 2009]

[54] SM Data. (2009) ¿Qué es RAID? [Online]. http://www.smdata.com/queesraid.htm [Oct 14, 2009]

[55] Sun Microsystems, Inc. (2003) Class NVList. [Online]. http://java.sun.com/j2se/1.4.2/docs/api/org/omg/CORBA/NVList.html [Oct 14, 2009]

[56] Informática IES Virgen de la Paloma. Disco duro (visión software). [Online]. http://www.palomatica.info/juckar/sistemas/hardware/perifericos/disco_duro_sfw.html [Feb 9, 2010]

[57] Jeimy Cano. (2007, Sept) Estrategias anti-forenses en informática: Repensando la computación forense. [Online]. http://www.alfa-redi.org/rdi-articulo.shtml?x=9608 [Ene 22, 2010]

[58] Red Hat Linux. Chapter 9. Network File System (NFS). [Online]. http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/ch-nfs.html [Ago 27, 2010]

[59] Dean Grimm. WinMerge. [Online]. http://winmerge.org/ [Ago 27, 2010]