Universidad de las Ciencias Informáticas
Facultad 2

"Tele Identificador Personal. Configuración de los servidores BIND de DNS y almacenamiento de la información de los subscriptores"

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas.

Autor:
Reyner Herrpinark Lugo

Tutor:
Ing. Denys Buedo Hidalgo

Ciudad de la Habana, 2009
"Año del 50 Aniversario del Triunfo de la Revolución"

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 2 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Reyner Herrpinark Lugo

_____

Ing. Denys Buedo Hidalgo

_____

Dedicatoria

Dedico este trabajo por entero a mi madre, por ser el eje de mi formación como hombre y para materializar uno de sus grandes sueños: Verme encaminado profesionalmente.

A mis abuelos Adelfa y Elier por sembrar en nuestra familia el valor de la unidad y el amor entre sus miembros.

A mi hermano, con la esperanza de motivarle a estudiar y a nunca dejar de superarse.

A esa grandiosa familia la cual me tomé la libertad de escoger: mis amigos.

Agradecimientos

Quisiera agradecer primeramente a mis padres, por siempre confiar y respetar sin cuestionar mis decisiones en cuanto a mi vocación. También a mi tía Niurka, por su constante apoyo a pesar de la distancia. Gracias a la UCI por brindarme los medios para superarme; gracias a la Revolución por permitirme estudiar en ella estos cinco años.

Y un muy especial agradecimiento a todas esas personas que me han ayudado en la confección de este documento, de modo particular, gracias Lisy por estar cuando más te he necesitado; a mis dos grandes compañeros Christian y René por soportarme y ayudarme tanto en lo profesional como en lo personal; gracias a los Prokers por enseñarme tanto; a Erick por resultar ser un gran compañero de trabajo; a Álvaro por sus imprescindibles aportes; gracias a mi tutor por la confianza que ha delegado en mí.

Gracias a usted, por leer este trabajo…….

Resumen

La Empresa de Telecomunicaciones de Cuba o ETECSA por sus siglas, con la visión de ampliar sus servicios hacia la población, pretende implementar el protocolo ENUM en su intranet para explotarlo y en dependencia de los resultados de su impacto, hacerlo llegar a los usuarios no gubernamentales en un futuro. Para este fin el proyecto Tele Identificador Personal (TIP por sus siglas) se encuentra desarrollando un conjunto de aplicaciones que hacen uso en todo momento de los servidores de DNS para su correcto funcionamiento, debido a que el protocolo en cuestión depende constantemente de la interacción con el sistema de nombres de dominio. La presente investigación va enfocada fundamentalmente hacia los servidores del DNS que atienden al sistema creado por el proyecto TIP, presentando una propuesta para el almacenamiento de la información de los subscriptores. Se exponen además los resultados alcanzados durante las pruebas realizadas a dicha propuesta; así como algunas cuestiones de seguridad a tomar para garantizar un correcto despliegue de los servidores del DNS.

Palabras claves

DNS, DLZ, NAPTR, ENUM, TIP, BIND, CRON, CRONTAB, PostgreSQL, Linux, ETECSA, UCI.

Índice

Índice de Figuras

Índice de Tablas

Introducción

El desarrollo actual de las telecomunicaciones ha alcanzado niveles muy elevados; el avance y surgimiento de nuevas tecnologías unido a las necesidades de los clientes ha propiciado el incremento y la diversidad de los servicios que son suministrados por las operadoras de telecomunicaciones en todo el planeta. Este desarrollo ha sido potenciado por la evolución de las redes y de internet, las cuales constituyen un canal cómodo y escalable que propicia la implantación de nuevas tecnologías.

Hoy en día una persona puede tener un teléfono donde se le pueda llamar, una dirección de correo a donde se le pueda enviar un e-mail, o un número de celular por el cual se le pueda contactar o mandar un mensaje, de igual modo podría tener una página web de referencia, en fin, puede brindar para ser contactada cuantos servicios posea y desee hacer público.

El uso de las tarjetas de presentación es una práctica relevante en pos de ayudar a organizar todos los servicios de una persona. Con ellas, se mantienen en un solo medio la información que la misma considere deba ser pública para contactarle, y es esta misma la encargada de gestionar su distribución.

Pero, ¿cómo queda este recurso ante un mundo que progresa y cada vez se enfoca más hacia la digitalización? ¿Un intento por sacar a las tarjetas de presentación del plano de lo obsoleto podría ser simplemente digitalizándolas?

La digitalización provee mucho más que el simple paso del medio convencional al medio computacional. Un mejor enfoque sería tener como identificador no una tarjeta digitalizada que contenga una serie de servicios para contactar, sino una llave que provea todos estos servicios como un todo. Esta llave podría ser un número o una cadena mixta con un formato predefinido; pero ha de cumplir la condición de que tiene que ser única en el universo en que se muevan todas las de su clase ya que ella identifica a una y solo a una persona. Existe actualmente una solución interesante que permite implementar este enfoque, la cual se denomina ENUM.

El mapeo de números telefónicos o ENUM (del inglés: Telephone Number Mapping), es un protocolo resultante de la labor del Grupo de Trabajo de Mapeo de Números Telefónicos perteneciente a las Fuerzas de Ingeniería para el Desarrollo de la Internet o IETF (del inglés: Internet Engineering Task Force) y tiene como filosofía correlacionar los números telefónicos de la red pública que cumplen con

la recomendación e.164 de la ITU-T (del inglés: International Telecommunication Union - Telecommunication) con los identificadores de recursos uniformes o URI (del inglés: Universal Resources Identifier) almacenados en los servidores del Sistema de Nombres de Dominio o DNS (del inglés: Domain Name System).

El empleo del DNS como base de datos se debe a que es una tecnología muy madura probada además con resultados confiables. Por otra parte existe globalmente toda una infraestructura de DNS creada, por lo que el desarrollo del ENUM no acarreará gastos en este sentido.

El DNS, brinda un servicio de búsqueda distribuido y se rige por una estructura jerárquica en modo de árbol. Su funcionamiento está soportado por aplicaciones servidoras desplegadas en algunos de sus nodos, que contienen y manipulan la información rectora de la zona que controlan en registros alojados en ficheros (ficheros de zona).

Todos los contactos de una persona en forma de URI son almacenados en ficheros de texto dentro del DNS y son accedidos mediante el identificador personal de cada una. La cantidad media de servicios que posee un cliente en el mundo es de 5, por lo que en un país donde la población sea relativamente elevada la cantidad de registros almacenados en el servidor puede ser muy grande, esto afectará significativamente el rendimiento del mismo y el tiempo de respuesta a las peticiones realizadas.

La Empresa de Telecomunicaciones de Cuba (ETECSA) en su afán de brindar servicios de calidad a la población y contribuir a la informatización de la misma, después de un estudio pertinente decidió la implementación de este novedoso servicio y en colaboración con la Universidad de las Ciencias Informáticas creó un grupo de desarrollo con tal propósito.

El desarrollo del ENUM de usuario en Cuba se adapta a las características propias del país y se rige por los estándares y recomendaciones dictadas por las organizaciones internacionales responsables de regular este sector a nivel mundial, es de vital importancia la distribución de la información en los DNS y la disponibilidad y velocidad de los mismos.

¿Será posible mejorar el rendimiento del servidor de DNS mediante el empleo de otras técnicas alternativas al uso de los ficheros de zona?

Cualquier técnica alternativa escogida para el despliegue de los servidores de DNS ha de estar enfocada a mantener correlacionados de alguna forma los datos pertenecientes al DNS y los datos propios del sistema TIP.

Las bases de datos constituyen un recurso inseparable al desarrollo de aplicaciones informáticas y en la actualidad su uso es casi obligatorio para el éxito de los productos desarrollados. Por estos motivos, existe en el sistema TIP una base de datos que funge como directorio y almacena además información propia de la administración del sistema en general.

Es de vital importancia entonces la existencia de una sincronización preferiblemente automatizada entre el servidor DNS y este servidor que funge como directorio en pos de lograr la correspondencia de los datos entre sí.

La presente investigación tiene como objetivo resolver el siguiente **problema**:

¿Cómo configurar el servidor de DNS para lograr un mejor rendimiento del mismo y una mayor disponibilidad de la información que él maneja?

Para ello se tiene enmarcado como **objeto de estudio** el protocolo ENUM, el funcionamiento y configuración de los DNS en Internet y el paradigma Cliente – Servidor ; siendo su **campo de acción** la configuración de los servidores de DNS para agilizar el desarrollo y administración del sistema TIP.

Con el fin de alcanzar los resultados esperados se plantea los siguientes objetivos:

Objetivo General:

- Configurar el servidor de DNS para lograr un mejor rendimiento del mismo y una mayor disponibilidad de la información para el sistema TIP.

Objetivos específicos:

- Diseño de la arquitectura de DNS a emplear en Cuba que permita la implantación del sistema TIP.
- Diseño de la base de datos de contactos que sirva de fuente de información al servidor de DNS.

- Integración del servidor de DNS con la base de datos mediante un driver o manejador.

Con el fin de alcanzar los objetivos trazados se desarrollarán las siguientes **tareas de la Investigación**:

- Buscar y analizar información relacionada con el protocolo ENUM.
- Analizar el estado del arte en el campo de la configuración de servidores DNS para valorar el desarrollo que han alcanzado y enfocar esta investigación en las tecnologías más idóneas.
- Analizar las principales tendencias en la implementación del protocolo ENUM en otros países, para identificar buenas prácticas y deficiencias.
- Elaborar un entorno para ejecutar las pruebas y documentar los resultados.

Capítulo 1: El ENUM y aplicaciones de soporte

1.1 Introducción

En este capítulo se hará énfasis fundamentalmente sobre las recomendaciones que da la IETF para llevar a cabo la implementación del protocolo ENUM a nivel mundial. Para tal fin se exponen algunos conceptos que en su conjunto definen a este protocolo. Se presenta además un análisis de las diversas herramientas que podrían brindar soporte al DNS orientado al ENUM, conjuntamente con una serie de posibles enfoques para el almacenamiento y administración de la información del DNS en las mismas.

1.2 ENUM según las recomendaciones de la IETF (RFC 3761)

Si se lee la RFC 3761 de la IETF, la cual define al protocolo ENUM, se puede decir que el mismo está regido fundamentalmente por cuatro conceptos:

- Números E.164
- Identificador de Recursos Uniforme (URI)
- Sistema de Nombres de Dominio (DNS)
- Dominio “e164.arpa”.

1.2.1 Números E.164

El Plan Internacional de Numeración para las Telecomunicaciones lo establece la Recomendación E.164 de la ITU-T.

Un número telefónico que aplique este estándar está compuesto de un número variable de cifras dispuestas en bloques de código específicos. Los bloques de código del mismo son el indicativo de país o CC (del inglés: Country Code) y el número nacional (pudiendo reflejar un campo significativo) N(S)N (del inglés: National Significant Number).

Las normas establecen que el indicativo de país debe de tener de 1 a 3 dígitos y ha de ser la cabecera del mismo, además de ser único para cada país. La longitud total del número no ha de exceder los 15 dígitos, por tanto, la suma de los dígitos de la cabecera (el indicativo de país) más el número nacional (resto del número) ha de ser menor o igual que 15.

A continuación se muestra una figura que desglosa un número perteneciente a una oficina de ETECSA en Las Tunas, Cuba (+53 31 346988).

Fig.1. Desglose en campos de un número con formato E.164.

El indicativo de país para Cuba es el 53, siendo el resto de este número el número nacional.

Cada país, de forma interna, puede definir un código para cada una de sus zonas, en el ejemplo se ha separado el código que pertenece a la provincia de Las Tunas (31) del número del abonado (346988), pero en su conjunto se está refiriendo al número nacional (N(S)N). Ha de notarse como nunca se ha excedido de la longitud de 15 dígitos en el ejemplo expuesto.

1.2.2 Identificadores de Recursos Uniformes (URI)

Los Identificadores de Recursos Uniformes pueden ser vistos como un puntero único a las direcciones de Internet. Las URI son cadenas de caracteres que identifican recursos de diferentes naturalezas como documentos, imágenes, archivos, bases de datos, direcciones de correo electrónico u otros recursos o servicios en un formato estructurado común. Los tipos de URI más conocidos son los

Localizadores de Recursos Uniforme o URL (del inglés: Uniform Resource Locator) que se utilizan para localizar recursos de la WWW (del inglés: World Wide Web).

1.2.3 Sistemas de Nombre de Dominio (DNS)

Internet es basado fundamentalmente en direcciones IP. Estas direcciones identifican dentro de la gran red de redes a la interfaz de conexión de un dispositivo (generalmente de una computadora) de forma unívoca, y están compuestas por números que varían su longitud en dependencia de la versión que puede ser 4 ó 6. A los usuarios les resulta engorroso tener que manipular cada uno de estos números, además no se hace intuitivo el trabajo con ellos.

La creación del Sistema de Nombres de Dominio da solución a este problema, permitiendo una traducción bidireccional entre estos números a cadenas nemotécnicas (nombres de dominio) fáciles de memorizar y manipular. Por otra parte el sistema brinda una estructura jerárquica a modo de árbol que permite la organización en distintos niveles de estas direcciones IP, optimizando la administración de las mismas.

Desde el punto de vista práctico, cuando se escribe en el navegador la URL www.etecsa.cu el Sistema de Nombres de Dominio la traduce en su respectiva dirección IP y el usuario permanece ajeno a este proceso, sin tener que saber que realmente está empleando una dirección IP perteneciente a un recurso denominado **www** dentro del dominio **etecsa.cu**. Incluso la dirección IP de este recurso podría cambiar, que si se actualiza el DNS se podrá seguir accediendo al recurso por www.etecsa.cu y el usuario sigue abstraído de los cambios efectuados.

Como se ilustra en la figura 2, en la cima del Sistema de Nombres de Dominio se encuentra el nodo raíz (.), seguido hacia abajo por un grupo de nombres de dominio denominados TLD's (del inglés: Top Level Domains) los cuales pueden ser genéricos (gTLD's : .com, .edu, .net, .gob, .org) o relacionados con códigos de países (ccTLD's: .cu, .es, .mx, .fr). A un nivel más profundo le siguen entonces los SLD's (del inglés: Second Level Domains) y en lo sucesivo cada vez más en profundidad cualquier número de subdominios.

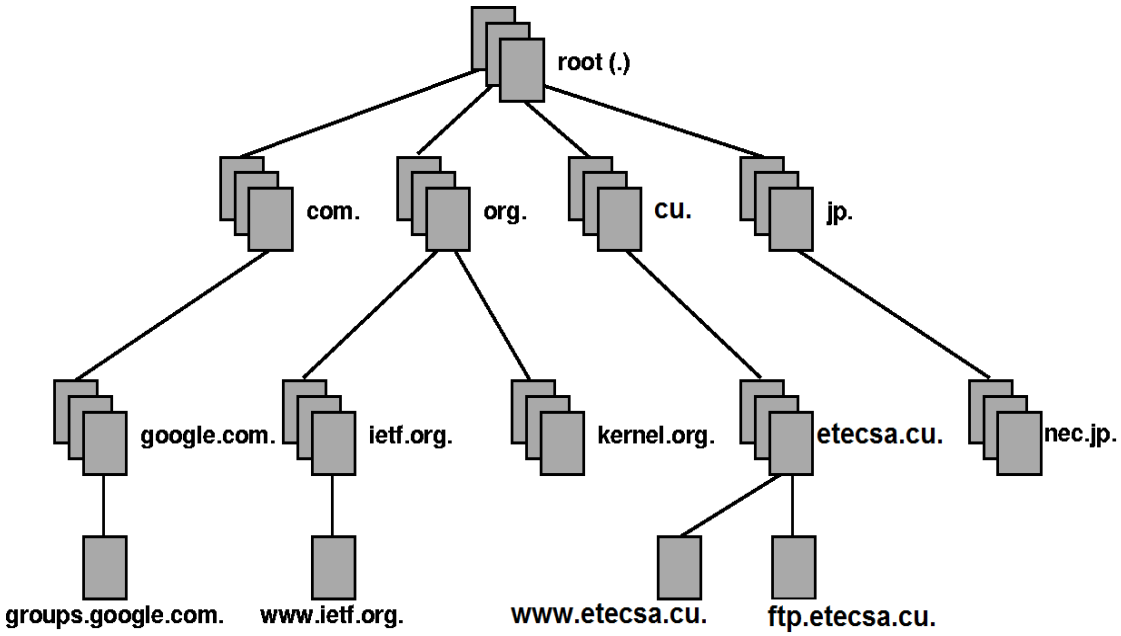

Fig.2. .Fragmento del árbol de dominio del DNS.

Por ejemplo, en el caso **.etecsa.cu** se hace referencia al SLD **.etecsa** que está dentro del TLD **.cu**. Nótese primeramente que se utiliza en todo momento el carácter "." como separador dentro de la URL para delimitar el nombre de un dominio con sus subdominios inferiores. Y por segundo se debe prestar atención sobre cómo los nombres de dominio de mayor nivel quedan siempre más a la derecha.

1.2.4 Dominio e164.arpa

Para poder correlacionar los números que cumplen con el estándar **E.164** con los Identificadores de Recursos Uniformes y almacenarlos dentro del Sistema de Nombres de Dominio se ha propuesto el empleo del subdominio **.e164.arpa**.

Aunque no existe una aprobación sobre esta propuesta, la ITU-T ha adoptado de forma provisional el dominio de nivel superior "arpa" y designado a la RIPE NCC (del francés: Réseaux IP Européens Network Coordination Center) como registro internacional para el dominio de segundo nivel ".e164.arpa". Los códigos nacionales individuales se colocan bajo el subdominio "e164.arpa". De esta forma por ejemplo los números del plan nacional de numeración de Cuba se harían corresponder a "x.x...3.5.e164.arpa".

El TLD “arpa” (del inglés: Address and Routing Parameters Area) se usa exclusivamente para propósitos de infraestructura de Internet, no es visible para usuarios comunes de Internet y tiene un número muy limitado de dominios de segundo nivel.

1.3 ¿Cómo es soportado ENUM por el DNS?

Conformar entonces un dominio ENUM para insertarlo dentro del DNS se regirá según la RFC 3761 por los siguientes pasos:

- Se toma el número que se desea almacenar y lo se lleva a formato internacional con las especificaciones E.164. Ejemplo: **31346988** de Cuba sería **+5331346988**.

- Se substraen todos los caracteres no numéricos y se invierte el orden de los dígitos, luego se separa por puntos para concatenarle finalmente el subdominio **e164.arpa**. De esta forma el número antes expuesto se transforma como sigue **5331346988 → 8.8.9.6.4.3.1.3.3.5.e164.arpa**.

El motivo de esta inversión se debe a poder mantener así la estructura de mayor jerarquía hacia la derecha dentro de la entrada en el DNS, correspondiendo al código del país, una posición jerárquicamente superior al resto del número telefónico.

En la Figura 3 se muestra la posición que ocuparía cada porción de la numeración dentro del dominio ENUM formado para el número +53 31 34-6988.

Fig.3.Desgloce en campos de un nombre de dominio ENUM.

En la práctica, ENUM se hace visible al mundo haciendo uso del DNS cuando se crea una zona dentro del espacio de nombres, que almacene los distintos contactos referenciados por un nombre de dominio ENUM.

El protocolo ENUM utiliza según define la RFC 3403 los registros de recursos o RR (del inglés: Resource Record) del DNS llamados Punteros de Entidad de Denominación o NAPTR (del inglés: Naming Authority Pointer) que es un nuevo tipo de registro que admite expresiones regulares o forma normalizada de escribir las cadenas usadas especialmente para buscar patrones de textos determinados. De este modo se puede determinar los métodos o servicios disponibles para contactar un nodo específico identificado mediante un número de la Recomendación E.164.

1.3.1 Estructura de los registros NAPTR aplicados en el ENUM

Los registros de tipo NAPTR empleados por el ENUM están compuestos por la siguiente serie de campos:

NAME [TTL] CLASS RR ORDER PREF FLAG SVC REGEXP REPLACE

Campo NAME:

En este campo se almacenaría el número (nombre de dominio) ENUM (**8.8.9.6.4.3.1.3.3.5.e164.arpa**), provocando entonces una entrada dentro del DNS para dicho número. Generalmente las entradas en el campo **NAME** para todos los registros de recursos son de tipo cadena de caracteres y solo es omitida en pos de abreviar múltiples entradas similares, si con anterioridad el nombre del dominio ENUM es referenciado con una directiva **$ORIGIN** que le contenga.

Campo TTL:

Este campo define el Tiempo de Vida del Registro o TTL (del inglés: Time To Live). En si se refiere a la cantidad de tiempo expresado en segundos que un registro de recurso ha de ser almacenado por otros servidores del DNS dentro de su memoria caché. Es un campo que generalmente se suele llenar con números enteros. Las últimas versiones de BIND9 permiten el empleo de notaciones nemotécnicas para aumentar la comprensión como el uso de "**h, d, w**" al final del valor entero para expresar horas, días y semanas de almacenamiento respectivamente. Ejemplo **14d** significa que el registro en cuestión ha de ser retenido (almacenado) dentro de la memoria caché de los servidores del DNS por un período de 14 días. Este campo para los registros de tipo NAPTR se torna opcional, ya que existe una directiva que impone el comportamiento del TTL para todos los registros de recursos de forma global en un fichero de zona.

Campo CLASS:

Este campo para casi todos los registros, incluyéndose los de tipo NAPTR, siempre va a tomar un valor constante que es "**IN**", e indica que el registro es de clase Internet.

Campo RR:

Este campo hace referencia al tipo de registro de recurso, y para el caso de los NAPTR tomará siempre un valor constante el cual será "**NAPTR**".

Campo ORDER:

En este campo se define el orden en que los registros de recursos de tipo NAPTR han de ser procesados. Admiten un rango de valores establecidos desde 0 hasta 65565 donde los valores más bajos presentan la mayor prioridad. Si dos registros de recursos de tipo NAPTR poseen el mismo valor en este campo entonces el campo **PREF** es usado para establecer el orden de procesamiento.

Campo PREF:

Este campo define la preferencia para el caso de entradas con igual valor en el campo **ORDER** a la hora de procesar los registros de recursos de tipo NAPTR. Admite un rango de valores que se extiende desde 0 hasta 65535 donde los valores más pequeños son los de mayor preferencia. Realmente rige la preferencia del usuario para ordenar el modo en que se mostrarán los registros hacia el cliente que hace las consultas independientemente de la preferencia que establece el administrador del servidor almacenada en el campo **ORDER**.

Campo FLAG:

En este campo para los registros de recursos de tipo NAPTR utilizados en el ENUM siempre se pone el valor "u", que indica que el registro define una condición terminal donde el resultado será una URI. Siempre este campo ha de ir entrecomillado.

Campo SVC:

Este campo tiene el formato **Rs+Protocol[:Protocol]**. **Rs** es el servicio de resolución que indica las reglas de transformación que en el caso del ENUM tomaría casi siempre el valor de E2U, acrónimo de ENUM-to-URI. El símbolo de **+** (signo de suma) es tomado como un separador y siempre ha de estar presente. **Protocol** define un protocolo conocido por las aplicaciones clientes como por ejemplo SIP,

H323 o mailto por citar algunos, y ha de tener una longitud no mayor de 32 caracteres alfanuméricos destacando que siempre ha de comenzar con uno alfabético. En su totalidad este campo ha de estar encerrado entre comillas como en el caso del campo **FLAG**.

Campo REGEXP:

Este campo va entrecomillado siempre y encierra una expresión regular extendida de tipo POSIX utilizada posteriormente en el proceso de transformación.

Campo REPLACE:

Este campo dentro del universo de los registros de recursos de tipo NAPTR generales es opcional. Para el caso del ENUM su participación es nula y su ausencia se representa mediante un punto (.).

1.4 Aplicaciones servidoras

Actualmente existen en el mundo una gran cantidad de aplicaciones servidoras que permiten dar soporte al Sistema de Nombres de Dominios. Se ha realizado un estudio desglosando las principales características de las más conocidas, como por ejemplo su portabilidad entre sistemas operativos, soporte para los registros NAPTR, gama de opciones para el almacenamiento de la información, licencias y términos de uso. A continuación se le expone a los lectores lo más relevante de esta investigación.

1.4.1 DJBDNS

Es una colección de herramientas creadas por Daniel J. Bernstein que brinda a la comunidad un paquete para el trabajo con el DNS, libre de hoyos de seguridad (al menos hasta el momento), como desbordamiento de memoria y envenenamiento de la caché. Se considera muy segura, al punto de que hoy en día existe una recompensa no reclamada de $10000 dólares a quien le encuentre alguna vulnerabilidad. Se caracteriza además por un diseño modular enfocado a cuidar la seguridad y la velocidad de procesamiento mediante el uso de bases de datos constantes. De manera nativa no posee soporte para los registros NAPTR, por lo que necesita ser recompilado con algunos ajustes si se desea utilizar este tipo de registro. Actualmente su código no está siendo activamente mantenido y fue liberado en el 2007 a la licencia de GPL. Se puede instalar sobre plataforma BSD, Solaris, Linux y Mac OS X.

1.4.2 MaraDNS

Es un servidor que se caracteriza por su sencillez de administración y un historial respetable en cuanto a seguridad. Tiene soporte para registros de tipo NAPTR y está distribuido bajo licencia BSD. Se puede instalar sobre plataforma BSD, Solaris, Linux y Mac OS X, y actualmente se está trabajando para que se pueda ejecutar igual sobre plataformas Windows. Este servidor lamentablemente posee algunas limitaciones en cuanto a soporte para comportarse como DNS esclavo, pues a pesar de tener una herramienta que le permite recibir ficheros de zona, el proceso ha de ser programado para que se ejecute desde otra aplicación propia del sistema operativo. Otra limitante que posee es que ha resultado ser vulnerable a ataques de denegación de servicios mediante errores por desbordamiento de memoria, por lo que tiene que ser recompilado con algunos ajustes para mitigar los riesgos.

1.4.3 PowerDNS

Es una versátil aplicación escrita en C++ y distribuida bajo licencia GPL V2. Posee una gran variedad de características para el almacenamiento de datos, desde la forma clásica de ficheros de zona hasta soporte para Sistemas Gestores de Base de Datos como PostgreSQL, MySQL, Oracle y SQLite, entre otros. Brinda también opciones para configurar algoritmos de balance de carga y respuestas ante caídas de sistema. Las funcionalidades para trabajar como autoritativo y como servidor de DNS recursivo son puestas en aplicaciones separadas. Puede ser ejecutado en plataforma BSD, Solaris, Linux, Windows. En Mac OS X está lanzado pero aún no es estable. Posee soporte para registros NAPTR de manera nativa. Como pequeña desventaja se puede señalar que no trae soporte para el protocolo TSIG (del inglés: Transaction Signature), muy empleado principalmente para proveer un método de autenticación a la hora de actualizar el contenido de una base de datos de DNS dinámica.

1.4.4 MyDNS

Este servidor fue construido con el propósito principal de poder soportar el almacenamiento de sus zonas en Sistema Gestores de Base de Datos como PostgreSQL y MySQL. No posee la funcionalidad de Servidor de Nombres Recursivo. Soporta actualmente manipulación para los registros de tipo NAPTR. Puede ser compilado para plataformas FreeBSD y Linux, y está liberado bajo la licencia GPL.

1.4.5 NSD

Este servidor es para dar soporte de modo autoritativo nada más. Es distribuido por NLNet Labs bajo licencia BSD. El propósito de su desarrollo se basa fundamentalmente en la adición de variantes de los códigos de los servidores de nombres de dominios de más alto nivel en pos de crear uno mucho más

flexible contra las fallas y ataques de exploits. Utiliza el estilo de almacenamiento de zonas de BIND9 con el empleo de ficheros de zona. Actualmente se puede instalar en BSD, Solaris, Linux y Mac OS X.

1.4.6 Simple DNS Plus

Es un servidor exclusivamente para plataformas Windows. Su característica fundamental es que provee de una administración sencilla dado que todas sus opciones de configuración están disponibles con el uso de ventanas de esta plataforma. Presenta diversos ayudantes para ejecutar tareas consideradas comunes en el trabajo administrativo de los servidores como puesta en marcha de nuevas zonas, importar zonas, hacer copias de resguardo, entre otras. Como desventaja se debe señalar que no posee soporte para las extensiones de seguridad DNSSEC y que solo está soportado en plataformas Windows. Se distribuye bajo licencia propietaria.

1.4.7 BIND9

Acrónimo de Berkeley Internet Name Domain. Es uno de los servidores del Sistema de Nombres de Dominio más popular que existe dada su alta aceptación y uso. Su código fue reescrito desde 0 basándose en sus anteriores versiones (BIND4 y BIND8) para proveer una mejor arquitectura que eliminara errores de seguridad y dar soporte a las Extensiones de Seguridad DNS o DNSSEC (del inglés: DNS Security Extensions). Posee soporte para registros NAPTR de manera nativa y para el protocolo TSIG. Está liberado bajo licencia BSD. Las versiones actuales de BIND9 a partir de la 4 brindan soporte mediante el driver DLZ para el almacenamiento de sus datos en Sistemas Gestores de Base de Datos tales como PostgreSQL, MySQL, LDAP, Berkeley DB. Tanto las plataformas BSD, Solaris, Linux y Mac OS X lo tienen pre empaquetado o integrado para su sistema operativo, y de igual modo en Windows puede ser instalado.

1.5 Enfoques para la administración de la información del DNS

El modo en que se almacena la información del DNS en las distintas aplicaciones servidoras, actúa significativamente en el rendimiento de los sistemas que hacen uso de él, e influye igual en el nivel de comodidad para la administración de la aplicación servidora. A continuación se exponen un grupo de enfoques para esta tarea y se destacan además sus principales ventajas y desventajas.

1.5.1 Enfoque Clásico

Se define como **Enfoque Clásico** al uso de ficheros de zona (ficheros de texto) para el almacenamiento de los datos que rigen a una zona en específico. La administración del servidor de DNS siguiendo este enfoque se centra fundamentalmente en mantener estos ficheros actualizados. La

edición de los mismos requiere de conocimientos avanzados en el tema e indistintamente no deja de implicar grandes riesgos si se insertaran mal algunos datos dentro de los nombrados ficheros, constituyendo por tanto este detalle una desventaja significativa. Para que los cambios (actualizaciones) se propaguen y se hagan tangibles en los dispositivos clientes es necesario de una recarga o reinicio del sistema. De esta forma el servidor de DNS compilaría estos ficheros y los almacenaría dentro de la memoria RAM. Obviamente los procedimientos de lectura, compilación y almacenamiento de la información de estos ficheros de zona en la memoria implica un coste significativo en los recursos tiempo y memoria RAM de la computadora que ejecute este servidor; coste que irá en aumento de forma proporcional a la cantidad de entradas dentro del fichero de texto que rige a esta zona, siendo sin lugar a dudas, esta la desventaja más crítica de este enfoque.

1.5.2 Vía WEB mediante Scripts

Propone el uso de una interfaz WEB que permita manipular mejor el contenido de los ficheros de zona. La administración del servidor de DNS se simplificaría mediante el uso de una página WEB con todas las posibles acciones a realizar sobre los ficheros de zona. La modificación de un registro sería desglosada por ejemplo en campos con regiones editables que variarían dependiendo de la naturaleza del registro. Se podrían programar entonces validaciones para estas regiones editables que controlen en todo momento la validez de las inserciones hacia los ficheros que rigen la configuración de las zonas en el servidor. Efectuar dichos cambios físicamente dentro de los ficheros de zona sería una acción delegada a un código script que modificaría los ficheros en cuestión y reiniciaría el servicio para propagar los cambios. Como lenguaje script se podría emplear a perl, python, ruby o cualquiera que permita hacer llamadas al sistema. Lamentablemente este enfoque aún conserva las limitantes existentes dentro del **Enfoque Clásico** para los recursos de tiempo de recarga y empleo de memoria.

1.5.3 Uso de Sistemas Gestores de Base de Datos y Scripts

Básicamente es una extensión del anterior (**Vía WEB mediante Scripts**) y propone hacer uso de lenguajes interpretados para tomar los datos almacenados en un Sistema Gestor de Base de Datos (como PostgreSQL o MySQL) y realizar los cambios dentro de los ficheros de zona. Estos mismos lenguajes interpretados serían los encargados de reiniciar el servicio de DNS, por tal motivo no incorporan optimizaciones en este sentido y mantiene las mismas limitantes del enfoque **Vía WEB mediante Scripts**.

1.5.4 DNS Update

Esta metodología está bien definida en la RFC 2136, donde se expone un proceso por el cual los registros de una zona pueden ser actualizados desde uno o muchos recursos externos (otros servidores de nombre de dominio por ejemplo). La principal limitación en esta especificación consiste en que una zona (o dominio) no puede ser añadida o eliminada dinámicamente pero dentro de una zona existente se pueden adicionar, modificar o eliminar registros (a excepción de los registros de tipo SOA (del inglés: Start of Authority) que definen esencialmente a una zona). El empleo de este enfoque presupone el uso de métodos de seguridad para garantizar la autenticidad de las actualizaciones en pos de mitigar los riesgos contra los distintos tipos de ataques a través de la red. Generalmente se da en conjunción con TSIG y TKEY (del inglés: Transaction KEY) definidos en la RFC 2845 y la RFC 2930 respectivamente. Este enfoque surge para eliminar las limitantes anteriormente expuestas en los otros y logra resolverlas acarreando para su solución el uso las técnicas adicionales de aseguramiento del servidor de nombres de dominio.

1.5.5 DLZ

Mediante el uso de **DLZ** (del inglés: Dynamically Loadable Zones) surge como alternativa del enfoque de **DNS Update**. El empleo del mismo permite cargar y administrar cualquier número de zonas desde un gestor de base de datos. Actualmente **DLZ** soporta los DBMS (del inglés: Data Base Manager System) más difundidos a nivel mundial, como PostgreSQL, MySQL, LDAP, Berkeley DB, además de otras formas de almacenamiento de datos. Usar **DLZ** implica una gran variedad de ventajas, dentro de las cuales se pueden citar que no es necesario la recarga o reinicio del sistema para poder hacer inmediatas las actualizaciones que se puedan efectuar dentro de las zonas que manipule el servidor, además, el consumo de memoria es mucho menor si el contenido de estas zonas están insertadas en una base de datos. Otra ventaja que muestra este enfoque es que sí se pueden insertar zonas completas a diferencia del uso de **DNS Update** donde los registros de recursos de tipo SOA permanecían invariables. Realmente **DLZ** es una funcionalidad propia del BIND9 que puede ser activada (mediante la compilación) para que las peticiones que se le hacen al DNS sean traducidas en consultas SQL y transferidas al sistema gestor de base de datos elegido como contenedor de la información que antes se tenía en un fichero de zona, luego el mismo **DLZ** crea la respuesta con el resultado de esta consulta y se la envía a la aplicación cliente que solicitó el servicio del DNS.

1.5.1.1 Sistema Gestor de Base de Datos para el uso del DLZ

De optar por el uso del DLZ, se propone como sistema gestor de base de datos el servidor PostgreSQL, primeramente porque el TIP posee su directorio y base de datos administrativa bajo este DBMS y es muy recomendable preservar la compatibilidad entre las aplicaciones que manipulan la información del subsistema administrativo y el DNS del TIP. Por otra parte PostgreSQL ha gozado de mucha preferencia dentro de la comunidad de desarrolladores de aplicaciones empresariales por las ventajas que a continuación se exponen:

- **Distribuido bajo licencia BSD:** Esto permite básicamente un número ilimitado de instalaciones sin costes adicionales para los desarrolladores.
- **Buen soporte y disponibilidad de información:** PostgreSQL posee muy buena documentación para el uso y la administración del servidor, independientemente de que posee el respaldo en la web de una comunidad muy activa dispuesta en todo momento a cooperar y a brindar soluciones.
- **Posee soporte para una alta concurrencia:** Mediante el uso del MVCC (del inglés: Multiversion Concurrency Control), este DBMS brinda un enfoque superior eliminando el uso de los bloqueos explícitos hacia la base de datos cuando un proceso o usuario quiera hacer cambios en la misma.
- **Es multiplataforma:** PostgreSQL es actualmente soportado por los sistemas operativos más populares del mercado.
- **Alta gama de herramientas para la administración y diseño de sus base de datos:** Por mencionar algunas se podría hablar de pgAdmin (tanto la versión escritorio como su versión WEB mediante el uso de PHP denominada PhpPgAdmin), pgAccess, psql (herramienta en modo consola), ER/Studio en su versión 7.5 o superiores soporta modelado para este DBMS, entre otras no nombradas en este trabajo.

1.6 Conclusiones

Como bien se ha visto en el transcurso de este capítulo, el establecimiento del protocolo ENUM de manera básica, no implica un costo económico elevado orientado a requerimientos de hardware, pero sí requiere la necesidad de una vasta comprensión de su funcionamiento y una correcta elección de las herramientas a emplear para su puesta en marcha. Hablando de modo específico y orientado hacia el sistema TIP, se podrá observar que existen disímiles características que descartan el uso de

algunas de estas herramientas y modos de administrarlas. En todo momento además, se ha de garantizar al menos que estas posean soporte para los registros de tipo NAPTR y se debería pensar con igual importancia en las licencias y términos de uso por las que se rigen, valorando en darle mayor preferencia a las que promuevan el software libre, debido a la convicción de la Universidad. Existen además pautas de vital importancia para ETECSA que no pueden quedar omitidas, de las cuales se ha de destacar su necesidad por la elección de una herramienta segura y con prestaciones de alto rendimiento, que en conjunto con las otras lógicas de servicios propuestas para el sistema TIP hagan del mismo un servicio confiable para sus usuarios.

Capítulo 2: Herramientas y enfoques para el sistema TIP

2.1 Introducción

En este capítulo se abordará sobre las herramientas a emplear en el sistema TIP, específicamente sobre el modo de instalación de las mismas y la configuración de estas sujetas a las particularidades del protocolo ENUM, el cuál impone el uso de enfoques administrativos especiales. Además se presenta lo que podría constituir una propuesta de despliegue sostenida sobre una arquitectura distribuida para las tecnologías de base de datos.

2.2 Elección del servidor y enfoque administrativo para el DNS orientado al ENUM

El sistema TIP posee características específicas que influyen a gran escala sobre la elección de la aplicación para el servidor de DNS y los enfoques de almacenamiento a utilizar en la misma.

Por ejemplo, cuando una persona decide unificar todos sus contactos en un solo número ENUM (igualmente conocido como número TIP) haciendo uso del sistema TIP, pide una subscripción que será atendida por un registrador. Este registrador le solicita datos propios y necesarios para la administración y control del sistema en general, como el/los nombre(s), apellidos, número de identidad, formas de contactarle, entre otros campos obligatorios para comenzar a validar la subscripción. Dichos datos conformarán parte de la base de datos administrativa del sistema TIP que a su vez funge como directorio. Una vez validados los mismos, han de ponerse visibles en el DNS, y esto se logra mediante una inserción de N registros de tipo NAPTR en la zona ENUM a la que pertenezca el subscriptor, donde N coincidiría con la cantidad de contactos que se desean hacer públicos. Como ya se presupone, toda la información necesaria para conformar estos registros, ha sido almacenada y validada durante el proceso de subscripción. Ahora debería analizarse la siguiente pregunta, ¿sería lo más óptimo realizar la inserción de estos registros campo a campo en el DNS mediante la intervención de una persona, cuando la información existe y está disponible en una base de datos posibilitando la automatización? Obviamente mientras menos responsabilidades se deleguen a seres humanos en actividades tan críticas como esta, menos riesgos existen de dejar en estados inconsistentes al DNS. Por tanto se hace inminente que los enfoques escogidos para el almacenamiento y administración de la información del DNS orientado al sistema TIP contemple la sincronización con la base de datos del mismo.

Otra característica del sistema TIP que impone un comportamiento dinámico al DNS enfocado en el ENUM viene dada por la posibilidad que se le brinda a los subscriptores de elegir una planificación que regule la disponibilidad de algunos registros para un momento dado del día.

Por ejemplo, si un subscriptor posee dos números telefónicos, uno particular y otro laboral, podría definir que en horario laborable el particular no figurara dentro de la respuesta cuando se le haga la consulta al DNS, dejando solo visible el teléfono laboral, y viceversa para el horario no laborable. El subscriptor tiene la posibilidad de establecer este comportamiento para los rangos de tiempos que él defina, pero para todos los casos, lo más crítico sería que este comportamiento ha de ser aplicado a diario.

Como es de esperar, ninguno de los enfoques que posean la limitante de que para actualizar se necesite de un reinicio o recarga del servidor DNS, tiene cabida en este ambiente tan dinámico, pues a medida que comiencen a poblarse las zonas, y se alcance un número elevado de registros, el proceso de recarga del servidor DNS podría ser más frecuente y por tanto se pasaría más tiempo recargando que respondiendo consultas que se le hagan.

Por tales motivos, se va acotando cada vez más la lista de opciones en cuanto a herramientas a utilizar y enfoques para aplicar. Por un lado sobre sale la aplicación PowerDNS que admite el almacenamiento de sus zonas en un Sistema Gestor de Base de Datos, pudiéndose poblar estas zonas con los datos del sistema TIP alojados en su base de datos administrativa. Y como otra opción se tiene a BIND9 haciendo uso del driver DLZ para lograr un comportamiento similar que PowerDNS. El resto de las aplicaciones y enfoques quedan como opciones descartadas por las limitantes que presentan frente al dinamismo del sistema TIP.

La entidad que contrata los servicios de la UCI para la creación del sistema TIP es ETECSA. En todo momento se ha hecho por parte de la misma alusión al uso del BIND9 como servidor para el DNS, debido a que ya ellos poseen algunos dominios soportados con aplicaciones de esta clase, y es con ellos con los que están familiarizados. La elección del BIND9 como servidor para el DNS puede sustentarse además por su soporte para algunas facilidades en cuanto la seguridad, como por ejemplo para el protocolo TSIG, en cambio PowerDNS como ya se había visto en el capítulo anterior, no posee esta ventaja. Por tales motivos, se ha elegido a BIND9 haciendo uso del DLZ para dar soporte a los DNS’s del sistema TIP.

2.3 DLZ y su integración con el DBMS PostgreSQL

DLZ en un principio surge como un parche para dotar al BIND9 con la capacidad de actualizar sus zonas sin necesidad de perder tiempo en recargas o reinicio del sistema, permitiéndole además del uso de los enfoques de almacenamiento mediante ficheros de texto, el almacenamiento de este mismo tipo de información pero en un servidor de base de datos. Abundando un poco más a fondo: en realidad, los desarrolladores de BIND9 propiciaron un conjunto de interfaces para integrar su aplicación con base de datos, las cuales son denominadas como el conjunto SDB (del inglés: Simplified Database Interface) e incluidas a partir de la versión 9.1. El desarrollo del driver DLZ entonces se simplificó en reutilizar estas interfaces para dar soporte a un grupo de sistemas gestores de base de datos, pero a diferencia de otros drivers que igualmente hicieron uso del SDB, este no limitó a los usuarios al uso de un esquema en específico para sus tablas, sino que bastaba con tener toda la información disponible y un modo (generalmente una consulta o llamada a una función) para hacérsela llegar al driver. Por otra parte, nunca solapó el uso de los ficheros de zona, ambos enfoques pueden coexistir en un mismo servidor de DNS y operar correctamente. Sin lugar a dudas fue la flexibilidad lo que posibilitó que DLZ ganara en popularidad, y obtuviese un visto bueno por parte de la fundación NLnet[1] desde los inicios del proyecto, la cual accedió a patrocinar el desarrollo del mismo siempre que fuera un proyecto de código abierto y disponible para todo el mundo. Luego, a partir de la versión 9.4 ya DLZ podía ser incluido dentro del núcleo del BIND9 puesto que su código fue considerado como parte de las fuentes de este servidor de DNS.

2.3.1 Habilitando el uso del DLZ en BIND9

La inclusión del driver DLZ dentro del núcleo del servidor BIND9 solo se logra mediante una compilación de las fuentes de este último que solicite y habilite la integración entre ambos.

En la mayoría de los sistemas basados en GNU/Linux este proceso consta de 3 pasos básicos ordenados de la siguiente forma:

1. *Configuración del proceso de compilación*: En este proceso el usuario establece los valores de aquellas opciones que rigen el comportamiento de la aplicación que desea instalar. Posteriormente el proceso hace un análisis del sistema y en caso de que se cuenten con todos

[1] **NLnet**: Fundación que patrocina proyectos en pos del desarrollo de Internet.

los prerrequisitos para las opciones que pidió el usuario, pues se genera un "Makefile", el cual servirá de entrada para el siguiente paso.

2. *Compilación y generación de binarios:* En este proceso los compiladores instalados en el sistema haciendo uso de la información alojada en el "Makefile" toman el código fuente de la aplicación y generan los ficheros binarios para su posterior ejecución con las opciones preestablecidas en el punto anterior.
3. *Instalación de los ficheros binarios:* Se resume fundamentalmente en realizar tareas relacionadas con ubicar los binarios y otros ficheros de salida del proceso anterior en los directorios que les correspondan.

Las herramientas que intervienen en el proceso de instalación desde las fuentes en la mayoría de los sistemas operativos GNU/Linux están incluidas dentro de la misma distribución o se encuentran disponibles en sus respectivos repositorios de aplicaciones y paquetes. Específicamente en el caso de los basados en Debian, el paquete alojado en los repositorios, denominado build-essential, provee herramientas como compiladores de C y C++, la herramienta de generación o automatización de código “make” e instala conjuntamente una serie de dependencias que permitirán hacer el procedimiento de configuración, compilación e instalación del servidor BIND9.

A continuación se hará un desglose de todas las operaciones a ejecutar para que se lleve a cabo la integración del servidor BIND9 con el DLZ. Se ha de recalcar al lector que todas las operaciones subsiguientes han de ser ejecutadas con privilegios administrativos para que puedan tomar efectos. Se asume además como prerrequisito, la puesta en marcha y correcta configuración del sistema gestor de base de datos PostgreSQL en la estación de trabajo donde se monte el nodo del DNS mediante BIND9.

Se parte entonces desde descargar las fuentes del servidor BIND9, lo cual puede hacerse desde la siguiente dirección web: http://www.bind9.net/download. El actual documento fue redactado basándose en las fuentes del bind-9.5.0-P2, por tal motivo el fichero descargado fue el bind-9.5.0-P2.tar.gz.

Como bien indica la extensión de este archivo, se trata de un contenedor comprimido (igualmente conocido como archivador). Para extraer su contenido se inserta en la consola la siguiente línea de comandos:

1. *tar –xvzf bind-9.5.0-P2.tar.gz*

Esta orden extrae todo el contenido de las fuentes para este release[2] del BIND9 dentro de una carpeta nombrada bind-9.5.0-P2. Introducirse dentro de esta y comenzar el proceso de configuración para la instalación se logra con las siguientes órdenes:

2. *cd bind-9.5.0-P2*

3. *./configure --with-dlz-postgres=yes --sysconfdir=/etc/bind --enable-libbind --with-libtool --with-openssl --enable-threads*

Dentro de la tercera instrucción se destaca el uso de ***./configure***, lo cual indica una llamada a ejecución del fichero script denominado ***configure*** dentro la carpeta de las fuentes. Todo lo sucesivo y precedido por "**--**", son argumentos de esta llamada, que establecen las opciones que se desean contempladas durante el posterior proceso de compilación del código fuente. Nótese como se solicita que durante la compilación se incluya la funcionalidad del driver DLZ, específicamente integrado al gestor de base de datos PostgreSQL con el argumento ***--with-dlz-postgres=yes***. La opción ***--sysconfdir=/etc/bind*** preestablece al directorio /etc/bind/ como punto de ubicación de los ficheros de configuración por defecto; será en este directorio donde se ubiquen posteriormente los ficheros que el servidor BIND9 lee para cargar su configuración. Es importante recalcar que de existir este directorio todo su contenido será eliminado, y en caso de que no exista pues será creado el directorio solamente. Con el argumento ***--enable-libbind*** se solicita que se construyan además las librerías compartidas (del inglés: shared libraries) *libbind* que utiliza el servidor para su ejecución. La opción ***--with-libtool*** demanda el empleo de la herramienta *libtool* del proyecto GNU durante el proceso de construcción. Si se desea hacer uso además de las extensiones de seguridad para BIND9 (DNSSEC) es obligatorio la inclusión del argumento ***--with-openssl***. Ya para terminar se determina que la aplicación resultante pueda soportar el multiprocesamiento siempre que el sistema operativo lo provea con la opción ***--enable-threads***.

El proceso de configuración puede tomar algunos segundos, siempre en dependencia de las prestaciones del computador en el que se ejecute. En todo momento su comportamiento por defecto es imprimir el estado de cada paso por la salida estándar del terminal donde ha sido lanzada la instrucción; de esta forma, el usuario puede ver y analizar dichas salidas, cancelar el proceso para habilitar nuevas opciones o corregir las que insertó.

[2] **Release**: Liberación (publicación) de la versión de un software en específico.

Durante este proceso, el script de configuración hace llamadas a diversas herramientas, dentro de las cuales se debe citar por su importancia a *pg_config*, cuya utilidad es la de proveer una gran variedad de información acerca del servidor de base de datos PostgreSQL instalado. Fundamentalmente facilita la ubicación de sus archivos de cabecera y sus librerías de código objeto, imprescindibles para enlazar el BIND9 con el servidor de PostgreSQL. Es un error asumir que el simple hecho de haber instalado el DBMS y tenerlo funcionando correctamente es suficiente para disponer de esta valiosa aplicación. En este punto no todos los sistemas GNU/Linux tienen igual comportamiento, pues en sistemas como en CentOS la herramienta es instalada automáticamente cuando el administrador de sistema emplaza el DBMS pero en otros basados en Debian (Ubuntu por ejemplo), solo se instala esta cuando se le pide la incorporación del paquete *libpq-dev*. En caso de que no se cuente con este paquete, entonces, la opción debe ser llamada en la orden con el formato ***--with-dlz-postgres=/path***, siendo */path* el camino hacia la raíz de la instalación del DBMS PostgreSQL.

Por otra parte, un similar comportamiento tiene el uso de la opción ***--with-openssl***, la cual requiere que se le proporcione el camino hacia el directorio que contiene al archivo de cabecera denominado *opensslv.h*. En caso de que no se encuentre dentro de sus ubicaciones por defecto, pues será requerido que se haga la llamada del script de configuración con esta opción bajo el formato ***--with-openssl=/path*** siendo */path* la ruta hacia la raíz del directorio que contenga esta cabecera. El lector puede auxiliarse de algunas utilidades de los sistemas GNU/Linux para encontrar diversos ficheros o caminos si conoce el nombre parcial de ellos. Por ejemplo, actualice la base de datos de nombre de ficheros con el comando ***updatedb***, suele tomar unos segundos o a veces minutos, en dependencia de cuán poblado esté su sistema de ficheros y de cuán potente sea el microprocesador del computador. Una vez concluida la ejecución del comando ejecute un ***locate opensslv.h*** y si este existe le dará la ruta completa a reemplazar por /path en el argumento con nuevo formato para el uso del SSL. Si no existe el camino, entonces, ha de ser instalado el paquete que le provee de este archivo de cabecera. En sistemas basados en Debian (como Ubuntu), este se denomina *libssl-dev* mientras que en sistemas basados en Red Hat (como CentOS) el mismo se denomina *openssl-devel*.

De terminarse el proceso de configuración de manera exitosa, se generará un archivo denominado “Makefile”, el cual contendrá toda la información que necesitan los compiladores del sistema para construir los binarios y ubicarlos posteriormente en sus respectivas ubicaciones dentro del árbol de directorio. La próxima fase será entonces la de compilar, y la misma se ordena con la instrucción:

4. *make*

Esta instrucción sin lugar a dudas ha de ser la más morosa. El proceso de compilación pondrá a prueba las potencialidades del microprocesador y la paciencia de los administradores del sistema. Por la salida estándar, al igual que en el proceso de configuración, se podrá ir viendo las disímiles acciones que ejecutan los compiladores y enlazadores del sistema operativo. Serán creados entonces diversos ficheros que contienen código objeto listos para ser enlazados y suministrarle funcionalidades agregadas a los binarios que se incluyen en el paquete BIND9 para el trabajo con el DNS. Una vez culminado, solo resta ubicar los binarios generados, esto se logra con la instrucción:

5. *make install*

Sin dudas esta instrucción es la de menos consumo de tiempo. Una vez llevada a término, se podría a modo de prueba hacer una ejecución del comando siguiente:

6. *named –v*

Si la salida mostrada es *BIND 9.5.0-P2* se ha corroborado dos detalles importantes: primero que la versión de la aplicación instalada coincide con la que desde un principio se está tratando, y segundo con que se encuentra mapeada dentro de un directorio que permite el acceso de la aplicación sin el uso de rutas completas para su llamada. Este ha de ser siempre */usr/local/sbin/named* pero para estar seguro se puede hacer uso de la herramienta *type* y ver si los caminos de salida coinciden. Se hace la salvedad de que a pesar de que el servidor de nombres de dominio se denomina BIND9, el binario generado durante el proceso de compilación y alojado en el sistema no porta igual identificador, el mismo se denomina "*named*". Generalmente la terminación con la letra "*d*" en el identificador de algunas aplicaciones en GNU/Linux indica que se trata de un "demonio" (del inglés: daemon), como por ejemplo "*syslogd*", "*smbd*" o "*acpid*" por citar algunas de estos sistemas operativos.

No solo el binario "*named*" ha sido alojado en el sistema como parte de todo el proceso de instalación. Compilar el paquete de BIND9 además provee de diversas herramientas como "*named-checkconf*": una herramienta para chequear la sintaxis del fichero de configuración de "*named*"; "*named-checkzone*" y "*named-compilezone*": herramientas para validar y convertir respectivamente ficheros de zona. De igual modo un conjunto de librerías y otros tipos de ficheros fueron generados y alojados para que la suite completa de BIND9 se ejecute correctamente.

Pero la instalación utilizando las fuentes, no provee una funcionalidad crucial y que ha de estar presente para todos los servicios (o al menos en los demonios del sistema operativo); la misma es la

capacidad de iniciarse o detenerse de modo correcto y automático cuando la estación en la que residen es iniciada o apagada.

Esta carencia puede ser solucionada sabiendo que los sistemas operativos GNU/Linux poseen una estrategia interesante haciendo uso de scripts de ejecución que llaman a los servicios. A continuación se detalla esta estrategia con mayor profundidad para el lector.

Un ordenador que trabaja bajo GNU/Linux contiene 10 estados de ejecución (igualmente conocidos como niveles de ejecución) pero solo opera en uno de ellos. Los estados con los que más se interactúan son el de apagado (estado 0), el de modo mono usuario (estado 1), los estados en modo multiusuario con sus diversas particularidades (estados del 2 al 5) y el de reinicio del sistema (estado 6). Cuando el sistema operativo se inicia, recurre a la ejecución de una serie de rutinas y posteriormente entrará en uno de los estados del 2 al 5 en los cuales se ejecutan una gran diversidad de scripts de inicialización para los servicios que se quiere que sean suministrados en el nivel al cual se ha entrado. En el momento en que la estación es apagada o reiniciada pues se entra en el nivel de ejecución 0 ó 6 respectivamente.

Ahora bien, de seguro surge la interrogante de cómo el sistema operativo sabe qué scripts ejecutar y cuáles no. Para ello se crearon una serie de directorios dentro del sistema de ficheros los cuales son accedidos en dependencia del estado de ejecución en que se encuentre y leyendo su contenido se sabe entonces los scripts a ejecutar y su orden. Estos directorios son ***/etc/rc0.d***, ***/etc/rc1.d***, ***/etc/rc2.d*** y así sucesivamente para cada uno de los niveles de ejecución. En su interior estos directorios no albergan los scripts de inicialización sino un enlace hacia la ubicación de ellos, la cual por defecto ha de ser en ***/etc/init.d***. Este script de inicialización ha de cumplir algunos requerimientos básicos, como por ejemplo, hacer una correcta llamada al servicio que se desea ejecutar con las opciones y argumentos para el mismo bien establecidas; y además, ha de proveer la posibilidad de que se le puedan pasar como argumentos las cadenas ***start*** y ***stop*** para el inicio y la detención respectivamente del servicio en cuestión.

En los sistemas basados en Debian existe una plantilla con la cual elaborar un script de inicialización se vuelve un proceso sumamente intuitivo. La misma se encuentra en la carpeta ***/etc/init.d/*** y se denomina ***skeleton***.

Una vez confeccionado este script y ubicado en la carpeta referida, pues solo resta informar en cuáles niveles de ejecución se desea llamar. Para ello dentro de la carpeta respectiva al nivel de ejecución se crea un enlace con una de las siguientes líneas de comandos:

ln –s /etc/init.d/nombre_script_inicialización S25named

ln –s /etc/init.d/nombre_script_inicialización K25named

Ambas líneas apuntan hacia el mismo script de inicialización y solo difieren en el uso de la ***S*** o la ***K*** para el identificador del enlace. Cuando un enlace en estos directorios comienza con ***S*** el script hacia el cual apunta es ejecutado con el argumento ***start***, en caso de inicializarse el identificador del enlace con la letra ***K***, el script hacia el cual apunta será ejecutado con el argumento ***stop***. El número ***25*** permite establecer una preferencia entre los restantes enlaces contenidos dentro de la carpeta del nivel de ejecución en la que se esté trabajando. Ha de hacerse la observación al lector de que el identificador del enlace no tiene una funcionalidad más allá que la de nombrarle.

Es recomendable en el caso del servidor de DNS crear los enlaces para las llamadas de ***start*** en las carpetas referentes a los estados de ejecución del 2 al 5 y para las llamadas con argumento **stop** en los estados de ejecución 0, 1 y 6.

2.3.2 Configuración del servidor de nombres de dominio

Por defecto, el programa ***named*** tiende a buscar dentro del directorio definido en el argumento ***--sysconfdir*** durante su proceso de compilación, un fichero denominado ***named.conf***. Este fichero es su fichero primario de configuración y el mismo le sirve como punto de partida para cargar todas las opciones de comportamiento e incluso referenciar a otros ficheros que contenga información u opciones de ejecución para él.

La presente investigación no pretende de manera exhaustiva abarcar todo lo referente a la configuración de un servidor BIND9, pero sí se considera necesario que el lector tenga noción de algunos conceptos aplicados en el enfoque clásico (ficheros de texto), pues a la larga, el DLZ podría verse como el paso de estos fichero de texto al mundo de las base de datos relacionales.

En el enfoque clásico, dar soporte a un dominio (de igual modo conocido como zona para los efectos prácticos) es tan sencillo como hacer uso de la cláusula ***zone*** para indicar en el ***named.conf*** la

presencia del dominio. Así por ejemplo el dominio 3.5.e164.etecsa.cu sería dado de alta de la siguiente forma:

```
zone "3.5.e164.etecsa.cu" {
    type master;
    file "/etc/bind/db.fichero";
};
```

Donde ***type master;*** indicaría el modo en que trabaja el servidor para esta zona, lo cual queda claro que es de forma maestra, o sea, el servidor es primario para la zona ***3.5.e164.etecsa.cu*** y por tal motivo se comporta autoritario para la misma.

La sentencia ***file "/etc/bind/db.fichero";*** dentro de la cláusula ***zone*** indica en qué fichero se encuentran definidos los distintos registros que dan soporte a este dominio. A continuación se expone una porción de lo que podría figurar como su contenido:

```
$TTL    604800
3.5.e164.etecsa.cu. IN  SOA querubin rherrpinark.uci.cu. (
                    3     ; Serial
               604800     ; Refresh
                86400     ; Retry
              2419200     ; Expire
               604800 )   ; Negative Cache TTL
;
@    IN  NS  querubin
querubin    IN  A   10.7.20.2
7    NAPTR 11 120 "u" "E2U+sip" "!^.*$!sip:a@example.com!" .
7    NAPTR 11 130 "u" "E2U+msg" "!^.*$!mailto:b@example.com!" .
```

La directiva ***$TTL*** como ya se había dicho, impone un valor por defecto a todos los registros incluidos en lo sucesivo a su aparición para el tiempo de almacenamiento expresado en segundos en la caché de otros servidores; independientemente de que cualquier registro puede definir un valor propio

haciendo uso del campo ***TTL*** presente en la definición propia del mismo. Ha de notificarse al lector de dos aspectos importantes en la descripción de esta porción: primeramente que en una línea del fichero todo lo sucedido después de un punto y coma (***;***) es tomado como comentario y por tal motivo ignorado por la aplicación; y segundo que no es lo mismo en el contexto de este fichero hacer uso de un nombre de dominio que termine por "***.***" (nodo raíz) a terminarlo en ausencia de este. Se explica mejor la segunda salvedad:

En el ejemplo, el uso de las cadenas "***querubin***" y "***rherrpinark.uci.cu.***" tienen comportamientos distintos en el momento de ser tratados por el servidor: a la primera se le trata concatenándole el dominio en la que se encuentra ("***querubin.3.5.e164.etecsa.cu.***") y la segunda no sufre cambios.

El primer registro que aparece es el registro de recurso de tipo SOA, y su inclusión es obligatoria y única (solamente una vez) para todas las zonas. En el ejemplo se ilustra de forma partida por líneas haciendo uso de la notación con paréntesis para desglosar cada uno de sus campos, pero el mismo puede ponerse en una sola línea siempre que sus campos estén separados por espacios. Ejemplo:

3.5.e164.etecsa.cu. IN SOA querubin rherrpinark.uci.cu. 3 604800 86400 2419200 604800;

La línea que contiene "@ ***IN NS querubin***" es sencillamente la inclusión de un registro de tipo NS (del inglés: Name Server) y en sí se indica con el mismo, el nombre de un ordenador que actúa como servidor autoritario para esta zona. Ha de estar presente al menos una ocurrencia de este registro. Nótese el uso del caracter "@", el cual es usado para abreviar y no poner de nuevo el nombre de la zona.

La línea siguiente muestra la dirección IP del servidor DNS que funge como autoritario para la zona que se está definiendo, haciendo uso de un registro de tipo A.

Y las últimas dos líneas proveen unas entradas de dos registros de tipo ***NAPTR*** para el nombre de dominio ***7.3.5.e164.etecsa.cu***, los cuales por su simpleza han de verse solo de modo ilustrativo para simular unas entrada de un dominio ENUM en este DNS.

Estos 5 registros definen de un modo muy escueto el contenido de una zona que podría dar soporte a un dominio para el sistema TIP haciendo uso de los ficheros de zona.

Pero el uso del DLZ posee una óptica distinta al momento de configurar el BIND9 para dar soporte a sus zonas. Si bien el enfoque mediante el uso de los ficheros contiene de modo preciso la información y los registros que atañen a sus zonas, la configuración del DLZ provee un modo de consultar a un

DBMS predefinido por si existe una respuesta para la información solicitada al servidor de DNS. En caso de existir, entonces es que es retornada, pero el servidor de DNS a priori desconoce si es autoritario o no a una solicitud que se le haga; él sencillamente haciendo uso del lenguaje de consultas que emplea el DBMS para el cual fue compilado, descubre su comportamiento (autoritario o no) para las solicitudes que los clientes le efectúan.

En las últimas versiones del BIND9 ya viene reconocida la cláusula para el uso del DLZ y se puede definir entonces en el mismo fichero ***named.conf*** toda la lógica de consultas y los parámetros de configuración para la interacción con el DBMS. Por motivos de organización se recomienda al lector mantener la configuración del DLZ separada al fichero de configuración del servidor de DNS, y solo dentro del ***named.conf*** hacer una referencia al fichero donde se encuentra dicha configuración mediante el uso de la cláusula ***include***. La siguiente línea ilustra mejor con un ejemplo:

include "/etc/bind/dlz.conf";

En el fichero referenciado se pondría la configuración del DLZ y al momento de carga del servidor BIND9, el mismo será leído y tratado como si estuviese incluido dentro del ***named.conf***.

Pero definir un fichero de configuración es un proceso que ha de estar precedido de un buen análisis del modo en que se alojarán los datos del servidor DNS en el DBMS. Resulta muy aconsejable contar con un esquema de base de datos que soporte los registros y esto se debe a que el fichero de configuración del driver DLZ en su mayor parte está conformado por consultas que serán utilizadas en algunos métodos del DLZ y no se considera una buena práctica comenzar de modo empírico a redactar dichas consultas sin un previo conocimiento de la estructura de la base de datos que retiene los registro necesarios.

La Tabla 1 a continuación presenta un esquema propuesto como ejemplo para el almacenamiento de los registros en el DBMS PostgreSQL, pero siempre se hace constancia en que una de las particularidades fundamentales del driver DLZ es su flexibilidad, por tal motivo los administradores de los servidores de DNS quedan en total libertad de hacer un diseño de base de datos más acorde a las necesidades de su sistema.

Columna	Tipo de Dato
zona	text

host	text
ttl	integer
type	text
mx_priority	integer
data	text

Tabla 1.Esquema propuesto para la tabla[3] ***"dns_records".***

Si se hace una comparación entre el modo en que se almacenan los registros cuando están representados por un esquema con enfoque de ficheros y el modo en que se almacenan en el mundo de las base de datos relacionales, se pueden identificar campos análogos. Por ejemplo, todo registro pertenece a una zona en específico, posee un valor para el ***TTL*** ya sea referenciado por la directiva ***$TTL*** o explícitamente en la declaración del registro, además todo registro posee un identificador dentro de una zona (conocido como campo ***name*** para el enfoque de ficheros y representado por el campo ***host*** en la tabla ***dns_records***) y de manera general todos poseen un valor que devolver siempre y cuando se le consulte, que varía su estructura en dependencia de la naturaleza del registro en cuestión y este valor sería el almacenado en el campo ***data*** del esquema propuesto.

Un ejemplo vale más que mil palabras; por tal motivo se mostrará en la Tabla 2, todo el ejemplo de fichero de zona expuesto al inicio de esta sección en su representación homóloga en el mundo de las base de datos relacionales, específicamente haciendo uso del esquema propuesto. Nótese como el campo ***data*** posee diversos volúmenes de valores, que en todo momento va regido por el tipo de registro respectivo (almacenado en el campo ***type*** de la tabla dns_records).

zona	host	ttl	type	mx_priority	Data
3.5.e164.etecsa.cu	@	604800	SOA	NULL	querubin rherrpinark.uci.cu. 3 604800 86400 2419200 604800
3.5.e164.etecsa.cu	@	604800	NS	NULL	querubin
3.5.e164.etecsa.cu	querubin	604800	A	NULL	10.7.20.2
3.5.e164.etecsa.cu	7	604800	NAPTR	NULL	11 120 "u" "E2U+sip"

[3] ***Tabla***: Término de igual modo conocido como relación en términos de base de datos relacionales.

					"!^.*$!sip:a@example.com!" .
3.5.e164.etecsa.cu	7	604800	NAPTR	NULL	11 130 "u" "E2U+msg" "!^.*$!mailto:b@example.com!" .

Tabla 2.Paso del ejemplo enfocado en ficheros al enfoque de base de datos relacionales.

Retomando el campo ***data***, es necesario interiorizar la importancia de separar por un espacio los valores que lo componen en caso de ser más de uno. De igual modo se puede almacenar en la base de datos en campos separados (o campos auxiliares) los distintos valores de este campo que en su conjunto lo comprendan en su totalidad, pero se ha de tener en cuenta que siempre su retorno ha de estar englobado como un solo campo (el campo ***data***), y para lograr tal efecto podría usarse la concatenación entre los sub campos siempre y cuando el orden de retorno de los mismos coincida con la estructura de retorno del registro subyacente. Este tipo de filosofía podría resultar engorrosa, y solo se aplicaría para facilitar la lógica administrativa del servidor de DNS. Pero en todo momento ha de prestarse vital atención en la generación de valores nulos para las ocurrencias de registros que no tengan por qué llenar estos campos auxiliares.

Teniendo definido el esquema de la base de datos y algunos valores insertados en la misma, se procede entonces a pasar al proceso de análisis de la configuración del DLZ alojado en el fichero dlz.conf.

La cláusula para definir una zona haciendo uso del driver DLZ es precisamente ***dlz*** y posee algunas similitudes con la cláusula ***zone*** empleada en el enfoque de ficheros presentada al inicio de este acápite. A continuación en la Figura 4 se presentan ambas cláusulas juntas y se espera que el lector verifique las similitudes identificadas, las cuales son resaltadas con iguales colores para el caso de los términos homólogos:

- Ambas cláusulas después de su llamada requieren de una cadena encerrada entre dobles comillas (***“”***).
- Sus bloques de definición se encuentran agrupados entre los símbolos llave de apertura para el inicio (***{***) y llave de cierre seguido de un punto y coma para cerrar el bloque (***};***).

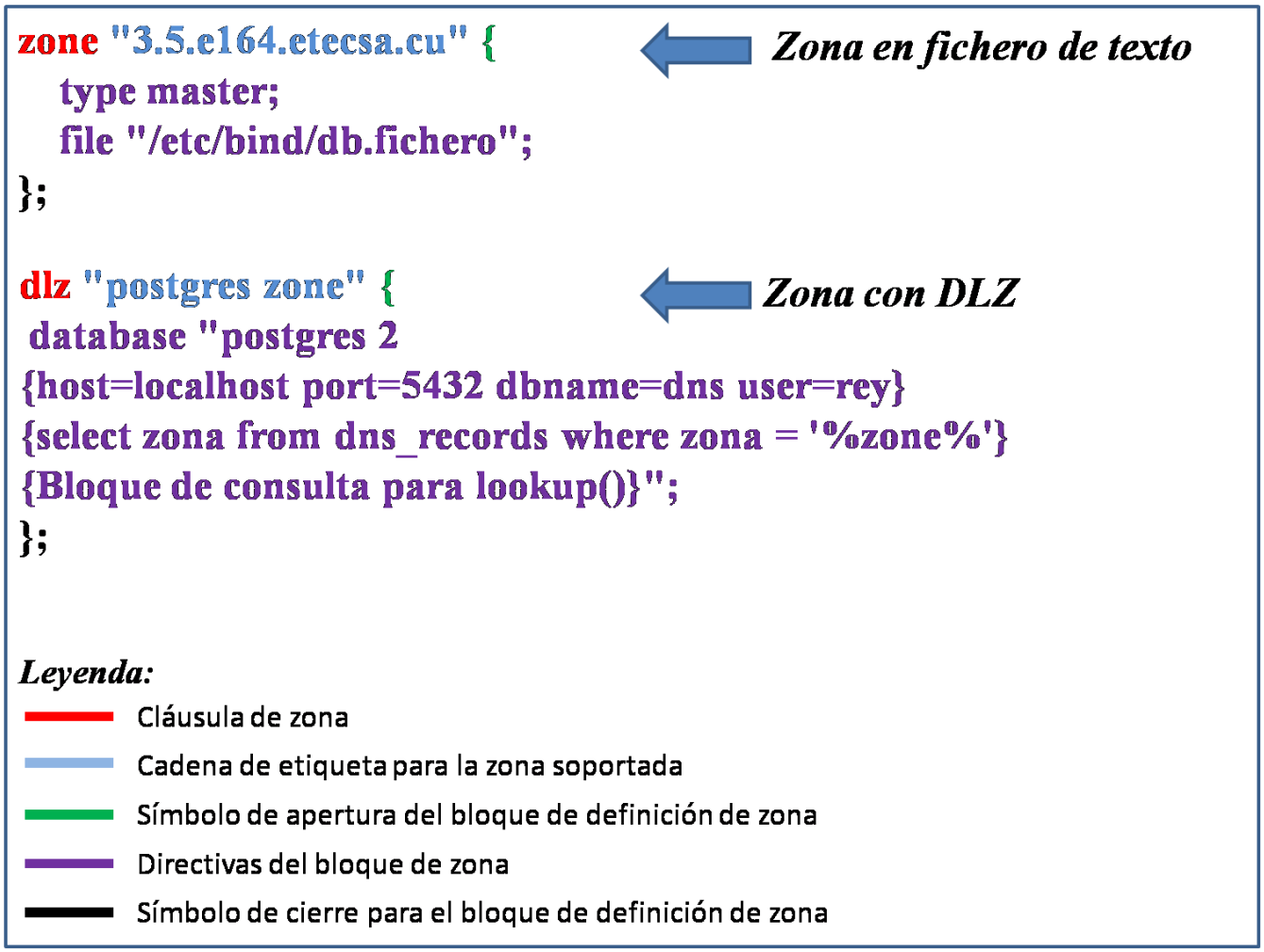

Fig.4.Similitudes entre cláusulas de diferentes enfoques.

Se explicará entonces de modo mesurado todo lo expuesto para la cláusula ***dlz***. La primera línea, ***dlz "postgres zone"*** notifica al servidor de DNS que se va a hacer uso de la funcionalidad DLZ y que todo este segmento de configuración tiene como etiqueta a "**postgres zone**". Esta línea es usada de forma particular por el BIND9 durante la rutina de reinicio para mandar mensajes más ilustrativos en caso de existir algún error mientras el procesamiento del fichero de configuración en general se efectúa.

La segunda línea, "***database "postgres 2***", notifica al servidor de DNS que se hará uso del driver para DLZ integrado con el DBMS PostgreSQL y se solicitarán 2 conexiones hacia la base de datos, las cuales se mantendrán abiertas mientras se encuentre corriendo el servicio de DNS. Si el servidor de DNS fue compilado para no hacer uso de multiprocesamiento (multihilos) sin importar el número que se ponga para este campo él siempre abrirá una sola conexión, pero siempre poner un valor entero en este campo es obligatorio. Se ha de hacer una aclaración en este momento, y es que el driver siempre

mantiene abiertas las conexiones que se le especifican en su configuración. Durante los períodos de inactividad no las libera para luego reabrirlas cuando las necesite nuevamente, en ningún momento las cede.

La tercera línea, ***{host=localhost port=5432 dbname=dns user=rey}***, indica la cadena de conexión hacia la base de datos y obviamente es obligatoria. La cadena de conexión de cada driver posee parámetros distintos, pero el driver DLZ se rige por el estándar de cada uno de los DBMS's que soporta, por tal motivo, ante cualquier duda con este campo que no sea contemplada en este documento puede referirse al manual del DBMS en cuestión. En este caso particular se quisiera aclarar que el driver DLZ se muestra bastante flexible en cuanto a los distintos modos de conexión hacia el DBMS contra el cual interactúa, por ejemplo en lugar de "***localhost***" perfectamente se pudo haber esclarecido la dirección IP de un DBMS PostgreSQL que se encuentre operando y escuchando peticiones por el puerto 5432. El resto de los parámetros se consideran auto explicativos y solo en aras de abundar se esclarece al lector que el parámetro referente a la contraseña de conexión se omite ya que se han confiado todas las conexiones para el usuario "***rey***" que vengan desde "***localhost***" en el fichero "***pg_hba.conf***", el cual regula el comportamiento de autenticación para el DBMS PostgreSQL. Nótese además el uso de las llaves para esta línea. Las llaves proveen un modo de agrupar diferentes argumentos pasados al driver DLZ, han de estar incluidas siempre en un orden lógico y se ha de tener siempre en cuenta de que el uso de los espacios dentro de las mismas no es ignorado en ningún momento, por ejemplo, poner {alguna cosa} no es lo mismo que poner { alguna cosa }. Por otra parte, dado que el uso de las llaves es específicamente para mantener organizada la estructura de los argumentos del driver, anidarlas no es correcto y provocaría resultados inesperados para la ejecución del servidor de DNS.

La cuarta línea, ***{select zona from dns_records where zona = '%zone%'}***, posee todas las características de una consulta SQL (del inglés: Structured Query Language) salvo que la cláusula "***where***" contiene una cadena poco común, se hace referencia a ***'%zone%'***. Esta pequeña cadena constituye uno de los símbolos (de igual modo conocido como token) que utiliza el driver para poder reemplazar en su lugar información proveniente desde los clientes del servidor DNS. Nótese como se encuentra encerrado entre los símbolos ***'%%'*** la palabra ***zone***, verá en lo sucesivo como además de este token existen otros de los cuales igualmente se abundará.

Retomando la línea ***{select zona from dns_records where zona = '%zone%'}***, se ha de esclarecer que es una consulta utilizada por el método "***findzone()***" del driver y su uso es obligatorio. La base de

datos a la cual se ha conectado ("***dbname=dns***") ha de poseer una tabla con el nombre "***dns_records***" que contenga un campo "***zona***" cuyos valores contemplarán las zonas a las que el servidor será autoritario. El método "***findzone()***" del driver reemplaza automáticamente en la consulta anterior, la ocurrencia del símbolo ***'%zone%'*** por la cadena de nombre de dominio consultada desde las aplicaciones clientes hacia el servidor de DNS. Por ejemplo, si un cliente del DNS hace una consulta por "***pc1.example.com***", el token ***'%zone%'*** será reemplazado en su totalidad del modo anteriormente expuesto y la consulta quedaría preparada por el driver de la siguiente forma: "***{select zona from dns_records where zona = 'pc1.example.com'}***". Acto seguido, la consulta es transferida al DBMS (PostgreSQL en el presente caso) y de encontrarse al menos una tupla[4] para retornar, se asume entonces que el servidor DNS es autoritario para la zona "***pc1.example.com***" y se continúa efectuando las consultas de las restantes líneas sucesivas del fichero de configuración. Si no se retorna nada, entonces se asume que no es autoritario para dicha zona y se procede entonces a hacer un corte en el nombre de dominio suministrado por el cliente (se elimina lo que para esta iteración implica el dominio de menor nivel), quedando preparada para una nueva iteración la consulta de la siguiente forma: ***{select zona from dns_records where zona = 'example.com'}***. Así sucesivamente se va iterando y acortando el nombre de dominio mientras no se encuentre una tupla para retornar. Se ha de destacar que el token ***'%zone%'*** siempre toma el valor que le corresponde en la iteración por la cual se encuentre el proceso de búsqueda, y si se alcanza el final de la cadena suministrada como nombre de dominio por el cliente del DNS, sin encontrar un registro para retornar, pues sencillamente se asume que lo almacenado en la base de datos no funge como autoritario para lo consultado por este cliente.

La quinta y última línea del esquema reserva el espacio para incluir el bloque de consulta que será utilizada por el método "***lookup()***" del driver. Este método es el encargado de retornarle una respuesta bien estructurada al cliente del servidor de DNS, por tales motivos el formato de salida de la misma ha de cumplir con ciertos requisitos, dentro de los cuales el más importante es el orden en que se retornan los campos para cada uno de los registros demandados. Un ejemplo de consulta para el método "***lookup()***" del driver podría ser:

{select ttl, type, mx_priority, data from dns_records where zone ='%zone%' and host='%record%'}

[4] ***Tupla***: Término de igual modo conocido como registro o fila en el marco de las base de datos relacionales.

Del ejemplo anterior nótese el orden de los campos ttl, type, mx_priority y data. Es de vital importancia recalcar al lector que este orden no debe ser violado puesto a que se podría acarrear errores graves que darían con el paro del servicio de DNS.

El método "***lookup()***" vuelve a hacer uso del símbolo ***'%zone%'*** anteriormente expuesto, y además incluye el nuevo token ***'%record%'***, el cual contendría entonces todo lo que fue recortado de la cadena del nombre dominio original para lograr encontrar una zona soportada dentro de la base de datos que nutre al servidor DNS. Es de vital importancia comunicar al lector que ambos token's han de estar presentes en la consulta de este método, en caso de no ser cumplida esta condición, sencillamente el proceso de carga del servidor es detenido y notificado el error por la salida estándar.

Por ejemplo, si la consulta siguiese siendo "***pc1.example.com***" demandando un registro de tipo "***A***" y el servidor posee una zona "***example.com***" con un registro de tipo "***A***" cuya entrada sería "***pc1***" y valor del campo "***data***" 10.7.20.2, pues como ya se había visto, en la primera iteración el método "***findzone()***" no encuentra la zona "***pc1.example.com***" y arremete a hacer un corte del dominio de menor nivel ("***pc1***") para dejar a el token ***'%zone%'*** con el valor de "***example.com***". Reformula la consulta y entonces sí encuentra una zona con este nombre; acto seguido pasa a ejecutar al método "***lookup()***" con la consulta especificada para el mismo y reemplazando al símbolo ***'%zone%'*** con "***example.com***" y al ***'%record%'*** con "***pc1***". Como existe una tupla en la base de datos que cumple con la cláusula "***where***" de esta consulta pues los campos que están asociados a ella son devueltos y el método "***lookup()***" con estos valores conforma una respuesta acorde con el estándar del BIND9 para el cliente que solicitó el servicio del DNS.

Se vuelve a enfatizar en que las consultas anteriormente vistas, para nada constituyen un estándar, los administradores de sistema pueden modificarlas del modo que mejor se ajuste a sus diseños de base de datos, pero siempre han de retornar los campos en el orden visto hasta el momento para el caso de la consulta "***lookup()***", y de proveer un modo de retornar al menos una fila en caso de encontrarse una zona a la cual se es autoritaria durante el proceso iterativo que sigue el driver DLZ para el método "***findzone()***".

La consulta que nutre al método "***lookup()***" y el mismo método en cuestión, son suficientes para dar un soporte básico desde un DBMS haciendo uso del DLZ. Por tal motivo, se considera como una práctica inteligente garantizar que todos los registros de la zona sean retornables por dicha consulta.

Existe un detalle interesante notado de seguro por los lectores más osados, y es que esta consulta muy bien puede retornar registros de distintos tipos. Por ejemplo, tómense los registros almacenados en la relación ***dns_records*** según la Tabla 2. Si la consulta del método "***lookup()***" posee en su cláusula ***where*** el uso del campo ***zona*** y el campo ***host*** de la relación, para la primera y segunda fila estos campos son iguales respectivamente. Surge la duda entonces sobre cuál registro retornar, si los de tipo (***type***) SOA o los de tipo NS. Pues bien, el driver en todo momento se comporta de modo inteligente ante esta situación y retorna solo los registros que coinciden con el tipo solicitado por el cliente del DNS, desechando entonces las tuplas con tipos distintos a la petición.

A modo de información se notifica que existen tres métodos más en el driver DLZ. Los mismos son ***authority()***, ***allnodes()*** y ***allowzonexfr()***, pero el uso de todos estos es opcional.

El método ***authority()*** será llamado únicamente si después del bloque de consulta para el método ***lookup()*** existe otro bloque de consultas (se asume que sea entonces el del método ***authority()***). La finalidad de este método es retornar los registro de tipo SOA y NS en caso de que el método ***lookup()*** no los contemple, por tal motivo, solo utiliza el símbolo ***'%zone%'*** y aunque la inclusión de otros token's no representa un motivo de error, los mismos no serán utilizados, serán ignorados. Un señalamiento extremadamente importante que ha de tener el lector en cuenta es que no se puede habilitar el uso del método ***authority()*** si el método ***lookup()*** provee los registros de tipo SOA y NS. Si se incumple en esta regla al momento de retornar algún registro de este tipo se ocasionará un error en el servicio del DNS conllevando al término del mismo.

Los métodos ***allnodes()*** y ***allowzonexfr()*** son siempre utilizados en conjunción, y sus finalidades son proveer un sistema de réplica entre las base de datos maestras y las esclavas, para contemplar la redundancia de datos entre los servidores de DNS. Pero el principio utilizado para este fin no se considera el más óptimo, dado que en realidad ejecutan una transferencia completa de zona (***allnodes()*** retorna todo los registros dada una zona), y lo mejor consiste en elaborar un sistema distribuido haciendo uso de las potencialidades de los DBMS y así simular una ***transferencia* incremental**[5] en lugar de una total.

Se le debe precisar al lector que al escribir una consulta para el método ***allnodes()*** siempre hay que ubicarla como siguiente a la consulta del método ***authority()***. En caso de que no se posea con una consulta para el método ***authority()*** y aún se desee usar el método ***allnodes()*** pues hay que

[5] ***Transferencia Incremental***: Solo se transfieren los cambios siguiendo un principio de actualización.

representar la consulta para ***authority()*** en su forma de vacío, o sea solo poniendo un bloque encerrado entre llaves pero sin contenido (***{}***). Note que para este caso no se puede poner siquiera un espacio entre las llaves, pues se consideraría el cuerpo de la consulta para ***authority()*** y obviamente se incurriría en un error.

2.4 Estrategia de Despliegue para los servidores del DNS

El despliegue de los servidores de DNS para el sistema TIP ha de contemplar todas las características de un sistema distribuido de base de datos que simule el árbol de DNS. La Fig.5.Interacción *entre el subsistema administrativo y el sistema distribuido de base de datos para el DNS.* muestra un esquema que ilustra el modo en que deberían de interactuar las aplicaciones administrativas del TIP y el sistema de base de datos distribuidas que respalda cada uno de los DNS subyacentes a ellas.

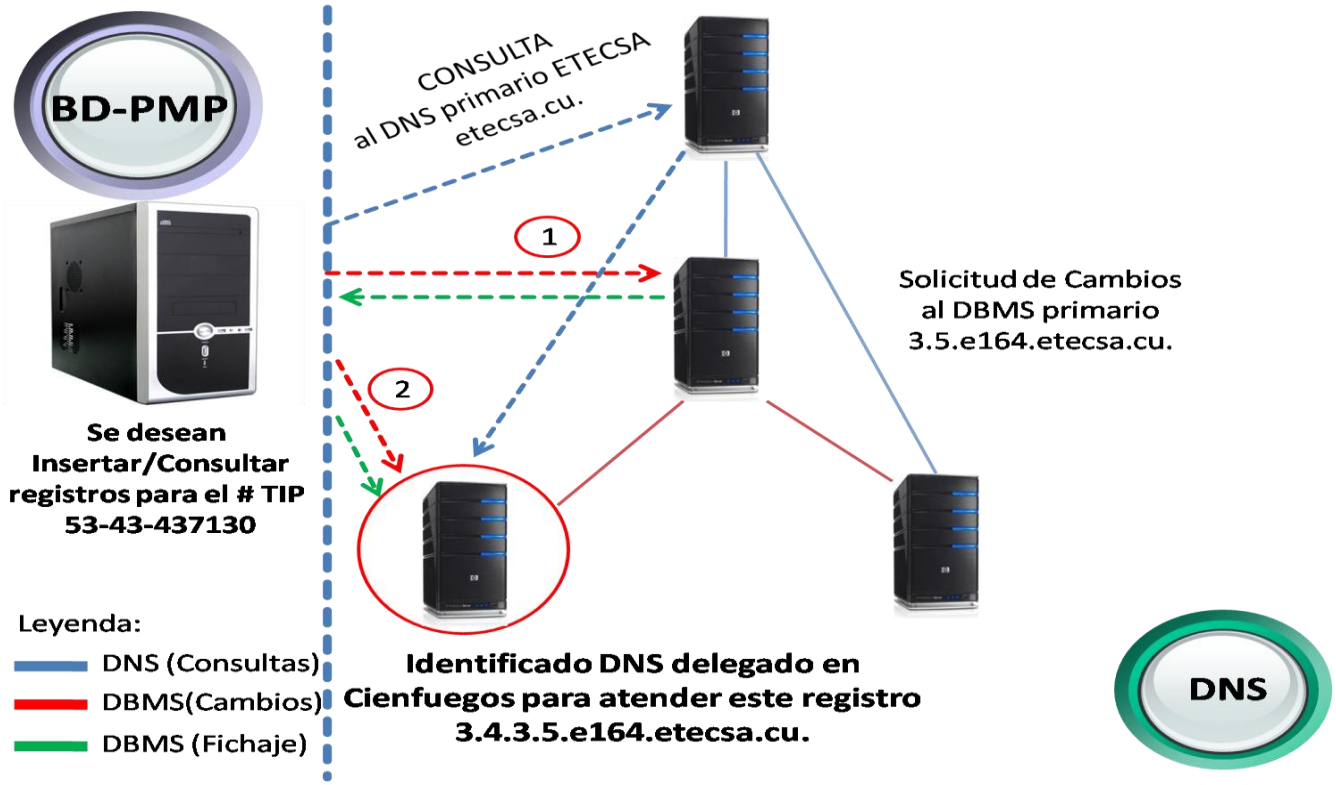

Fig.5.Interacción entre el subsistema administrativo y el sistema distribuido de base de datos para el DNS.

Abordando la figura, se pretende esclarecer con una descripción los elementos más significativos que en ella se exponen.

Primeramente se refiere como PMP a la Plataforma Manejadora de Peticiones. La misma está constituida por una serie de servicios webs encargados de la gran mayoría de las funcionalidades del sistema TIP, como la búsqueda de contactos dado un número TIP, proceso de subscripción, entre otros. La dirección de las flechas enmarca hacia dónde van dirigidas las transacciones, y el color de las mismas, como bien sale en la leyenda, distingue el tipo de transacción. Los servidores de DNS fueron representados como una sola estación y solo hacia aquellos que poseen una base de datos asociada puede llegar una transacción de tipo DBMS.

Se toma como punto de partida el momento en que un registrador inserta los datos de un nuevo subscriptor en la base de datos administrativa representada como BD-PMP en la figura. Los datos mientras no sean validados, permanecerán almacenados con un estado de "inactivos". Un subscriptor si desea puede seguir incluyendo nuevos contactos a su subscripción, pero nunca estos pasarán a formar parte del DNS hasta tanto no sean validados, independientemente de que la subscripción posea datos en estado "activo". Durante el proceso de subscripción se presupone que exista una asignación de número TIP para este subscriptor; este número será el empleado para conformar el nombre de dominio en el espacio que le corresponde en el DNS al sistema TIP.

El caso expuesto en la Figura 5 muestra las posibles operaciones que se pueden hacer con los contactos pertenecientes a la subscripción con número TIP **53-43-437130**. De validarse dichos contactos, la operación de inserción de los registros NAPTR's en el DNS comienza de manera automática haciendo una búsqueda en todo el árbol de dominio del DNS para el TIP, por un DBMS que respalde a un servidor de DNS autoritario para la zona a la que pertenece el nombre de dominio generado acorde con el número TIP de la subscripción, comenzando por la base de datos primaria que posee los registros de la zona **3.5.e164.etecsa.cu.** .

Esta búsqueda se efectúa de modo práctico debido a un comportamiento que poseen todas las base de datos de los DBMS que dan soporte a los servidores de DNS para el TIP, y es que al intentar la inserción en la base de datos primaria (saeta en rojo desde la BD-PMP hacia el DBMS **3.5.e164.etecsa.cu.**), una función se ejecuta para conocer si existe alguna base de datos delegada para la zona a la que pertenece el registro que se está intentando insertar, en caso de que no exista tal base de datos delegada, pues se asume que el DNS subyacente es autoritario para este nombre de dominio y por tanto el registro es insertado. Para el caso en que se encuentra una base de datos delegada, la primaria le retorna a la BD-PMP la información necesaria para intentar la inserción nuevamente en la delegada (saeta en verde desde el DBMS **3.5.e164.etecsa.cu.** hacia la BD-PMP).

En el ejemplo, como existe una base de datos delegada para la zona que atiende a todos los nombres de dominio relacionados con Cienfuegos (**3.4.3.5.e164.etecsa.cu.**), la primaria (**3.5.e164.etecsa.cu.**) no se hace cargo de los registros NAPTR's enviados, pero retorna la información necesaria para que BD-PMP intente insertar nuevamente pero en Cienfuegos. Se repite luego el procedimiento pero esta vez en la base de datos delegada (saeta en rojo desde la BD-PMP hacia el DBMS **3.4.3.5.e164.etecsa.cu.**), en caso de poderse insertar, la información de conexión suministrada para Cienfuegos será retenida en la BD-PMP (se ha encontrado entonces una base de datos que atiende la zona de Cienfuegos) y así queda fichada como la autoritaria a esa zona; de este modo, en lo sucesivo futuras inserciones, modificaciones o eliminaciones de registros relativas a la zona que ella soporta (**3.4.3.5.e164.etecsa.cu.**) serán ejecutadas directamente en la base de datos de Cienfuegos sin pasar por la primaria.

Hasta el momento se ha representado todo lo relativo con los DBMS que respaldan a los servidores DNS del TIP. Pero del lado de la otra tecnología implicada (los DNS's), existen además algunos detalles que por su simplicidad no han de ser obviados.

La PMP como bien se había dicho, posee la funcionalidad de directorio que dado un número TIP devuelve todos los contactos asociados con este, haciendo uso de las búsquedas realizadas en el DNS. Para ello solo se necesita tener en la estación servidora en la que se ejecuta, al menos a un servidor de DNS que pueda recorrer una porción del árbol de dominios hasta alcanzar al servidor que contenga los registros NAPTR's asociados con dicho número TIP. En la Figura 5 la BD-PMP posee como servidor de nombres al **etecsa.cu.** (saeta en azul desde BD-PMP hacia **etecsa.cu.**) y producto a que en este fueron dados de alta varios subdominios dentro de los cuales se cuenta con el **3.4.3.5.e164.etecsa.cu.** y el **3.5.e164.etecsa.cu.**, pues las consultas por los registros NAPTR's contenidos en cualquiera de ellos serán respondidas sin problemas. En un futuro no muy lejano la lógica de búsqueda que utilizan los servidores de DNS garantizará que incluso los clientes fuera del territorio nacional, puedan acceder a estos registros una vez que en la práctica se deleguen estos dominios del TIP.

2.4.1 Estrategia para garantizar la redundancia de datos

La réplica para garantizar la redundancia de datos sin lugar a dudas constituye una práctica común que asegura la disponibilidad de la información en caso de fallas del sistema sin importar los motivos de las recaídas del mismo. Cuando se trata el tema de réplicas en el contexto de los servidores DNS se tienen presente dos conceptos fundamentales: **Servidor Primario** y **Servidor Esclavo**.

Desde el punto de vista práctico, el **Servidor Primario** es aquel que posee de manera original los datos de la zona para la cual es autoritario; y el mismo ha de ser uno solo. En cambio el comportamiento de **Servidor Esclavo** puede para una misma zona ser aplicado a varios servidores de DNS, y serán esos servidores los que posean una copia de la información del **Servidor Primario**, la cual han de actualizar regularmente en pos de conservar la sincronización de los datos contenidos para las zonas bajo este enfoque, tomando únicamente como recurso lo que les sea cedido por el **Servidor Primario**.

En una zona que posea réplica para redundancia de datos, la inserción de nuevos registros, modificaciones y eliminaciones de los mismos, han de ser hechas en todo momento en el **Servidor Primario.** Para lograr la sincronización entre los distintos servidores, se ha implementado la siguiente estrategia.

Primeramente se intentan hacer de modo simultáneo las operaciones de inserción, modificación y eliminación de registros tanto en el **Servidor Primario** como en el(los) **Esclavo(s)**, haciendo uso de triggers que redirijan las operaciones realizadas en el **Primario** con éxito, hacia los restantes **Servidores Esclavos** implicados.

Pero esta estrategia dista mucho de ser óptima, pues no contempla una limitante que puede reflejarse, la misma es la indisponibilidad de algún(os) **Esclavo(s)** en el momento en que se efectúan cambios en el **Servidor Primario**. Esta limitante se ha abordado siguiendo el principio de transferencia incremental, o sea, se le transmitirá durante la copia a el(los) **Esclavo(s)** solo los cambios y no todo el contenido de la zona.

Para ello, existe en la base de datos del **Servidor Primario** una tabla en la cual se almacena por cada fila un registro que ha cambiado su estado, el tipo de cambios que ha sufrido y la dirección IP de un **Servidor Esclavo** al que se le desee comunicar el cambio. Para realizar la sincronización, entonces una función será la encargada de leer cada una de las filas de esa tabla y establecer el enlace con el **Servidor Esclavo** que se refiere en cada caso. Una vez establecido el enlace se pasa a reflejar en el **Esclavo** el cambio asociado al registro en dependencia de su tipo, que podrá ser inserción, actualización o eliminación del registro vinculado. Si la transacción se completa con éxito, pues la fila en cuestión en la tabla de cambios pendientes es eliminada.

La función que se encarga de la sincronización ha de ejecutarse de forma periódica, pero hasta la fecha no existe para el DBMS PostgreSQL un mecanismo que gestione esta funcionalidad. Por tal

motivo, se hará uso del demonio de tareas programadas del sistema operativo (***cron*** en el caso de GNU/Linux) para dar cumplimiento a esta funcionalidad.

En realidad ***cron*** lo que hará es llamar al cliente en línea de comandos del DBMS PostgreSQL. Este cliente se denomina ***psql*** y constituye una versátil herramienta que entre sus funcionalidades más interesantes posee la de ejecutar hacia una base de datos todas sentencias SQL contenidas en un fichero de texto y volcar el resultado de las mismas hacia la salida estándar.

Se detallan a continuación los pasos a seguir conjuntamente con algunos conceptos que ayudarán a comprender mejor la secuencia de los mismos.

Primero, se le hace saber al lector que la filosofía de uso del ***cron*** se basa en que cada usuario del sistema puede tener una tabla de planificación (realmente es un fichero de texto con formato predefinido) para programar sus tareas periódicas. La herramienta ***crontab*** es la que administra el contenido de estas tablas, y de modo específico con la opción –e se editan el contenido de la misma.

Los campos de todas las filas de las tablas para el ***crontab*** se desglosan de la siguiente manera:

Minutos Hora DíaMes Mes DíaSem Comando

Donde algunos de los valores admisibles para los mismos son:

Minutos: Desde el minuto 0 hasta el minuto 59 de una hora en específico.

Hora: Desde la hora 0 hasta la hora 23 según el formato militar.

DíaMes: Desde 1 hasta 31.

Mes: Desde 1 hasta 12.

DíaSem: Desde 0 hasta 6 partiendo de que 0 es el corresponde a Domingo, 1 a Lunes y así sucesivamente.

Comando: Comando a ejecutar. En este campo se hace una aclaración muy importante, y es que las órdenes de ejecución han de hacerse en todo momento haciendo uso de rutas absolutas y no relativas, incluso para el uso de binarios del sistema. Si usted desconoce la ruta absoluta de un binario de sistema, ejecute ***type nombre_binario*** y obtendrá la respuesta deseada.

Existen otros tipos de valores muy útiles para el trabajo con ***cron***, como por ejemplo el uso del * simbolizando todas las veces admisibles para el campo donde se encuentre (sería como decir *siempre* para el campo en específico), el uso de rangos (1-5 por ejemplo), el uso de valores específicos (3,5,10) para garantizar la ejecución propiamente en cada uno de esos valores y el uso del caracter **/** que especifica períodos en dependencia del campo.

Agregar una tarea a contemplar por el demonio ***cron*** es tan sencillo como añadir una línea en una tabla de planificación para cualquier usuario. El administrador del sistema podría añadir la siguiente:

```
*       */2     *       *       *       /usr/bin/psql -f /root/sentencias.sql -U enum -d dns
```

La cual indica que siempre cada 2 horas el programa ***psql*** ejecuta el contenido del fichero /root/sentencias.sql con los privilegios del usuario enum en la base de datos dns. La cadena /usr/bin/psql fue buscada haciendo uso de la herramienta ***type***. El contenido del fichero /root/sentencias.sql ha de contemplar como es lógico, la ejecución de la función de sincronización anteriormente referenciada.

2.5 Conclusiones

Hasta este punto de manera general se ha contemplado todo lo referente en cuanto a los motivos de elección, la instalación y configuración del driver DLZ integrado con PostgreSQL. De igual modo se provee una propuesta para contemplar en el despliegue de los servidores DNS en el país acorde con las demandas del sistema TIP. No se quiere cerrar el capítulo sin antes exponer la existencia de una gran variedad de aplicaciones clientes de DNS que permiten hacerle consultas a modo de prueba a los servidores, dentro de las cuales se destacan por venir incorporada en la mayoría de las distribuciones de GNU/Linux de modo nativo la herramienta ***host***, la utilidad ***nslookup*** (presente también en sistemas de la familia de Windows) y la muy conocida y completa herramienta ***dig***. Cada una de ellas posee manuales en línea de comandos disponibles con solo antecederles a su llamado por la consola el comando ***man***.

Capítulo 3: Pruebas y buenas prácticas de configuración

3.1 Introducción

En el siguiente capítulo se presenta el resultado de las pruebas realizadas a los servidores del DNS en un ambiente que recrea las posibles circunstancias futuras en las cuales se explotará el sistema TIP. De igual modo se le cede al lector algunas ideas de optimización y consejos que potencian en gran medida los tiempos de respuestas en los servidores de DNS que hacen uso del driver DLZ. También se darán a conocer algunas buenas prácticas en pos de garantizar la seguridad del servicio. Como bien ya se va haciendo costumbre, es necesario que el lector haya leído y comprendido en su totalidad el capítulo anterior para que se lleve una total comprensión del presente.

3.2 Metodología y entorno para la ejecución de pruebas

Como bien se había propuesto en una de las tareas de la presente investigación, era necesario idear un ambiente que permitiese comprobar de manera práctica lo expuesto en el capítulo anterior. Se le informa al lector que todo lo escrito referente a la instalación y configuración del BIND9 puede verse como una recopilación de las experiencias del autor durante el proceso de la puesta en marcha del servidor de DNS; en otras palabras, todo sobre lo que se ha escrito en este sentido funciona actualmente y fue sometido a diversas pruebas de rendimiento.

Las pruebas de rendimiento se centran de modo fundamental en la capacidad que posee un servidor de DNS para responder las consultas que han llegado a él. Estas pruebas como es de esperarse no comprenden entonces el retardo que se pueda generar dadas las particularidades de las distintas redes por donde pueda transitar una petición hacia el servidor de DNS, ya sea por características físicas de infraestructura u otras ajenas al medio informático. Otro indicador interesante a tener en cuenta es la velocidad de recarga o reinicio del servicio, pues como se ha de suponer, un servidor de DNS caído compromete fundamentalmente a todas las zonas a las cuales él se comporta de forma autoritaria y para el caso del sistema TIP esta situación constituiría un punto crítico dada la estrecha relación entre el servicio de nombres de dominio y funcionalidades como el directorio.

Se define a continuación los tres tipos de pruebas realizadas a los servidores de DNS que hacen uso del driver DLZ:

- ***Prueba de Latencia:*** Esta prueba se efectúa desde un cliente hacia el servidor, y su aplicación es ir generando una gran cantidad de consultas continuas a las cuales el servidor es autoritario, para saber del total de consultas enviadas cuántas fueron respondidas y el tiempo en que demoró en responder todo un ciclo de consultas. Siempre el DBMS aloja una cantidad igual o superior al número máximo de consultas realizadas en un ciclo de prueba.

- ***Prueba de Latencia Cargada:*** Esta prueba se efectúa con dos clientes, uno corriendo la misma aplicación generadora de consultas continuas que en la prueba de latencia y el otro cliente ejecutando otra aplicación que evalúa cuánto tiempo demora en responder una simple consulta. En aras de hacerla más factible se corre la aplicación que mide la latencia de una sola consulta 10 veces y se promedian las salidas de cada una de las ejecuciones.

- ***Prueba de inicialización del sistema:*** Esta prueba es la más sencilla de todas y solo mide el tiempo que se tarda un servidor de DNS en inicializarse.

Se hace la aclaración de que todas estas pruebas fueron hechas en servidores donde solo corrían los servicios de ***ssh*** (del inglés: Secure SHell), el de DNS y el DBMS PostgreSQL.

La velocidad de respuesta de un servidor de DNS se determina mediante un cálculo sencillo a partir de los resultados lanzados por las aplicaciones: se toma el número total de consultas realizadas sobre el servidor y este es dividido por el tiempo total que dura la prueba en cuestión (diferencia entre el momento de inicio y el momento de término). Su unidad de medida son las ***cps*** (Consultas Por Segundo) o ***qps*** (del inglés: Query Per Second).

Las herramientas utilizadas para realizar las pruebas fueron las siguientes:

- ***queryperf:*** Herramienta incluida en la sección "***contrib***" de las fuentes del BIND9, la cual recibe como parámetros un servidor del DNS como objetivo de prueba y un fichero de entrada el cual contiene toda la gama de consultas autoritarias para realizar a dicho servidor objetivo. Esta herramienta ha de ser compilada para poder acceder a su ejecución y será la utilizada para generar una gran carga de trabajo en las pruebas de ***Latencia*** y ***Latencia Cargada***.

- ***timeDnsLatency:*** Este script en lenguaje Perl es provisto por los desarrolladores del driver DLZ y simplemente mide en milisegundos el tiempo en que demora responder un servidor de DNS a una consulta en específico. Será la utilizada por el segundo cliente en la prueba de ***Latencia Cargada*** y en la ***Prueba de inicialización del Sistema***.

- ***dataSetGenerator:*** Este script en lenguaje Python es provisto por el autor de la presente investigación, y su utilidad se basa en que genera de modo secuencial diversos registros, tanto de tipo ***A*** como de tipo ***NAPTR*** con los cuales se nutren tanto el fichero de entrada del software ***queryperf*** como la base de datos que contiene los registros en el DNS de prueba.

Los ambientes de prueba de los servidores del DNS solo variaron en cuanto al hardware y la distribución del sistema operativo a utilizar, debido a que los experimentos fueron realizados lo mismo con las estaciones disponibles en el laboratorio de desarrollo de la universidad (los cuales emplean GNU/Linux Ubuntu) como en los servidores destinados para su puesta en marcha en el futuro en las edificaciones de los clientes (los cuales emplean CentOS). Por tal motivo se desglosa las prestaciones de cada estación de trabajo dado su rol, en dependencia del lugar de las pruebas:

Pruebas realizadas en el laboratorio de la Universidad:

- ***PC# 1 – Servidor de Prueba:*** Placa Base ASUS P5L2-VM con velocidad del microprocesador a 3.0 GHz, 512 MB de memoria RAM y una tarjeta de red a 100MB de tipo Ethernet.
- ***PC# 2 - Cliente de Prueba:*** Modelo SAMSUNG R60P, con microprocesador DualCore a una velocidad de 1866 MHz y 3 GB de memoria RAM con una tarjeta de red a 100MB de tipo Ethernet.
- ***PC# 3 - Cliente de Prueba:*** Placa Base ASUS P5L2-VM con velocidad del microprocesador a 3.0 GHz, 512 MB de memoria RAM y una tarjeta de red a 100MB de tipo Ethernet.

Pruebas realizadas en los servidores de ETECSA para el TIP:

- ***PC# 1 – Servidor de Prueba:*** Servidor con arquitectura Quad Core Xeon a velocidades de 2.4 GZ por cada núcleo (de un total de 16), con 4 GB de memoria RAM y tarjeta de red a 1GB de tipo Ethernet.
- ***PC# 2 - Cliente de Prueba:*** Servidor con arquitectura Quad Core Xeon a velocidades de 2.4 GZ por cada núcleo (de un total de 16), con 4 GB de memoria RAM y tarjeta de red a 1GB de tipo Ethernet.
- ***PC# 3 - Cliente de Prueba:*** Modelo SAMSUNG R60P, con microprocesador DualCore a una velocidad de 1866 MHz y 3 GB de memoria RAM con una tarjeta de red a 100MB de tipo Ethernet.

Se procede entonces a documentar los resultados para las pruebas realizadas en el laboratorio de desarrollo de la universidad, las cuales fueron las primeras en realizarse.

3.2.1 Pruebas con 1'000 registros y esquema complejo para la base de datos

El primer esquema de base de datos para las pruebas poseía un diseño cargado de tablas y era en general un tanto complejo, el cual defendía las buenas prácticas como normalización y eliminación de registros nulos y valores repetidos. En consecuencia el fichero de configuración del driver DLZ hacía tanto uso de consultas para el método ***authority()*** como consultas para el método ***lookup()***; y en ambas consultas se establecían relaciones entre tablas para poder retornar los datos necesarios y se hacían además uso de funciones propias del PostgreSQL. Por otra parte, la arquitectura de hardware de los servidores en cuestión solo permitía 2 conexiones abiertas hacia el DBMS. Los resultados como eran de esperarse fueron alarmantes. La Tabla 3 ilustra las principales salidas de una ***Prueba de Latencia***.

Indicador	Valores alcanzados (unidad)
Consultas enviadas	**1004 consultas**
Consultas completadas	**1004 consultas**
Tiempo máximo de respuesta	**1.687278 seg**
Tiempo mínimo de respuesta	**0.018893 seg**
Tiempo completo de la prueba	**23.347885 seg**
Velocidad de respuesta	**43.001754 cps**

Tabla 3.*Resultados para una prueba de Latencia con base de datos compleja.*

La ***Prueba de Latencia Cargada*** generó un promedio para los tiempos de respuestas de 46 milisegundos, lo cual no significaban resultados alarmantes y relativamente esperados ya que se contaba con dos conexiones abiertas hacia la base de datos.

La ***Prueba de inicialización del sistema*** retornó un resultado no superior a 1 segundo, lo que sin dudas será un comportamiento ideal para los servidores de DNS en caso de fallos.

Pero sin lugar a dudas el rendimiento del servidor del DNS era pésimo para una gama de consultas tan baja. Además, ese volumen de registros alojados en la base de datos no significa ni la milésima parte de lo que una zona del TIP debería contener, y obviamente se espera que a medida que los registros

aumenten en la base de datos, pues empeoren los tiempos en que demoran las respuestas del servidor de DNS.

Se determinó entonces, la necesidad de minimizar la base de datos en cuanto a la complejidad de su esquema, llegándose a tener solamente una tabla. De modo consecuente, el fichero de configuración del driver DLZ dejó de ser complejo; ya no era necesario las consultas para el método ***authority()*** pues todo podía ser retornado con el método ***lookup()*** y las consultas que atienden a este último no necesitaban relacionarse con ninguna tabla.

3.2.2 Pruebas con 1'000 registros y una base de datos minimizada

Hechas todas las operaciones de minimización a la base de datos, se procedió a repetir la misma prueba con el mismo volumen de registros, generándose los resultados siguientes:

Indicador	Valores alcanzados (unidad)
Consultas enviadas	**1004 consultas**
Consultas completadas	**1004 consultas**
Tiempo máximo de respuesta	**0.175771 seg**
Tiempo mínimo de respuesta	**0.030104 seg**
Tiempo completo de la prueba	**4.456982 seg**
Velocidad de respuesta	**225.264540 cps**

Tabla 4.*Resultados para una prueba de Latencia con base de datos minimizada.*

La prueba de ***Latencia Cargada*** dio como resultados un valor de 20 milisegundos.

Por otra parte la ***Prueba de inicialización del sistema*** siguió dando un resultado aproximado al segundo.

En conclusión se pudo observar como básicamente se aumentó en 5 veces la velocidad de respuesta del servidor de DNS. Entonces, tener en una sola tabla todos los registros, y sacrificar en cambio las buenas teorías de diseño de base de datos como la normalización, es una práctica que promueve de modo óptimo el rendimiento del servicio. De igual modo lo constituye la acción minimizar el volumen de las consultas alojadas en el fichero de configuración del driver DLZ y prescindir del método ***authority()***.

3.2.3 Pruebas con 100’000 registros. Empleo de los sockets de UNIX. Influencia de las potencialidades del hardware

En pos de observar de qué modo era afectada la velocidad de respuestas si se incrementa el volumen de datos en el DBMS, al mismo esquema asociado a la Tabla 4 se le preparó una ***Prueba de Latencia*** con un ciclo de 100’000 registros. Los resultados fueron los expuestos en la Tabla 5 para las intersecciones entre todas las filas y la columna ***Valores con localhost***.

Otra medida de optimización para potenciar la velocidad de respuestas del servidor es hacer uso de los sockets de Unix. Esta medida está condicionada a que el DBMS que aloje los registros tenga que residir en la misma estación que brinda el servicio de DNS. Pero antes se hace necesario hacer una breve explicación del concepto que se quiere aplicar para tal caso.

Los sockets de Unix pueden ser vistos como puntos de conexión que poseen algunos procesos, listos para establecer una comunicación bidireccional con otro proceso que trabaje con este mismo concepto. Para suerte de los usuarios del driver DLZ, el mismo puede trabajar con el socket de Unix que provee el DBMS PostgreSQL. Si en lugar de utilizar en la cadena de conexión hacia la base de datos ***localhost*** se emplea la dirección del socket de Unix (generalmente un camino hacia una carpeta en el sistema de ficheros), pues se incrementará la velocidad de las interoperaciones ya que no es necesario tramitar el envío y recepción de información entre el driver DLZ y la base de datos como si fueran paquetes TCP/IP (así es hecho si se utiliza ***localhost***) sino que se tramita a nivel de proceso.

Los resultados de la misma prueba con los 100’000 registros pero haciendo uso de los sockets de UNIX son expuestos en la Tabla 5 para las intersecciones entre todas las filas y la columna ***Valores con sockets***.

Posterior a este ensayo, durante un despliegue realizado en ETECSA a petición de los clientes, se le hicieron ***Pruebas de Latencia*** a los servidores que sí fungirán en un futuro como los auténticos servidores del sistema TIP. El mismo esquema de base de datos empleado para generar los resultados de la Tabla 4 fue el desplegado en dichos servidores. Pero esta vez la base de datos fue poblada con un vasto volumen de 19'400'013 registros. Sin lugar a dudas ha sido la prueba que más registros contiene en su base de datos. La Tabla 5 demuestra cómo influye de modo determinante distintas prestaciones de hardware sobre un mismo esquema de base de datos. Se ha de destacar que la arquitectura de hardware de estos servidores permitió establecer en el fichero de configuración del driver DLZ, 4 conexiones abiertas hacia la base de datos. Los resultados de esta prueba son los expuestos para las intersecciones entre todas las filas y la columna ***Valores en ETECSA***.

Indicador	Valores con localhost	Valores con sockets	Valores en ETECSA
Consultas enviadas	**100004 consultas**	**100004 consultas**	**100004 consultas**
Consultas completadas	**100004 consultas**	**100004 consultas**	**100004 consultas**
Tiempo máximo de respuesta	**0.572623 seg**	**0.134679 seg**	**0.057691 seg**
Tiempo mínimo de respuesta	**0.007880 seg**	**0.007131 seg**	**0.000744 seg**
Tiempo completo de la prueba	**387.007385 seg**	**253.307981 seg**	**27.027160 seg**
Velocidad de respuesta	**258.403338 cps**	**394.792140 cps**	**3699.981796 cps**

Tabla 5.*Comparativa entre el uso de localhost, el uso de sockets y diferencia de hardware.*

Que los servidores de ETECSA respondan con la velocidad reflejada en la tabla anterior, constituye definitivamente una gran diferencia desde la primera prueba realizada en el laboratorio de desarrollo. Por otra parte, ***La Prueba de Latencia Cargada*** arrojó valores de 6 milisegundos y la de ***Inicialización de sistema*** igualmente mostró casi valores instantáneos, manteniéndose en el orden de los milisegundos.

3.3 Buenas prácticas de configuración orientadas a la seguridad

Hasta el momento se ha hecho énfasis en las optimizaciones para potenciar el rendimiento de los servidores de DNS, pero estas en su totalidad no han comprendido el tema de la seguridad. Nuevamente se le advierte al lector que la siguiente investigación no contempla exhaustivamente el cómo se debe administrar un servidor de DNS, pero se considera extremadamente necesario retocar algunos detalles en cuanto a este tema.

Primeramente, sería seguro que el ordenador donde se encuentra el servidor de DNS no posea ejecutándose otros servicios ajenos a los vinculados en la resolución de nombres de dominios. Para este caso, solo el demonio ***named*** y el DBMS PostgreSQL serían suficientes. De esta forma se minimiza la posibilidad de que un atacante explote vulnerabilidades presentes en otras aplicaciones que se encuentren corriendo en el servidor.

Otro punto a contemplar son los privilegios del usuario que se conecta a la base de datos mediante el driver DLZ. Aunque hacer llegar una inyección SQL a través del driver constituye una opción muy difícil, mitigar los riesgos en este sentido no procura complicaciones, basta con revocar los privilegios

administrativos y de modificación sobre los objetos de la base de datos al usuario empleado en la cadena de conexión del driver.

Si se utilizan los sockets de UNIX se puede regular y controlar el acceso al mismo modificando sus atributos de lectura, escritura y ejecución. Recuerde siempre que el sistema de ficheros constituye el espacio de ejecución de un socket, por tal motivo, pueden ser tratados como ficheros.

Ejecutar el servicio de DNS con un usuario sin privilegios comprometedores en el sistema de igual modo resulta inteligente. El demonio ***named*** requiere ser llamado por el usuario ***root*** para llevar a cabo algunas rutinas que demandan ejecución de máximo privilegio, como por ejemplo la creación de sockets de escucha en los puertos privilegiados. Posteriormente si no se le indica un comportamiento arbitrario, se mantiene corriendo con los privilegios de ejecución del usuario ***root***. Pero los sistemas GNU/Linux actuales pueden hacer uso de novedosos mecanismos del kernel para ceder todos los privilegios de ***root*** a otro usuario distinto, de esta forma no existe el riesgo de que se pueda comprometer al sistema tras la explotación de una vulnerabilidad. Primeramente cree un nuevo usuario (cuyo identificador puede ser bind) y especifique que no posee intérprete de línea de comandos (verifique en el fichero /etc/passwd que la cadena sucesiva a los últimos dos puntos (***:***) sea /bin/false para este usuario).

Una de las tantas rutinas que lleva a cabo el proceso de inicialización es la creación del fichero que contiene el identificador del proceso del demonio ***named***. Por defecto, se intenta en todo momento efectuar esta operación sobre el directorio /var/run pero solo el usuario ***root*** es el propietario de este directorio. En los sistemas basados en GNU/Linux es muy desaconsejable cambiar cualquiera de los atributos para las jerarquías del sistema de ficheros que atienden a la configuración del sistema operativo. Por tal motivo, hacer un cambio de propietario sobre el directorio /var/run, o cambiar sus atributos no es una solución para que el usuario bind pueda acceder al fichero que contiene el identificador del proceso del demonio ***named***. Pero el demonio ***named***, puede escribir este fichero en otra dirección opcional a la cual le sean concedidos privilegios de acceso para el usuario bind.

De crearse el directorio /etc/bind/run y de modo posterior asignándole los atributos de lectura, ejecución y escritura accesibles al usuario bind, se podría considerar que el fichero del identificador de proceso del ***named***, sea alojado dentro del mismo.

Para lograr este comportamiento arbitrario en el demonio ***named***, es preciso hacer un cambio en las opciones de ejecución del mismo. El fichero de configuración ***named.conf*** posee una cláusula

enfocada a cambiar dichas opciones, la misma se denomina ***options***, y dentro de ella, la sentencia ***pid-file*** recibe como argumento una cadena entrecomillada con la ruta de la ubicación del fichero que contiene el identificador de procesos del demonio.

El lector ha de prestar mucha atención si ya la cláusula ***options*** no está declarada, en caso de estarla, solo tendrá que añadir la sentencia correspondiente. Todo quedaría como sigue:

options {

….

//Otras especificaciones de opciones en caso de que existan

….

pid-file "/etc/bind/run/named.pid";

}

Después en el script de inicialización automática del servicio de DNS tratado en el apartado 2.3.1, se incluye para las opciones de ejecución del demonio la opción ***–u bind***.

Emplear la última versión de BIND9 siempre será una buena práctica, dado que a menudo se hacen mejoras al código del mismo tomando como retroalimentación los criterios de la comunidad.

En otro sentido, existen muchos sitios de seguridad en Internet donde se publican las vulnerabilidades que han logrado ser explotadas en algunas aplicaciones servidoras según sus versiones. Una vez encontradas las mismas, los desarrolladores intentan sacar cuanto antes una solución para arreglarlas, pero mientras, el servicio pudiese estar comprometido a atacantes que accedan a la información de la versión del servidor en cuestión y conozcan sus posibles debilidades. Por tal motivo, constituye una buena práctica de administración ocultar esta información, para ello en la cláusula ***options*** del fichero de configuración del demonio ***named*** agregue la sentencia ***version none;***, esto hará que no se devuelva nada cuando se consulte, o de igual modo se puede poner una cadena la cual será retornada cuando se pregunte por la versión del servidor, por ejemplo ***version "Dato no suministrado";*** (nótese en ambos casos la terminación de la sentencia con un punto y coma(***;***)). Se ha de comprobar el correcto funcionamiento de esta opción con cualquiera de las órdenes de comandos siguientes:

nslookup -q=txt -class=CHAOS version.bind. {ip del servidor}

dig version.bind chaos txt @{ip del servidor}

3.4 Conclusiones

De seguro ha constituido de gran interés para los administradores de sistemas conocer los tiempos de respuesta que brinda un servidor de DNS que emplea el driver DLZ. Registrar y comparar resultados sin lugar a dudas representa una práctica frecuente y que fomenta la retroalimentación. Siguiendo esta metodología, se han puesto a prueba una gran variedad de esquemas para la base de datos y configuraciones del driver DLZ, arrojando como principal conclusión que un diseño complejo del esquema de la base de datos actúa negativamente sobre su rendimiento.

De igual modo quedó demostrado cómo influyen de manera crucial las potencialidades de hardware con las que se cuente para el establecimiento del servicio. En cuanto a las medidas de seguridad propuestas en este capítulo, se ha de recalcar que llevar a cabo las mismas no garantiza un nivel de seguridad elevado, pero si el requerido para hacer a los servidores funcionales y seguros.

Conclusiones

El sistema TIP de forma general, se encuentra aún en desarrollo, y su correcta consolidación depende del término de algunos de sus módulos básicos. Dichos módulos en todo momento recurren al DNS para efectuar la búsqueda de información que ellos manipulan, por tal motivo se precisó que el sistema de nombres de dominio se comportara de la manera más ininterrumpida posible.

Después de un análisis sobre los disímiles modos de propiciar el servicio del DNS, se propone con la siguiente investigación, el empleo de un enfoque que lo capacita de un vasto dinamismo.

Por otra parte, sin lugar a dudas ha resultado mucho más familiar a los programadores, la interacción con base de datos relacionales que el empleo de librerías de desarrollo provistas por terceros, las cuales demandan conocimientos sobre conceptos propios y requieren dominio sobre terminologías relacionadas con la configuración y administración del DNS.

De modo general se espera que la aplicación de este enfoque agilice y minimice los costes en el desarrollo de los módulos implicados con el DNS.

Los tiempos de respuesta de los servidores de DNS que se acoplan a las especificaciones de la propuesta que defiende este trabajo, han sido bien aceptadas por los especialistas de la parte cliente, los cuales lo han calificado de óptimos para el entorno de su futuro despliegue.

Recomendaciones

No se quiere dar fin a esta investigación sin dejar de hacer algunas recomendaciones, dentro de las cuales se destacan las siguientes:

- Elaborar una aplicación web que permita una interacción más directa entre los administradores del servidor y el mismo, ha de ser una meta sin elevados costes de desarrollos ahora que la información se encuentra alojada en una base de datos.

- Durante los períodos de pruebas, los administradores de sistema para detectar errores y fallas pueden hacer uso de la llamada del demonio ***named*** con las opciones ***–d[número] –f –g***, las cuales establecen de modo respectivo las funcionalidades siguientes: un nivel descriptivo de las trazas operacionales del servicio (donde número define cuán preciso se ha de ser), que el servicio no se ejecute como demonio, y que todas las notificaciones sean impresas por la salida estándar.

- Para incrementar la seguridad del servidor de DNS se ha de investigar sobre cómo implementar la ejecución del servicio bajo una jaula en el sistema de ficheros. La opción ***–t*** del demonio ***named*** provee este mecanismo de ejecución.

Bibliografía consultada

Aitchison, Ronald. Pro DNS and BIND. New York, Apress, 2005.

Albitz, Paul y Liu, Cricket. DNS and BIND 5th Edition. Sebastopol, OReilly,2006.

Pérez, Marcos. Impacto del ENUM en las redes y los servicios, Revista de Telecomunicaciones, 2008. [Disponible en: http://www.ahciet.net/actualidad/revista/r.aspx?ids=10739&ids2=21829].

Telephone Number Mapping (enum), 2009. [Disponible en: http://www.ietf.org/html.charters/enum-charter.html].

International Telecommunication Union - ENUM, 2009. [Disponible en: http://www.itu.int/osg/spu/enum/].

Faltstrom, Patrik y Mealling, Michael. The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM), 2004. [Disponible en: http://www.ietf.org/rfc/rfc3761.txt].

Bradner, Scott, Conroy, Lawrence, Fujiwara, Kazunori. draft-ietf-enum-3761bis-04, 2009. [Disponible en: http://www.ietf.org/internet-drafts/draft-ietf-enum-3761bis-04.txt].

BIND DLZ; Phase 2 Project Plan, 2004. [Disponible en: http://www.nlnet.nl/project/bind-dlz/phase2/200210-plan.html#topofpage].

BIND DLZ Home, 2004. [Disponible en: http://bind-dlz.sourceforge.net/].

Using the BIND 9 Simplified Database Interface, 2002. [Disponible en: http://uw713doc.sco.com/en/NET_tcpip/dns.bind9sdi.html].

Glosario de términos

A continuación, en orden alfabético, se muestra el significado de algunos términos usados en este documento que pueden dificultar la compresión del mismo:

Base de datos administrativa: Contiene todos los datos para administrar el sistema TIP, como contraseñas, privilegios de cada usuario en dependencia de los roles, información personal de los subscriptores, entre otros. En el documento se hace referencia a un subconjunto de su información como base de datos directorio.

Base de datos de contactos: Se refiere a la base de datos que respalda a un DNS, debido a que los contacto donde realmente se almacenan es en la misma.

Base de datos directorio: Se refiere a un subconjunto lógico de la base de datos administrativa. A nivel de software ambas están desplegadas en el mismo DBMS.

Compilador: Programa destinado a generar a partir de un código fuente, ficheros que contienen código objeto.

Contactos: En el marco de esta investigación se entiende por contactos todas aquellas formas por las cuales se puede notificar a una persona, como por ejemplo, su correo electrónico, su número telefónico, en fin, todos los servicios que una persona posea para ser contactado.

Demonio: es un tipo especial de proceso informático que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario (es un proceso no interactivo).

Enlazadores: Programa destinado a enlazar los distintos ficheros con código objetos para conformar un binario ejecutable para un sistema.

Exploits: Código malicioso que aprovecha un error de programación y al ejecutarse provoca un malfuncionamiento intencionado de la aplicación víctima, otorgando privilegios a usuarios no autorizados en un sistema.

Kernel: Se refiere al núcleo del sistema operativo. Es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.

libtool: Herramienta de programación GNU proveniente del Sistema de construcción para GNU usada para crear librerías de software portables.

Nombre de Dominio: Cadena de caracteres que se utiliza para expresar que las computadoras pertenecen a una determinada organización. Los nombres de dominio son jerárquicos por naturaleza, y cada nivel de la jerarquía se separa de los otros niveles mediante un punto.

POSIX: Acrónimo de la expresión en inglés Portable Operating System Interface. Son en sí una familia de estándares de llamadas al sistema operativo definidos por el Instituto de Ingenieros Eléctricos y Electrónicos que persiguen generalizar las interfaces de dichos sistemas para que una misma aplicación pueda ejecutarse en distintas plataformas.

RAM: Se refiere a la memoria de acceso aleatorio de un ordenador. es la memoria desde donde el procesador recibe las instrucciones y guarda los resultados.

Registrador: Es un rol dentro del sistema TIP. El mismo recoge la información suministrada por los subscriptores durante el proceso de subscripción de un usuario.

RFC: Acrónimo de la expresión en inglés Request For Comments. Son en sí una recopilación de trabajos y estudios a través de los cuales se proponen y efectúan cambios en Internet, en general con orientación técnica.

root: Administrador, usuario de más alto privilegio en los sistemas GNU/Linux.

Script: En programación, se refiere a un conjunto de instrucciones estructuradas en un fichero para ser ejecutadas por un intérprete del lenguaje vinculado.

Triggers: Un trigger (o disparador) en una base de datos, es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción (INSERT), actualización (UPDATE) o borrado (DELETE).