Universidad de las Ciencias Informáticas

Facultad 1

Título: Propuesta de vías para reducir la aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias de la Informática.

Autor:

Ricard Miguel Barzaga Herrera.

Tutores:

Ing. Abel Alfonso Fírvida Donésteves.

Ing. Michel Evaristo Febles Parker.

La Habana, julio de 2015

“Año 57 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaración de Autoría

Declaro ser el autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo con carácter exclusivo.

Para que así conste firmo la presente declaración jurada de autoría en Ciudad de La Habana a los _____ días del mes de _____ del año _____ .

Ricard Miguel Barzaga Herrera

_____

Firma del autor

Ing. Abel A. Fírvida Donéstevez

_____

Firma del Tutor

Ing. Michel E. Febles Parker

_____

Firma de Tutor

"...a los héroes no hay que recordarlos solo después de la muerte, hay que apoyarlos y seguirlos cuando viven y cuando están junto al pueblo luchando".

Fidel Castro Ruz.

Agradecimientos

A mis tutores, Abel y Michel, pues sin ellos no estaría hoy escribiendo estas líneas.

A mi mamá Doralis, por darme la vida y hacer todo lo que estaba en sus manos, y hasta lo que no por que llegara hasta donde estoy hoy.

A mami Elena, papi Miguel, tío Miguelito, tía tita, tía Rosita, lilí, Dani, Eliannis, tío Alexander, tata, Leyanis y toda la familia, por preocuparse en todo momento por mí.

A tía mima que hace ya mucho tiempo no se encuentra entre nosotros, pero que hubiera estado muy feliz por lo que he logrado en la vida.

A mi abuelo Tescastro, Anay, tía Aliatna y tío Aliet, por aceptarme como parte de su familia a pesar de que no existía siquiera la seguridad de que así fuera.

A Magdalena, Pedroso, Victoria, Fela y Biksmar por guiarme y ser más que simples vecinos del barrio.

A mis compañeros de aula, el antiguo grupo 1109. A todos los que ya no están con el grupo, Luis, Juan Enrique, Javiel, Dairon, Diwel, Yunior, y los que lograron llegar hasta el final, Marlon, Yoandri, Daniel, Neyvis, Anaylis, Ivis, Yanet, Dayrol, Yisel, Manuel, Roger. A los que se unieron después y conformaron el actual 1508, Mayvis, Annareya, Ariagna, Yileni, Annia, Angel. A todos gracias por haber compartido junto conmigo estos años.

A los profesores que me han marcado en el transcurso de la carrera: Gabriel, Mónica, Zumeta, Eiger, Ilmaris, Osiris, Daylianis, Gendry y Saura.

A todos los profesores del proyecto que me brindaron su ayuda: Dayrelis, Alexis, Héctor, Jesús, Juan, Luis Daniel.

A los colegas Marbier, el vice, Enmanuel, Junquera.

Y por último pero no menos importante, a mis amigos, Yudelyn, Yaciel Molina y Yordanis. Gracias por nunca olvidarse de mí.

Dedicatoria

A mi mami Doralis, mami Elena, papi Miguel, tío Miguelito, tía tita, tía Rosita, lilí, Dani, Eliannis, tío Alexander, tata, Leyanis y toda la familia.

A mi abuelo Tescastro, Anay, Aliatna y Aliet.

A Magdalena y Pedroso.

Resumen

Con el presente trabajo de diploma titulado “Propuesta de vías para reducir la aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova”, se pretende contribuir al proceso de desarrollo de la distribución cubana de GNU/Linux Nova a través de la propuesta de algunas vías para reducir la probabilidad de aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova. En el mismo se defiende la idea de que si se disminuye la cantidad de bibliotecas y aplicaciones innecesarias para el correcto funcionamiento del sistema, será posible reducir la cantidad de vulnerabilidades en el mismo. En el documento, se recogen los resultados obtenidos a partir del estudio del estado del arte de algunos de los procedimientos que deben seguirse para implementar un plan de riesgo contra vulnerabilidades en un negocio, enfocándose en el proceso de desarrollo de *software*, del estudio de la aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova. Como resultado de la investigación se obtuvieron vías que permitirán reducir la probabilidad de aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova.

Palabras clave: Análisis, distribución, sistema, vulnerabilidad.

Índice

ÍNDICE

Índice de tablas

Introducción.

Las distribuciones GNU/Linux están ganando relevancia en todo el mundo ya que algunas administraciones públicas han promovido leyes para adoptar el *software* libre y de código abierto (FOSS)[1] como base tecnológica para sus desarrollos endógenos[2] .Unos de los argumentos que se presentan en todas estas leyes son la flexibilidad y la seguridad que brinda el *software* libre debido a la capacidad de buscar errores de programación o vulnerabilidades en el código fuente (1) (2) (3) (4).

En junio de 2013 el contratista de la Agencia de Seguridad Nacional (NSA)[3] norteamericana, Edward Snowden[4] hizo públicos a través de los periódicos *The Guardian* y *The Washington Post* documentos clasificados como alto secreto sobre varios programas de esta organización, incluyendo los programas de vigilancia masiva PRISM[5] y XKeyscore[6]. Snowden dio un vuelco a la opinión pública mundial con la revelación de que Estados Unidos de América (EUA) tenía la capacidad de espiar el contenido de gran cantidad de usuarios de Internet (5). Esto implicaba la posible existencia de puertas traseras en aplicaciones, incluidas las FOSS.

A partir de estas revelaciones, personal relacionado con las tecnologías de la informática y las comunicaciones se dio a la tarea de encontrar algunos de estos agujeros de seguridad y así fueron

1 FOSS: Terminología utilizada para nombrar el *software* libre y de código abierto, por *free*/libre *and open sourc*e *software*, en inglés.

2 Modelo de desarrollo que busca potenciar las capacidades de una región o comunidad local; de modo que puedan ser utilizadas para fortalecer la sociedad y su economía de adentro hacia afuera, para que sea sustentable y sostenible en el tiempo.

3 *National Security Agency* por sus siglas en inglés. Agencia de inteligencia criptológica del Gobierno de los Estados Unidos, administrada por el Departamento de Defensa.

4 Edward Joseph Snowden es un consultor tecnológico estadounidense, informante y antiguo empleado de la Agencia Central de Inteligencia (CIA) y de la Agencia de Seguridad Nacional (NSA).

5 PRISM es un programa de vigilancia electrónica considerado confidencial a cargo de la NSA de los Estados Unidos desde el 2007.

6 Xkeycore es un sistema informático secreto utilizado por la NSA e Estados Unidos para la búsqueda y análisis de datos en Internet.

descubiertas: *Heartbleed* que permite a cualquier persona en la red acceder a información sensible, como llaves de cifrado y *CCS Injection* que permite a un atacante acceder a la información que se transmite entre un cliente y un servidor.

La distribución cubana de GNU/Linux Nova no está exenta a la aparición de vulnerabilidades, y es preocupante el hecho de que estos puedan aparecer en cualquiera de las 1400 bibliotecas y aplicaciones de terceros que tiene el sistema instaladas.

Partiendo de esta situación, surge el **problema científico:** ¿Cómo incrementar la seguridad en las aplicaciones informáticas que componen la distribución cubana de GNU/Linux Nova?

Para darle solución a este problema se ha establecido como **objeto de estudio:** el proceso de detección de vulnerabilidades en una distribución GNU/Linux.

Estableciendo como **objetivo general:** Proponer vías para disminuir la probabilidad de aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova y como **campo de acción:** la distribución cubana de GNU/Linux Nova.

Para darle cumplimiento al objetivo general planteado, se definieron los siguientes **objetivos específicos:**

1. Analizar la frecuencia de aparición de vulnerabilidades en las aplicaciones y bibliotecas que componen la distribución cubana de GNU/Linux Nova.
2. Proponer vías para reducir la probabilidad de aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova.
3. Probar que es posible reducir la probabilidad de aparición de vulnerabilidades en una distribución GNU/Linux durante su proceso de desarrollo.

Con el presente trabajo de diploma se **defiende la idea** de que si se disminuye la cantidad de bibliotecas y aplicaciones innecesarias para el funcionamiento del sistema, será posible reducir la cantidad de vulnerabilidades en el mismo.

En proceso de investigación se utilizaron los siguientes **métodos teóricos**:

Histórico-Lógico: utilizado para realizar un estudio detallado sobre la existencia de vulnerabilidades en las aplicaciones y bibliotecas que componen la distribución cubana de GNU/Linux Nova y como se ha comportado la tasa de crecimiento de las mismas en el transcurso del tiempo.

Analítico-Sintético: mediante este método se descompuso el problema en pequeñas partes y se realizó un mejor análisis del comportamiento de las vulnerabilidades existentes en las aplicaciones y bibliotecas que componen la distribución cubana de GNU/Linux Nova. Este análisis permitió proponer vías para reducir la probabilidad de aparición de vulnerabilidades en un sistema.

El presente documento está conformado por introducción, capítulos 1, 2, 3, conclusiones, recomendaciones, bibliografía, anexos y glosario de términos.

En el **Capítulo 1** se realiza un estudio del comportamiento de las vulnerabilidades en las aplicaciones y bibliotecas que componen la distribución cubana de GNU/Linux Nova. También se realiza un estudio sobre las acciones que pudieran tomarse en el desarrollo de la distribución cubana de GNU/Linux Nova para reducir la aparición de vulnerabilidades en la misma.

En el **Capítulo 2** se propone una nueva arquitectura para la distribución cubana de GNU/Linux Nova, la cual permite reducir al mínimo las aplicaciones y bibliotecas que posee actualmente la distribución cubana de GNU/Linux Nova instaladas por defecto.

En el **Capítulo 3** se realiza una reducción de bibliotecas innecesarias para el correcto funcionamiento de la distribución cubana de GNU/Linux Nova. También se demuestra que es posible reducir vulnerabilidades en una aplicación reduciendo la cantidad de líneas innecesarias de código que la misma posee. Finalmente se demuestra que es posible reducir la cantidad de vulnerabilidades en la distribución cubana de GNU/Linux Nova reduciendo la cantidad de aplicaciones y bibliotecas en la misma al mínimo necesario para mantener la misma cantidad de funcionalidades en el sistema y para permitir el correcto funcionamiento del mismo.

Capítulo 1: Fundamentación Teórica.

En el presente capítulo se hace un profundo estudio sobre las principales formas utilizadas para incrementar la seguridad de un sistema informático, entre ellas, la que está relacionada directamente con el desarrollo de *software*. Se realiza un análisis sobre las técnicas de análisis estático y dinámico de código fuente. También se hace alusión al por qué los analizadores estáticos y dinámicos de código fuente no resuelven completamente el problema de la aparición de vulnerabilidades en un sistema. Se brinda un significado para *software* con propensión a errores. Se propone una fórmula que permite calcular la probabilidad de existencia de vulnerabilidades en una distribución GNU/Linux. Además, se deduce que es posible reducir la probabilidad de aparición de vulnerabilidades en una distribución GNU/Linux reduciendo la cantidad de *software* con propensión a errores que posee la misma y que no sean necesarios para su correcto funcionamiento.

1.1. Conceptos asociados al desarrollo de la investigación.

A continuación se presentan algunos conceptos asociados al objeto de estudio que son necesarios para comprender la presente investigación.

Vulnerabilidad: en seguridad informática, la palabra vulnerabilidad, hace referencia a una debilidad en un sistema que permite a un atacante violar la confidencialidad, integridad, disponibilidad, control de acceso y consistencia del sistema o de sus datos y aplicaciones (6). Las vulnerabilidades, son el resultado de errores o fallos en el diseño de los sistemas. Aunque en un sentido más amplio, también pueden ser el resultado de las propias limitaciones tecnológicas porque en principio, no existen sistemas cien por ciento seguros. Por lo tanto, existes vulnerabilidades teóricas y vulnerabilidades reales conocidas como *exploits* (6).

Las vulnerabilidades en las aplicaciones suelen corregirse con parches, *hotfixs*[7], o con cambios de versión. En tanto que algunas otras requieren un cambio físico en un sistema informático. Las vulnerabilidades se descubren muy seguidos en grandes sistemas, y el hecho de que se publiquen

7 Paquete que puede incluir varios archivos y que sirve para resolver un error específico dentro de una aplicación informática.

rápidamente por todo Internet (mucho antes de que exista una solución para el problema), es motivo de debate. Mientras más conocida se haga una vulnerabilidad, más probabilidades de que existan piratas informáticos que quieran aprovecharse de ellas (6).

Algunas vulnerabilidades típicas pueden ser:

- Desbordamiento de pilas y otros *buffers*.
- Errores de validación de entradas como: inyección SQL, errores en el formato de cadenas, etcétera.
- Secuestro de sesiones.
- Ejecución de código remoto.

Software Comunitario: se denomina *software* comunitario al *software* desarrollado por comunidades de usuarios en la red, el mismo, rompe con la dependencia ideológica del *software* libre, y su principal diferencia con este es su definición política (7).

Sistema Operativo: en informática, se denomina sistema operativo al conjunto de programas informáticos, que permiten una satisfactoria administración de los recursos que ostenta una computadora (8). También conocido como *software* de sistema, el sistema operativo comienza a funcionar en la computadora inmediatamente después de encenderla y gestiona el *hardware* desde los niveles más básicos permitiendo además la interacción con el usuario (8). Este tipo de sistema, se puede encontrar en la mayoría de los equipos electrónicos que emplean microprocesadores, tal es el caso de un teléfono celular, o de un reproductor de DVD (8).

En el sistema operativo se cumplen cinco funciones básicas:

1. Administración de recursos (esta es la función que le permite al usuario la dirección del *hardware*, incluyéndose tanto los periféricos como la red en caso de existir) (8).
2. Suministro de interfaz a los usuarios (a partir de esta el usuario podrá cargar programas, acceder a los archivos y realizar otras tareas en la computadora) (8).
3. Administración de archivos (permite crear, modificar y hasta eliminar los archivos) (8).
4. Servicio de soporte y de utilidades (permite actualizar las versiones, incorporar nuevas y más

utilidades, mejorar la seguridad del sistema en función de las necesidades, controlar los nuevos periféricos que ingresan y también la corrección de errores que se suscitan en alguno de los *software*) (8).

5. Administración de tareas (facilita la administración de todas las tareas informáticas que lleva a cabo el usuario) (8).

Actualmente existen varios sistemas operativos, entre los más populares se destacan: *Windows*, *Mac OS*, *Linux*, *AmigaOS, Unix,* y en teléfonos celulares se destacan: *Blackberry OS*, *Windows Phone, WebOS, Bada, Android y Symbian* (8).

Distribución GNU/Linux: una distribución Linux es una distribución de *software* basada en el núcleo Linux que incluye determinados paquetes de *software* para satisfacer las necesidades de un grupo específico de usuarios, dando así origen a ediciones domésticas, empresariales y para servidores (9).

1.2. Vías para incrementar la seguridad en un sistema informático.

Según dicta el Instituto SANS (*SysAdmin Audit, Networking and Security Institute*[8]) en el documento "*The critical security controls for efective cyber defense*", los 20 controles de seguridad (CSC) que toda empresa debe establecer e implementar con el objetivo de administrar riesgos de seguridad de la información en marco de sus riesgos empresariales generales son los siguientes:

CSC 1: Inventario de dispositivos autorizados y no autorizados.

CSC 2: Inventario de *software* autorizado y no autorizado.

CSC 3: Configuraciones seguras para *hardware* y *software* en los dispositivos móviles, ordenadores portátiles, estaciones de trabajo y servidores.

CSC 4: Evaluación y remedio continuo de vulnerabilidades.

CSC 5: Defensas de *malware*[9].

8 Institución con ánimo de lucro fundada en 1989, con sede en Bethesa (Maryland, Estados Unidos) que agrupa a 165000 profesionales de la seguridad informática. Sus principales objetivos son: reunir información sobre o referente a seguridad informática y ofrecer capacitación y certificación en el ámbito de la seguridad informática.

9 Término general que se le da a todo aquel *software* que perjudica a la computadora.

CSC 6: Aplicación de *software* de seguridad.

CSC 7: Control de acceso inalámbrico.

CSC 8: Capacidad de recuperación de datos.

CSC 9: Habilidades de seguridad de evaluación y entrenamiento adecuado para llenar los vacíos.

CSC 10: Configuraciones seguras para los dispositivos de red, tales como *firewalls*, *routers* y *switches.*

CSC 11: Limitación y control de puertos de red, protocolos y servicios.

CSC 12: El uso controlado de privilegios administrativos.

CSC 13: Límites de defensa.

CSC 14: Mantenimiento, monitoreo y análisis de registros de auditoría.

CSC 15: Control de acceso basado en la necesidad de conocer.

CSC 16: Monitoreo y control de cuenta.

CSC 17: Protección de datos.

CSC 18: Respuesta y gestión de incidentes.

CSC 19: Ingeniería segura de redes.

CSC 20: Pruebas de penetración y ejercicios de equipo rojo.

De ellos, el que se considera que está enfocado al proceso de desarrollo de un sistema operativo es el CSC 4, el cual describe entre otros aspectos:

1. Ejecutar herramientas automatizadas contra todos los sistemas en la red sobre una base semanal o más frecuentemente y entregar listas priorizadas de las vulnerabilidades más críticas a cada administrador del sistema responsable junto con la puntuación de riesgo que compara la eficacia de los administradores de sistemas y departamentos en la reducción de riesgos (10).
2. Correlacionar los registros de eventos con información de análisis para cumplir dos objetivos. En primer lugar, verificar que la actividad de las herramientas habituales de análisis de vulnerabilidades sí se registra. En segundo lugar, el personal debe ser capaz de correlacionar eventos de detección de ataques con anteriores resultados de análisis de vulnerabilidades para determinar si el *exploit* dado fue utilizado contra un objetivo, sabiendo si es vulnerable (10).

3. Realizar análisis de vulnerabilidades en modo autenticado ya sea con agentes que se ejecutan localmente en cada sistema para analizar la configuración de seguridad o con escáneres remotos que se dan derechos administrativos en el sistema que está siendo probado (10).
4. Suscribirse a los servicios de inteligencia de vulnerabilidades para mantenerse al tanto de las vulnerabilidades emergentes, y utilizar la información obtenida para actualizar las actividades de análisis de vulnerabilidades de la organización, por lo menos una vez al mes (10).
5. Implementar herramientas automatizadas de gestión de parches y herramientas de actualización de *software* para el sistema operativo y el *software* en todos los sistemas para los cuales están disponibles dichas herramientas (10).
6. Monitoreo de los registros asociados con cualquier actividad de exploración y las cuentas de administradores para asegurar que toda actividad de exploración y de acceso a través de la cuenta con privilegios se limita a los plazos legítimos de exploración (10).
7. Comprobar los resultados del escaneo *back-to-back* de vulnerabilidades para verificar que las vulnerabilidades hayan sido tratadas, ya sea mediante parches, la implementación de un control de compensación, o de la documentación y la aceptación de un riesgo de negocio razonable (10).
8. Mida la demora al parchear nuevas vulnerabilidades y compruebe que el retraso es igual o inferior a los valores de referencia establecidos por la organización (10).
9. Evaluar los parches críticos en un entorno de prueba antes de llevarlos a la producción en los sistemas empresariales (10).
10. Establecer un proceso riesgo-tasa de vulnerabilidades en función de la explotabilidad y el impacto potencial de la vulnerabilidad, y segmentada por grupos apropiados de activos (por ejemplo: servidores del tipo Zona Desmilitarizada (DMZ), servidores de red internos, ordenadores de escritorio, ordenadores portátiles) (10).

De los diez aspectos mencionados anteriormente y que describe el CSC 4; los números 4, 5, 8 y 10 son los que, si fuesen implementados en el proceso de desarrollo de la distribución cubana de GNU/Linux Nova, podrían disminuir considerablemente los riesgos críticos en el sistema operativo a la hora de liberarlo para su uso posterior. Estos aspectos, se encuentran enfocados en el proceso de detección,

evaluación y corrección de vulnerabilidades en un negocio determinado, el cual en el caso puntual del presente trabajo de diploma, está representado por el proceso de desarrollo de un sistema operativo.

1.3 Análisis estático y/o dinámico de código.

1.3.1. Análisis estático.

El análisis estático de *software* es el proceso de evaluar el *software* sin ejecutarlo. Es por lo tanto, una técnica que se aplica directamente sobre el código fuente tal cual, sin transformaciones previas ni cambios de ningún tipo. La idea es que en base a ese código fuente, podamos obtener información que nos permita mejorar la base del código manteniendo la semántica original (11).

El análisis de código estático es el proceso de detección de errores y defectos en el código fuente de un *software*. El análisis estático puede ser visto como un proceso automatizado de revisión de código (12). Es una evaluación del código generado para buscar defectos con la particularidad de que se realiza sin necesidad de ejecutar ese código (13). Las técnicas estáticas son las que se utilizan para garantizar la calidad antes de la ejecución: evaluando el diseño y el análisis del código (bien manualmente, o bien utilizando herramientas como un verificador de tipos) (14).

Por lo tanto se puede definir que el análisis estático no es más que una técnica de análisis que permite detectar vulnerabilidades en el código fuente de un *software* mediante la evaluación del diseño y el análisis del código sin que el mismo sea ejecutado.

Los tipos más comunes de herramientas de análisis estático para encontrar vulnerabilidades son conocidos como: analizadores de debilidades de código fuente, analizadores de seguridad de código fuente, herramientas estáticas para la prueba de seguridad de aplicaciones (SAST), o herramientas de análisis de vulnerabilidades. Un analizador de debilidades en el código fuente busca vulnerabilidades usando varios tipos de patrones (por ejemplo, pueden hacerlo realizando un seguimiento de los datos de fuentes no confiables para ver si son enviados a las operaciones potencialmente peligrosas) (15).

Una cuestión fundamental es que la mayoría de las herramientas no garantizan como encontrar todas las vulnerabilidades, ni siquiera garantizan encontrar vulnerabilidades de cualquier clase en particular (15). Dado que:

1. Los lenguajes de programación no están diseñados para que sean fáciles de analizar.
2. La mayoría del *software* no está escrito para que sea fácil de analizar por los analizadores estáticos de código fuente.

Como resultado, las herramientas de análisis estático pueden necesitar una gran cantidad de ayuda humana para ser aplicadas. Una razón parcial y más profunda es que los lenguajes de programación C, C++, y *Objective-C* son notoriamente difíciles de analizar de forma estática; pues los constructores como punteros (y especialmente los punteros de función) pueden ser difíciles de manejar de forma estática (15). Esto no quiere decir que los analizadores estáticos sean inútiles. Los analizadores estáticos pueden examinar la forma en que el *software* se comportará bajo un gran número de posibles entradas (en comparación con el análisis dinámico), y la heurística de la herramienta, puede limitar el número de falsos positivos. Pero la heurística utilizada por los analizadores estáticos a veces resulta en el fracaso a la hora de detectar vulnerabilidades importantes (15).

1.3.2. Pruebas con analizadores estáticos a la biblioteca OpenSSL vulnerable a los errores conocidos como *Heartbleed* y *CCS Injection*.

En el 2014 fue detectado un importante *zero-day exploit* (ZDE) en una de los FOOS SSL más utilizadas, la biblioteca OpenSSL. Se puede definir un ZDE como cualquier vulnerabilidad que posee un sistema informático que es atacada el mismo día que es descubierta y antes de que sea corregida por la compañía productora de dicho sistema (16). Es conocida de esta forma, porque existen cero días desde que la vulnerabilidad es descubierta y el primer ataque. El peligro en este tipo de vulnerabilidades es que no son detectadas por la compañía que desarrolla las aplicaciones que pueden contener un ZDE, característica que es explotada por el atacante a la hora de realizar el ataque (16). El error detectado, conocido como *Heartbleed*, le permite a cualquier persona en Internet, leer la memoria de los sistemas protegidos por las versiones vulnerables del *software* OpenSSL. Dicha vulnerabilidad compromete claves secretas usadas para identificar los proveedores de servicios y para cifrar el tráfico, los nombres y contraseñas de los usuarios, lo que permite a los atacantes espiar las comunicaciones, robar datos directamente de los

servicios tales como: aplicaciones web, mensajería instantánea, Redes Privadas Virtuales (VPN's)[10] y de los usuarios, además de permitirle suplantar servicios y a los propios usuarios. Esto afecta las últimas versiones de las principales distribuciones GNU/Linux (tabla 1) y ha sido demostrado que se introdujo en el código desde hace 2 años.

Tabla 1: Distribuciones GNU/Linux con OpenSSL potencialmente vulnerable.	
Distribuciones Afectadas.	**Versión de OpenSSL.**
Debian Wheezy (estable)	1.0.1e-2+deb7u4
Ubuntu 12.04.4 LTS	1.0.1-4ubuntu5.11
CentOS 6.5	1.0.1e-15
Fedora 18	1.0.1e-4
OpenSUSE 12.2	1.0.1c
Nova 4.0	1.0.1-4ubuntu5.12

Tabla 1: Distribuciones GNU/Linux con OpenSSL potencialmente vulnerable según el sitio web "Heeartbleed bug".

Pero fue aún más preocupante la detección de un error, conocido como *CCS Injection*, en la misma biblioteca a finales de este año. Aun cuando esta vulnerabilidad fue menos perjudicial para los sistemas informáticos, el principal problema radica en que ha estado presente en el código por 16 años sin ser notado por ninguna herramienta de inspección de *software*, desarrollador, empaquetador o colaborador.

Al parecer las herramientas automáticas para el análisis del código estático son una forma de reducir la aparición de vulnerabilidades en las aplicaciones, ya que hay algunas herramientas como: *Cppcheck*, *Flawfinder*, *Berkeley Lazy Abstraction Software verification Tool* (BLAST) y *Clang*, que pueden comprobar código C/C++ y detectar errores, excepciones de seguridad, fugas de memoria, uso no válido de *Standard Template Library* (STL)[11], variables sin inicializar y funciones sin usar. En el Centro de Soluciones Libres (CESOL), perteneciente a la Facultad 1 de la Universidad de las Ciencias Informáticas (UCI), se desarrolló una herramienta llamada Auditoría de Código Fuente (ACF), liberada para la distribución cubana de

10 Tecnología de red que permite una extensión segura de la red local (LAN) sobre una red pública o no controlada como Internet.

11 Biblioteca de *software* para el lenguaje de programación C++ que ha influido en varias partes de la biblioteca estándar de C++.

GNU/Linux Nova, que es usada para inspeccionar códigos fuente de lenguajes como: C/C++, *Python*, *Bash* y *Perl*. Estas herramientas podrían contribuir al incremento de la seguridad de las distribuciones GNU/Linux, combinándolas o insertándolas en el proceso de desarrollo del sistema. Sin embargo, al analizar códigos defectuosos con algunas de estas herramientas como ACF, *Clang*, *Flawfinder* y *Cppcheck* se produjeron gran cantidad de falsos positivos.

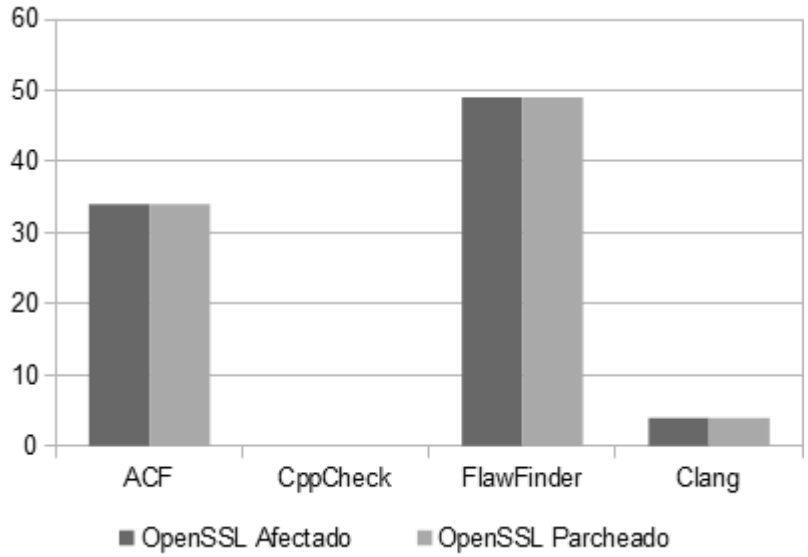

Ilustración 1: Análisis de la biblioteca OpenSSL afectada por la vulnerabilidad Heartbleed, y con dicha vulnerabilidad parcheada.

Como muestra la ilustración 1, al analizar una versión de la biblioteca OpenSSL afectada por los errores conocidos como *Heartbleed* y *CCS Injection* con estas herramientas, ninguna de ellas pudo encontrar los errores antes mencionados.

Por lo anteriormente planteado, se llega a la conclusión de que los analizadores estáticos de código fuente no resuelven completamente el problema de la detección de vulnerabilidades en un *software*.

1.3.3. Análisis dinámico.

Las técnicas dinámicas de análisis de código fuente, suponen la ejecución de un programa y la observación de su rendimiento (14). El análisis dinámico, implica la ejecución de un programa con

entradas específicas, tratando de encontrar vulnerabilidades (15). El análisis dinámico, se basa en el análisis de las propiedades de un programa mientras está corriendo (17). Por tanto, el análisis dinámico constituye una forma de análisis del código fuente de un programa. Se basa en el estudio de su comportamiento en tiempo de ejecución, a partir de las entradas de flujo de datos específicos que le son proporcionados.

Una limitación de los enfoques dinámicos, es que es imposible probar completamente cualquier programa en horarios humanos pertinentes. Por ejemplo, un programa trivial, que añade dos enteros de 64 bits tiene 2128 entradas posibles. Probando todas las entradas (suponiendo un procesador de 4GHz y 5 ciclos para probar cada entrada) requeriría 13.5 trillones de años (1,35x1022 años). Incluso la computación relativamente paralela no ayuda. Los programas reales, por supuesto, tienen entradas mucho más complejas que esta. Por lo tanto, los enfoques dinámicos, no pueden demostrar que un programa es seguro en un sentido fuerte; todo lo que se puede demostrar es la ausencia de vulnerabilidades con las pruebas que fueron utilizadas (15). Sin embargo, a pesar de sus limitaciones, los enfoques dinámicos, son capaces de detectar un mayor número de vulnerabilidades en el código fuente analizado. Además, constituyen una vía útil para mejorar la seguridad de un sistema, siempre y cuando se conozcan sus limitaciones (15).

1.3.4 Pruebas con *Valgrind* a sistemas vulnerables a *Heartbleed*.

Valgrind es un sistema con Licencia Pública General (GPL)[12] de depuración y perfilado para los programas de Linux. Con la *suite* de herramientas de *Valgrind* es posible detectar muchos errores de gestión de memoria, evitando horas frustrantes de caza de errores y permitiendo hacer programas más estables. También puede realizar perfiles detallados para ayudar a acelerar los programas que analiza (18). Las principales herramientas incluidas en la *suite Valgrind* son las siguientes (19):

- *Memcheck*: detecta problemas de manejo de memoria.

12 Licencia Pública General de GNU o más conocida por su nombre en inglés *GNU General Public License* es la licencia más ampliamente usada en el mundo del *software* y garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el *software.*

- *Cachegrind*: realiza un perfil del caché utilizado, realizando una simulación detallada de los caché L1, D1 y L2.
- *Helgrind*: encuentra *data-races* en programas multihilo.

Al realizar pruebas en la biblioteca OpenSSL con *Valgrind* no fue detectado el exceso de lectura en la memoria (*buffer over-read*). Se puede pensar, que al ejecutar *Valgrind*, es posible detectar un exceso de lectura en la memoria, sin embargo, no es así (20).

En primer lugar, la asignación de memoria de OpenSSL derrotó a todos los sistemas de prueba de asignación de memoria como *Valgrind*. Al analizar la asignación de memoria realizada por OpenSSL, *Valgrind* vería unos valores muy extraños, pues toda la asignación y desasignación se encuentra dentro de OpenSSL (20). En segundo lugar, era necesaria una manera de activar la detección de este tipo de error en el momento exacto de su ocurrencia, sin embargo, no existían pruebas de la existencia de este error, por lo que *Valgrind* no podía haberlo detectado (20).

Sin embargo, según Cridtopher T. Celi, al correr la versión de OpenSSL 1.0.1e, la cual se sabe, es vulnerable al error *Heartbleed*, *Valgrind* detectó una "lectura no válida" de una región de memoria que había sido asignada con *malloc*. La lectura se produjo dentro de la función *memcpy*. Esta prueba en particular, envió un mensaje que fue conocido por desencadenar el *Heartbleed*. Señala así que *Heartbleed* pudo haber sido detectado antes de que fuera explotado (15).

De lo anteriormente planteado se concluye que la utilización de analizadores dinámicos de código fuente no resuelve completamente el problema de detección de vulnerabilidades en un sistema.

1.4. ¿Por qué los analizadores estáticos y/o dinámicos no pueden detectar algunas vulnerabilidades?

Los productos tradicionales, tales como los analizadores estáticos y/o dinámicos de código fuente, son eficaces generalmente en la detección de problemas conocidos que puedan causar vulnerabilidades. Sin embargo, no pueden mantenerse al día con el rápido aumento del volumen de programas maliciosos y de programas con mayor cantidad de vulnerabilidades desconocidas (12). El hecho de suponer que todas las vulnerabilidades tendrán el mismo patrón de ejecución es un error, ya que un analizador estático y/o

dinámico de código fuente solo será capaz de detectar dichas vulnerabilidades una vez que estén activas y hayan sido ejecutadas (21).

Dado que un ZDE, es una vulnerabilidad que es detectada y atacada el mismo día, no es posible que esta sea detectada por una herramienta de análisis estático de código fuente, pues este desconoce la forma en que se genera dicha vulnerabilidad en el código de la aplicación que está analizando. Por las razones planteadas anteriormente y dado que ACF es una herramienta de análisis de código estático, se puede llegar a la conclusión de que ACF tampoco es capaz de detectar este tipo de vulnerabilidades.

1.5. Probabilidad de vulnerabilidades en distribuciones de FOSS.

Estudiando la aparición de vulnerabilidades en las aplicaciones que componen la distribución cubana de GNU/Linux Nova se puede regularizar que existen cuatro tipos de comportamiento en estas:

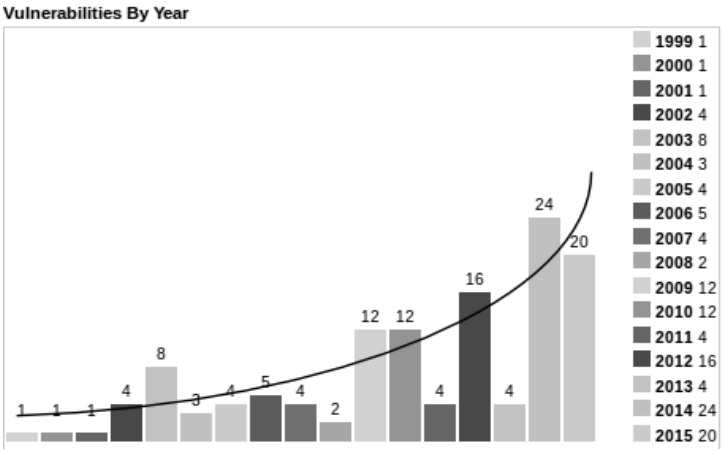

Ilustración 2: Comportamiento ascendente en la aparición de vulnerabilidades por año en la biblioteca OpenSSL de desarrollo comunitario.

La ilustración 2 muestra como se ha comportado la aparición de vulnerabilidades a través de los años en la biblioteca OpenSSL desarrollada por las comunidades de usuarios en la red. La misma muestra un comportamiento ascendente ya que, exceptuando algunos casos, posee una tendencia ascendente en el transcurso de los años (22).

La ilustración 3, muestra el comportamiento de la aparición de vulnerabilidades en el Kernel de Linux desarrollado por comunidades de usuarios en la red. La aparición de vulnerabilidades en dicha aplicación posee un comportamiento estable en el transcurso del tiempo (23).

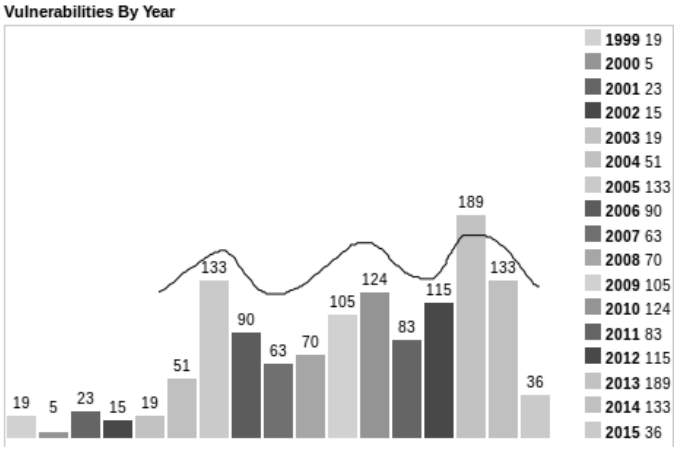

Ilustración 3: Comportamiento estable en la aparición de vulnerabilidades por año en el Kernel de Linux.

La ilustración 4 muestra cómo se ha comportado la aparición de vulnerabilidades en la biblioteca OpenSSL mantenida por *RedHat*. Esta biblioteca, a pesar de permitir el desarrollo comunitario, también es financiada por un proyecto empresarial, lo que ha provocado que hayan aparecido en la misma pocas vulnerabilidades en el transcurso de los años. Además de que dichas vulnerabilidades han ido en descenso (24).

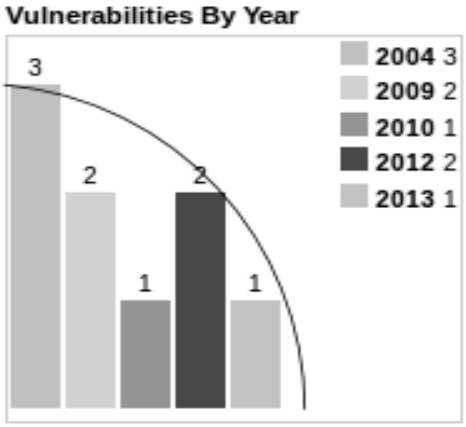

Ilustración 4: Comportamiento descendiente en la aparición de vulnerabilidades por año en la biblioteca OpenSSL RedHat empresarial.

La ilustración 5, presenta en comportamiento de la aparición de vulnerabilidades en la aplicación *Busybox*, financiada por el proyecto *Busybox*. Esta aplicación ha mantenido un comportamiento despreciable en cuanto al promedio de vulnerabilidades por año, manteniendo solo una vulnerabilidad en toda la vida del proyecto (25).

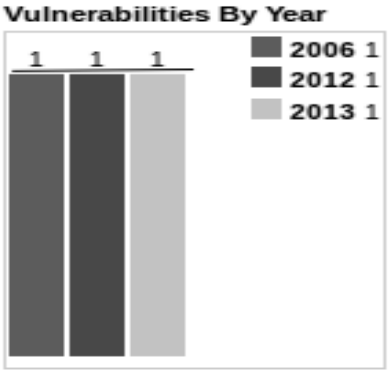

Ilustración 5: Comportamiento despreciable en la aparición de vulnerabilidades por año en la aplicación Busybox.

Luego de realizar un estudio que estuvo basado en las aplicaciones y bibliotecas que posee la distribución cubana de GNU/Linux Nova 4.0 con el objetivo de caracterizar la tasa de aparición de vulnerabilidades en

la misma, se obtuvo la información que se muestra en la ilustración 6. Dicha ilustración, constituye un gráfico de pastel que contiene el por ciento de aplicaciones y bibliotecas que poseen una tasa de aparición de vulnerabilidades ascendente, sostenido, descendente y despreciable. Del estudio se obtuvieron que la distribución cubana de GNU/Linux Nova cuenta actualmente con un 32 por ciento del *software* que posee una tasa sostenida o ascendente en la aparición de vulnerabilidades por año, y un 68 por ciento del *software* posee una tasa de aparición de vulnerabilidades decreciente o que puede despreciarse. De este estudio se puede concluir que más de un cuarto de las aplicaciones que componen la distribución cubana de GNU/Linux Nova posee una tasa ascendente o sostenida en la aparición de vulnerabilidades en el transcurso del tiempo.

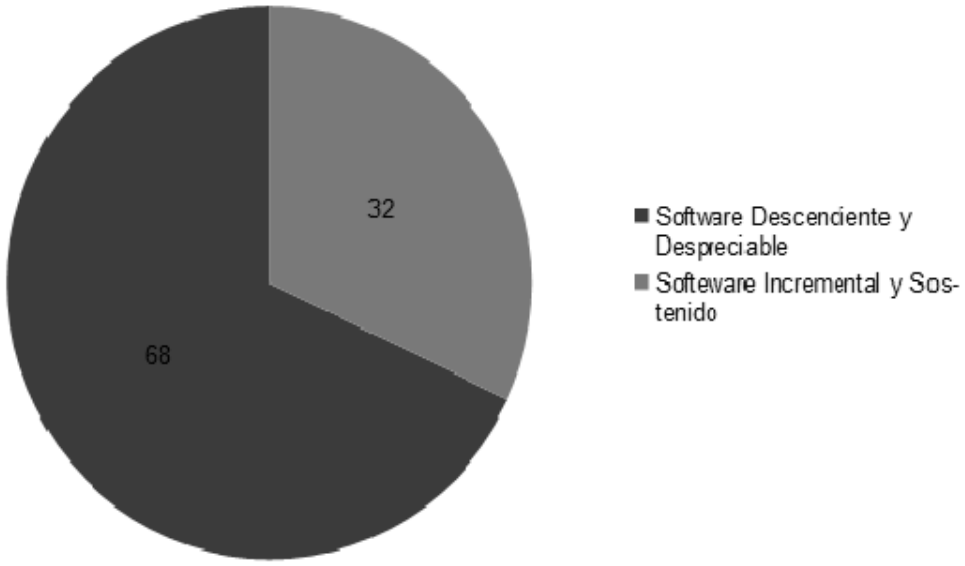

Ilustración 6: Por ciento de aplicaciones y bibliotecas que componen la distribución cubana de GNU/Linux Nova en las cuales la aparición de vulnerabilidades se comporta de manera ascendente, estable, decreciente o despreciable.

De este estudio se define que el *software* con propensión a errores (SPE) es aquel que tiene un ritmo ascendente o sostenido en la aparición de vulnerabilidades a través del tiempo. Al profundizar en los procesos de desarrollo de estos se puede observar que en su mayoría dependen de desarrollos

comunitarios cuya financiación está sostenida por donaciones, no así las que tienen un índice de vulnerabilidades decreciente o despreciable que tienden a ser financiadas por empresas y aunque tienen un componente comunitario en su desarrollo los cambios esenciales son conducidos por un equipo pagado.

Atendiendo a que una distribución de GNU/Linux, como cualquier otro *software*, está diseñada para cumplir con funcionalidades específicas y a estas responden directamente la inclusión de alguna aplicación o biblioteca, se puede definir a la probabilidad de aparición de vulnerabilidades en una distribución GNU/Linux como:

$$P_{Vul} = X / N \quad .$$

Donde X es el número de SPE y N es el número de funcionalidades que posee el sistema. Con la condición de que $X \leq N$, dado que cada biblioteca o aplicación brinda al menos una funcionalidad.

Por lo anteriormente planteado se puede definir que una distribución GNU/Linux posee una cantidad de funcionalidades N mayor o igual a la cantidad de bibliotecas y aplicaciones que posea el mismo instaladas.

De esto se deduce que al reducir el SPE se puede reducir la probabilidad de aparición de vulnerabilidades conservando las mismas funcionalidades.

La práctica en el desarrollo de proyectos basadas en código abierto como *Android* y *Chrome OS* han demostrado que otra vía para lograr esta meta es reducir la utilización de *software* de terceros, esto se sustenta en la teoría de que es treinta veces más costoso arreglar errores en *software* terminado que hacerlo en las etapas de diseño e implementación (26).

Conclusiones Parciales.

En el presente capítulo se pudo comprobar que las herramientas de análisis de código (estáticas y/o dinámicas) no resuelven completamente el problema de la reducción de vulnerabilidades en el código fuente de la distribución cubana de GNU/Linux Nova. Se planteó una fórmula para estimar la probabilidad

de aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova, arribándose a la conclusión de que un factor clave puede ser la reducción de la cantidad de SPE en la construcción de la distribución cubana de GNU/Linux Nova manteniendo la misma cantidad de funcionalidades que posee el sistema.

Capítulo 2: Propuesta de solución para la reducción de vulnerabilidades en la distribución cubana de GNU/Linux Nova.

Según Jorge Luis Machín Castillo (Anexo 1), Nova tiene 1400 aplicaciones en su sistema base (27). Sin embargo, existen distribuciones *de código abierto* que demuestran que esto no es totalmente necesario como *Android*, y que presenta una considerable reducción de las bibliotecas en su sistema base. Por esta razón, se establece que una de las formas alternativas de reducir la aparición de vulnerabilidades, es la reducción de las bibliotecas y aplicaciones que no afecten el correcto funcionamiento de la distribución cubana de GNU/Linux Nova, incluyendo entre ellas las que son consideradas SPE. Debido a esto, en el presente capítulo se presenta una propuesta de arquitectura para la distribución cubana de GNU/Linux Nova, la cual se ha modelado con el objetivo de reducir al mínimo la cantidad de bibliotecas y aplicaciones que posee el sistema por defecto.

2.1. Propuesta de arquitectura para la distribución cubana de GNU/Linux Nova.

En la ilustración 7, se presenta la organización entre las aplicaciones y dependencias que posee la distribución cubana de GNU/Linux Nova en la actualidad. La cual, sigue el patrón de dependencias que posee la distribución de GNU/Linux Ubuntu, en la cual está basada la distribución cubana de GNU/Linux Nova. Sin embargo es necesario realizar cambios en esta estructura con el objetivo de reducir las aplicaciones y dependencias que posee la distribución cubana de GNU/Linux Nova, para así lograr reducir la cantidad de errores y la posibilidad de aparición de vulnerabilidades en la misma.

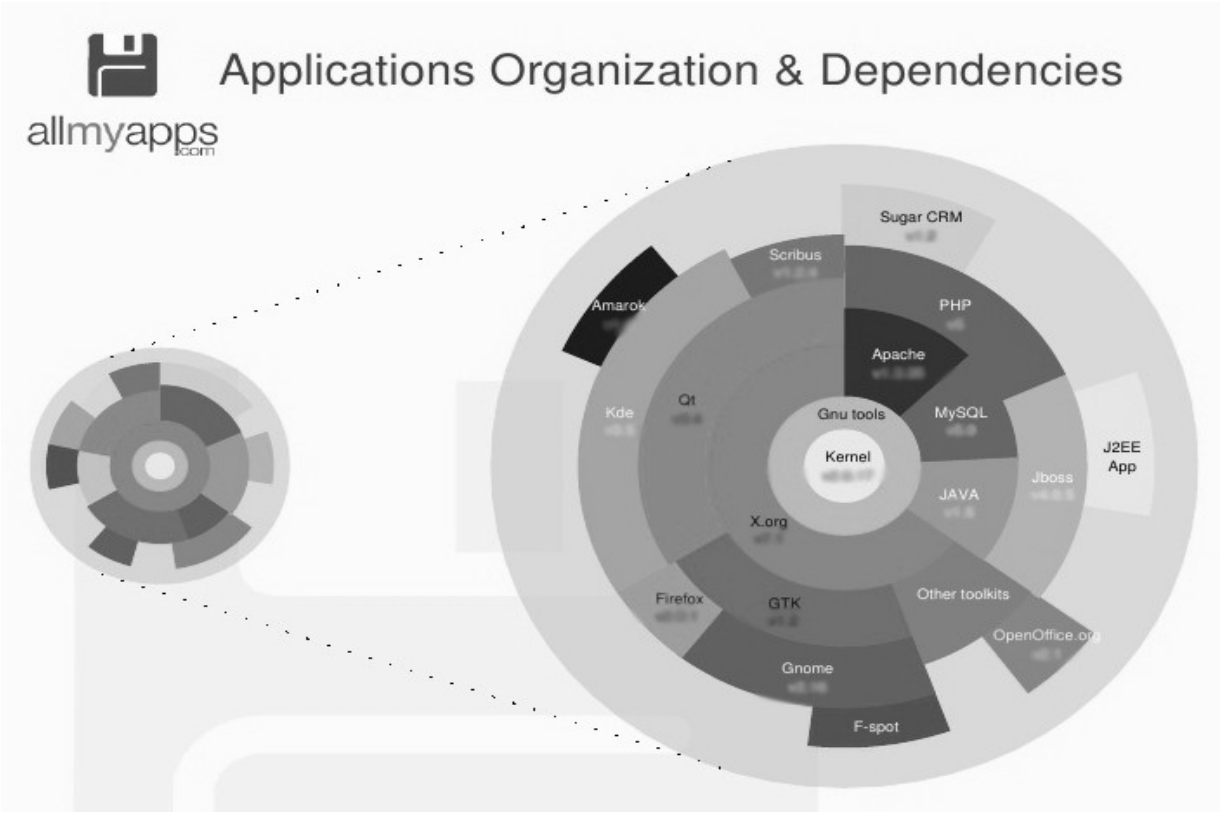

Ilustración 7: Organización de las aplicaciones y dependencias de la distribución cubana de GNU/Linux Nova.

Debido a la necesidad de, como se planteaba anteriormente, reducir la cantidad de aplicaciones y dependencias en la distribución cubana de GNU/Linux Nova, se propone una nueva arquitectura, basada en la arquitectura que utilizan algunos sistemas operativos que han reducido considerablemente las aplicaciones que poseen instaladas por defecto. El nuevo diagrama de arquitectura propuesta para la distribución cubana de GNU/Linux Nova se presenta en la ilustración 8 y se basa en el uso de cuatro capas para la construcción del sistema operativo, las cuales se explican a continuación, comenzando desde la de más bajo nivel hasta llegar a la capa que se encarga de la interacción del usuario con el ordenador:

1. Capa del Kernel de Linux: Contiene todos los controladores relacionados con periféricos de entrada y salida (teclado, *mouse*, cámara, pantalla, dispositivos externos, red y audio).
2. Capa de Bibliotecas: Esta capa contiene las bibliotecas necesarias para el funcionamiento del sistema haciendo una reducción considerable de las bibliotecas que posee actualmente la distribución cubana de GNU/Linux Nova instaladas por defecto. Las bibliotecas que la misma poseerá son las siguientes: libc (biblioteca estándar de C, es una biblioteca con funciones estándar que puede ser utilizada por todos los programas escritos en C). W*ebkit* (plataforma para aplicaciones que funciona como base para el navegador web Safari, Opera, Epiphany, Maxthon, Midori, QupZilla entre otros). SGL (*Scene Graph Library*, conjunto de bibliotecas multiplataforma de C++ construida encima de OpenGL que implementa la funcionalidad escena de gráficos 3D, algunos cargadores simples de modelos 3D tales como sgldb, sglobj, sgl3ds, y algunos servicios públicos diversos tales como sglu). LIBRESSL (Biblioteca que maneja el protocolo creado por *Netscape Socket Secure Layer (SSL)*[13] para garantizar transacciones seguras entre los servicios web y los navegadores). OpenGL (*Open Graphics Library* es una especificación estándar que define una Interfaz de Programación de Aplicaciones (API)[14] multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D). Freetype (biblioteca que implementa un motor de fuentes, utilizado para crear o modificar fuentes, desarrollándola en una interfaz intuitiva en donde cada carácter se rasteriza[15] en un mapa de bits). Sqlite (biblioteca escrita en C que brinda un sistema de gestión de bases de datos relacional). Por último el *media framework* (marco de *software* que se encarga de los medios de comunicación en un equipo y a través de una red).
3. Capa de aplicaciones del marco de trabajo: En esta capa se encuentran las aplicaciones mediante las cuales se puede gestionar el sistema. Entre ellas se encuentran: el gestor de actividades

13 Protocolo de seguridad de uso común que establece un canal seguro entre dos ordenadores conectados a través de Internet o de una red interna.

14 Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción.

15 Malla o matriz regular de celdas de un área determinada.

(utilizado por el sistema para gestionar sus actividades). El gestor de ventanas (Programa informático que controla la ubicación y apariencia de las ventanas bajo un sistema de ventanas en una interfaz gráfica de usuario). La vista del sistema, el gestor de paquetes, el gestor de recursos y el gestor de notificaciones.

4. Capa de aplicaciones de usuario: Esta capa proporciona un conjunto de aplicaciones que le permiten al usuario interaccionar con el ordenador y brinda las acciones básicas que todo usuario necesita en una computadora, esta capa posee aplicaciones tales como: editor de texto (edición de texto en formato sencillo). El paquete ofimática (permite realizar documentos científicos y presentaciones). Reproductor de audio y video (permite reproducir música y/o videos para brindar entretenimiento al usuario) y navegador web (permite al usuario navegar por la red).

A continuación se presenta la nueva arquitectura propuesta para la distribución cubana de GNU/Linux Nova. Para modelar la misma, fue necesario realizar un estudio de la matriz de trazabilidad de la distribución cubana de GNU/Linux Nova, mediante la cual se puede observar si alguna aplicación y/o biblioteca depende de otra y cuáles podrían ser reducidas sin afectar el funcionamiento del sistema. Entre las bibliotecas propuestas se encuentra el *fork* de OpenSSL llamado LibreSSL. Esta personalización de OpenSSL reduce considerablemente las líneas de código de OpenSSL, lo que le permitió reducir vulnerabilidades que contiene OpenSSL. Se presenta así que una vía factible para la reducción de vulnerabilidades en una aplicación es la reducción de código fuente innecesario en la misma. Es necesario añadir que las tres capas inferiores fueron reducidas al mínimo aceptado por el sistema para su correcto funcionamiento, mientras que la capa de aplicaciones del usuario se proponen algunas aplicaciones básicas para la interacción del usuario con el sistema.

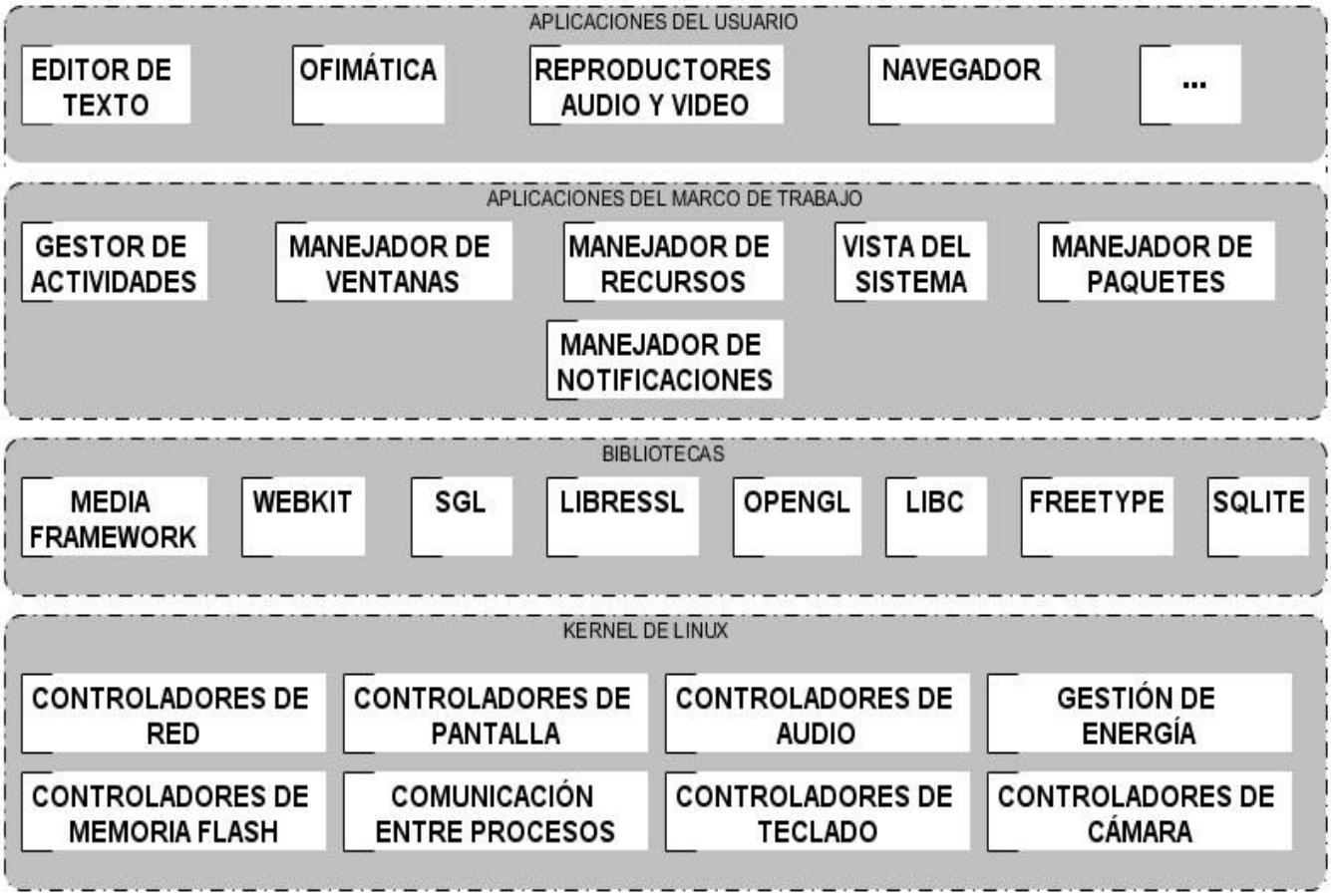

Ilustración 8: Propuesta de diagrama de arquitectura para la distribución de GNU/Linux Nova.

Esta propuesta para la arquitectura de la distribución cubana de GNU/Linux Nova, se logró realizar mediante el análisis de la matriz de trazabilidad de la distribución cubana de GNU/Linux Nova, en la cual se puede apreciar que solamente necesita 33 aplicaciones para el correcto funcionamiento del sistema. En la tabla 2 se listan las aplicaciones necesarias para el negocio las cuales pudieron obtenerse mediante la matriz de trazabilidad de la distribución cubana de GNU/Linux Nova, la cual se muestra en las ilustraciones 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20. En esta se listan en la primera columna los requerimientos y los paquetes del negocio del 1 al 104, y en la primera fila los paquetes, componentes y

casos de prueba del 1 al 117. La matriz de trazabilidad es una herramienta que se utiliza para saber que requerimientos quedan cubiertos por una prueba. Gracias a la matriz de trazabilidad, se puede observar que partes o módulos del *software* no están cubiertos por otras pruebas o identificar los requerimientos más críticos para saber si están lo suficientemente cubiertos.

Tabla 2: Aplicaciones necesarias para el correcto funcionamiento de la distribución cubana de GNU/Linux Nova.	
orca	firefox
nautilus	empathy
gcalctool	libreoffice
gnome-contact	evince
file-roller	totem
gnome-notes	gnome-sound-recorde
gnome-screenshot	rhythmbox
gnome-terminal	brasero
gusharpmap	baobad
palimsest	seahorse
yelp	gnome-system-monitor
gnome-dictionary	gnome-system-log
eog ó shotwell	software-center
simple-scan	usb-creator-gtk
nova-escritorio	thunderbird
alacarte	remina

Tabla 2: Aplicaciones necesarias para el correcto funcionamiento dela distribución cubana de GNU/Linux Nova.

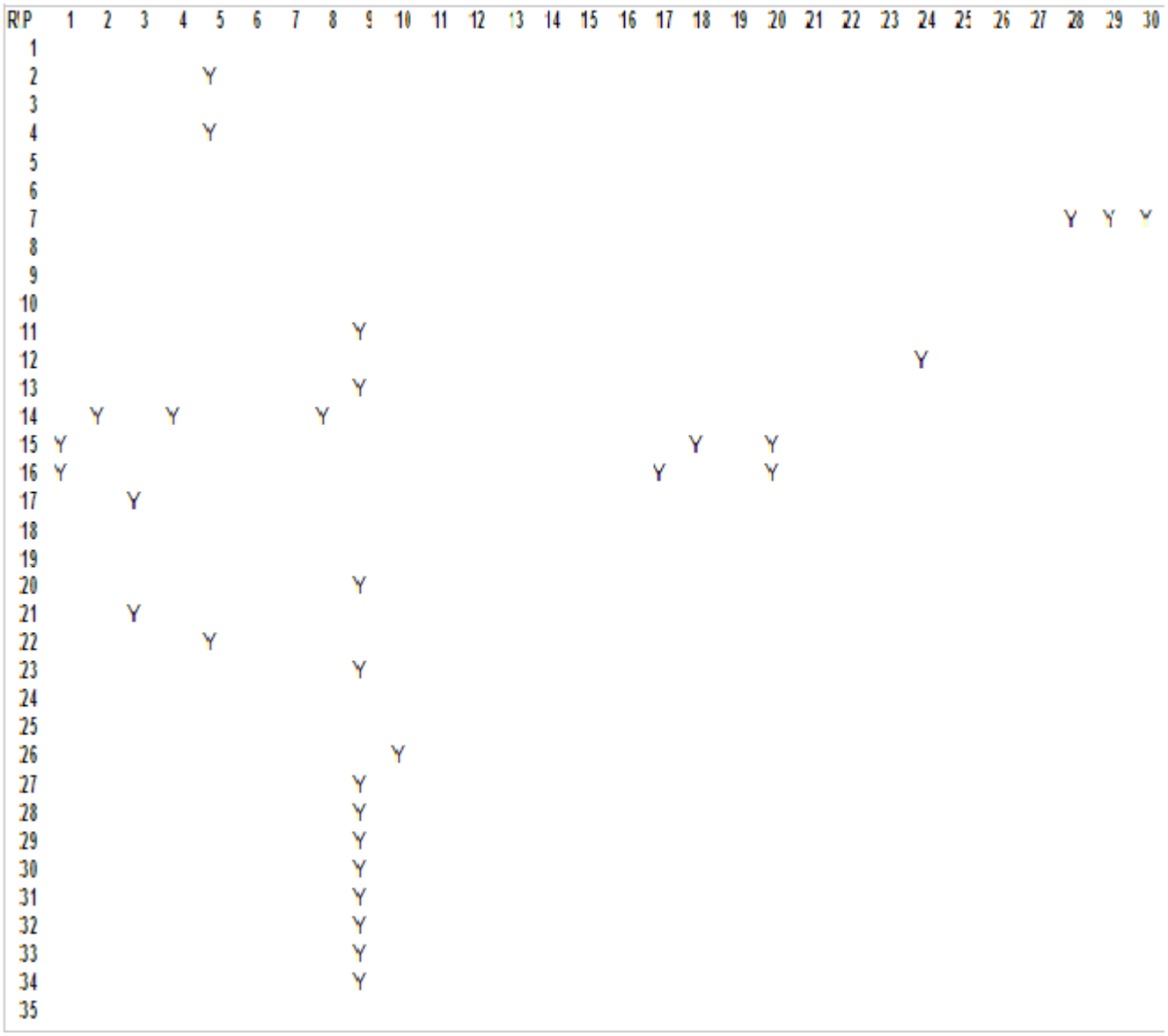

Ilustración 9: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

Ilustración 10: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

71				Y									
72				Y									
73				Y									
74				Y									
75													
76													
77		Y							Y				
78													
79													
80		Y						Y					
81				Y									
82													Y
83		Y			Y	Y	Y						
84													
85													
86			Y										
87											Y	Y	
88													
89		Y							Y				
90	Y									Y			
91		Y											
92													
93													
94													
95													
96													
97													
98				Y									
99		Y											
100													
101													
102													
103													
104			Y										

Ilustración 11: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

RNP	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
1																													
2																													
3																													
4																													
5																													
6																													
7	Y																												
8																													
9																													
10																													
11																													
12																													
13																													
14																													
15																													
16																													
17														Y	Y														
18		Y	Y																										
19								Y	Y																				
20																													
21																				Y	Y	Y							
22																													
23																													
24																													
25																													
26																													
27																													
28																													
29																													
30																													
31																													
32																													
33																													
34																													
35				Y	Y	Y	Y																						

Ilustración 12: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

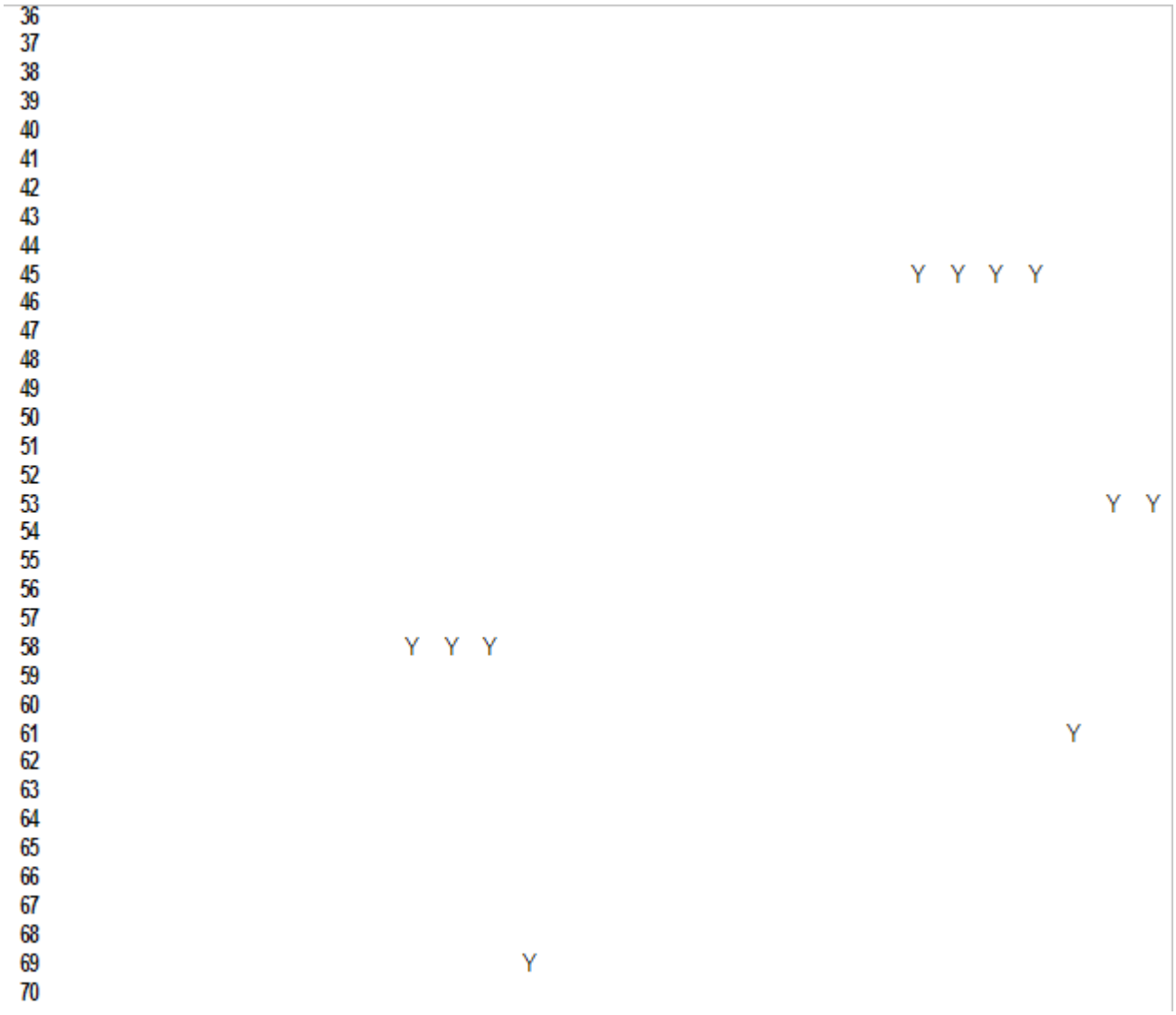

Ilustración 13: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

Ilustración 14: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

RP	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
1																														
2																														
3														Y													Y		Y	
4																														
5								Y																						
6										Y																				
7								Y																						
8												Y																		
9												Y																		
10												Y																		
11																														
12								Y																						
13																														
14							Y		Y		Y		Y				Y			Y										
15																														
16																														
17																														
18								Y																						
19								Y																						
20																														
21																														
22																														
23																														
24										Y																				
25										Y																				
26																														Y
27																														
28																														
29																														
30																														
31																														
32																														
33																														
34																														
35								Y																						

Ilustración 15: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

Ilustración 16: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

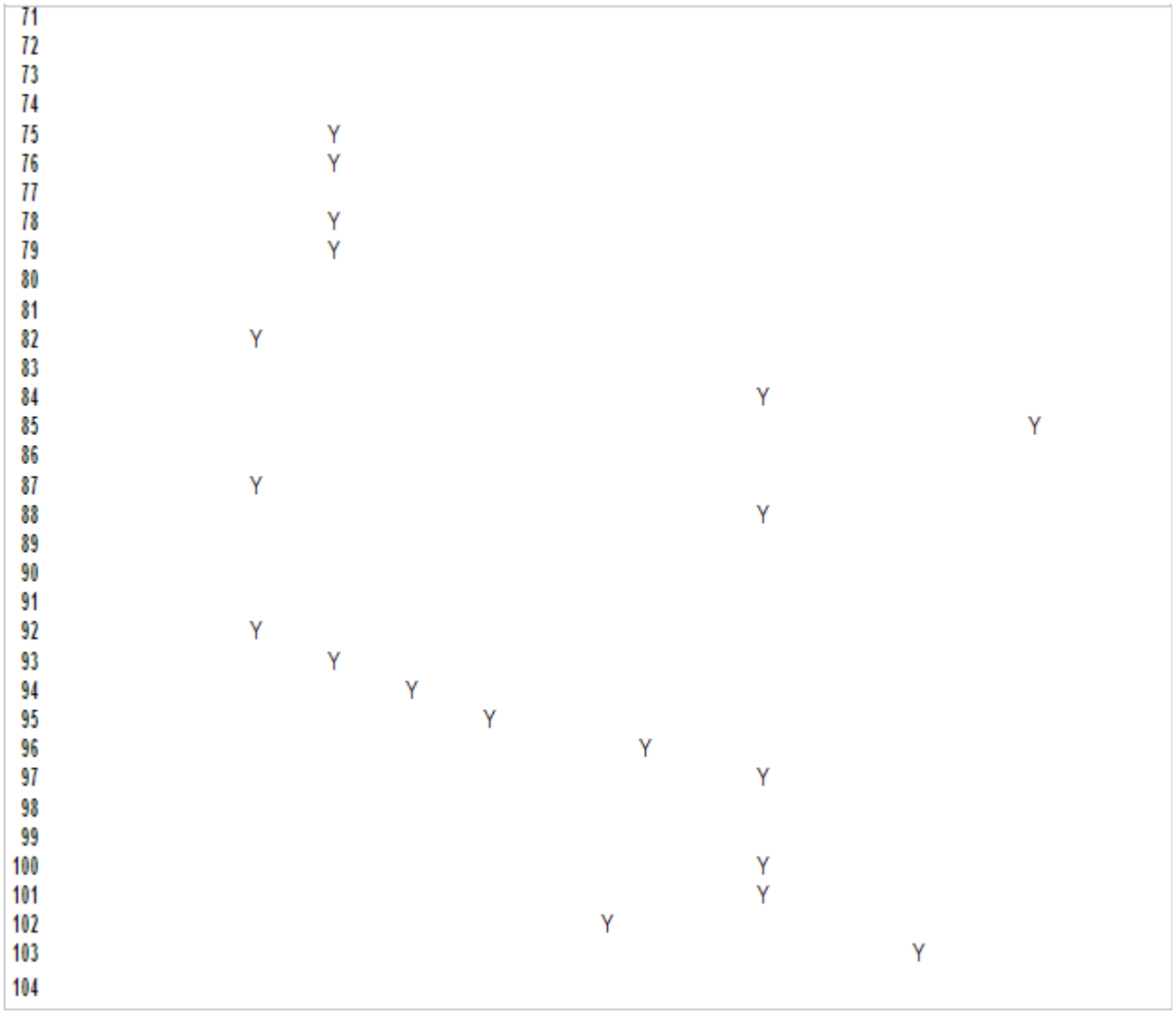

Ilustración 17: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

R\P	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117
1													Y														
2																											
3		Y	Y	Y	Y				Y	Y				Y													Y
4																											
5																											
6																											
7																											
8																											
9																											
10																											
11																											
12																											
13																											
14																											
15																											
16																											
17																											
18																											
19																											
20																											
21																											
22																											
23																											
24																											
25																											
26																											
27																											
28																											
29																											
30																											
31																											
32																											
33																											
34																											
35																											

Ilustración 18: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

Ilustración 19: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

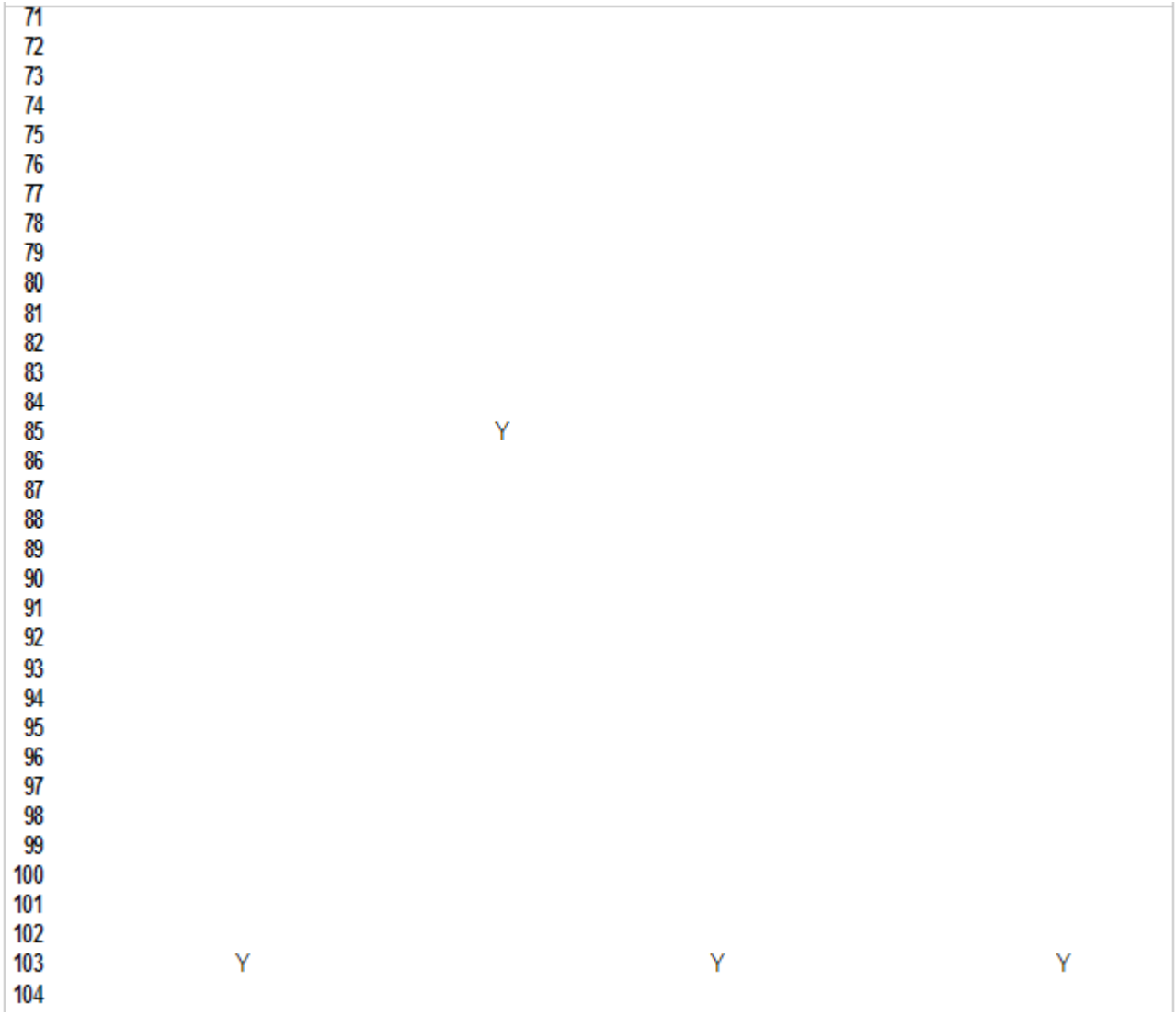

Ilustración 20: Matriz de trazabilidad de la distribución cubana de GNU/Linux Nova.

Conclusiones Parciales.

Luego del desarrollo del capítulo 2 de la presente investigación, se pudo llegar a las conclusiones de que es posible reducir el número de vulnerabilidades en una aplicación si se disminuyen la cantidad de líneas de código en la misma. El diseño del nuevo diagrama de arquitectura para la distribución cubana de GNU/Linux Nova permitió llegar a la conclusión de que es posible reducir el número de aplicaciones y bibliotecas que posee el sistema instaladas por defecto. El estudio de la matriz de trazabilidad de la distribución cubana de GNU/Linux Nova permitió detectar las aplicaciones necesarias para el correcto funcionamiento de la distribución cubana de GNU/Linux Nova. Después de analizar la distribución cubana de GNU/Linux Nova, se llegó a la conclusión de que reduciendo las bibliotecas y aplicaciones innecesarias para el sistema es posible reducir la probabilidad de aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova.

Capítulo 3: Validación de la propuesta.

En el presente capítulo se realiza una reducción de bibliotecas y aplicaciones innecesarias para el correcto funcionamiento de la distribución cubana de GNU/Linux Nova. También se realiza una comparación entre la biblioteca LibreSSL y OpenSSL las cuales poseen las mismas funcionalidades, sin embargo LibreSSL ha reducido el código de OpenSSL, reduciendo así la cantidad de vulnerabilidades que posee OpenSSL. Además, se realiza una comparación entre el sistema base utilizado para el Sistema Integrado de Gestión Estadística (SIGE) y un sistema de prueba generado para reducir al mínimo la cantidad de bibliotecas y aplicaciones para su correcto funcionamiento. A ambos sistemas se les aplica la fórmula para estimar la probabilidad de aparición de vulnerabilidades y se alcanza el resultado esperado.

3.1 Reducir la cantidad de bibliotecas en la distribución cubana de GNU/Linux Nova.

Al realizar un estudio detallado de los requisitos funcionales de la distribución cubana de GNU/Linux Nova 4.0 y establecer una relación de los mismos con las aplicaciones que necesita el sistema para funcionar correctamente se pudieron reducir 245 aplicaciones y bibliotecas que no son necesarias para el funcionamiento del sistema, las cuales se muestran en la tabla 4. Además, la tabla 3 presenta una relación entre los requisitos funcionales de la distribución cubana de GNU/Linux Nova 4.0 y las aplicaciones que necesita la misma para su correcto funcionamiento.

Tabla 3: Relación entre los requisitos funcionales de la distribución cubana de GNU/Linux Nova 4.0 y las aplicaciones y bibliotecas necesarias del sistema.

Nombre del requisito funcional	**Aplicaciones y bibliotecas en uso**
Proveer aplicaciones para la categoría accesos.	gnome-shell
Navegar por los archivos que forman parte del sistema.	nautilus
Mostrar ayuda para la realización de las actividades fundamentales del sistema.	yelp
Buscar archivos.	gnome-search-tool

Realizar cálculos matemáticos.	gcalctool, bc
Capturar pantalla.	gnome-screenshot
Gestionar contactos.	gnome-contact
Importar los contactos de aplicaciones de mensajería.	gnome-contact
Gestionar notas.	gnote
Mostrar mapa de caracteres.	gucharmap
Permitir interacción entre el sistema y el usuario por líneas de comandos.	xterm
Mostrar información sobre los dispositivos de almacenamiento.	usbutils
Gestionar archivos comprimidos.	file-roller
Brindar opciones para que las personas débiles visuales puedan realizar actividades en el sistema.	orca
Escanear imágenes.	simple scan
Editar imágenes.	shotwell
Organizar imágenes.	shotwell
Mostrar imágenes.	shotwell
Analizar utilización del disco duro.	baobab
Administrar *software* que forma parte del sistema.	software-center
Mostrar catálogo de *software*.	software-center
Buscar *software*.	software-center
Crear discos de arranque	usb-creator-gtk
Cambiar las preferencias del menú de Gnome.	gconf2
Guardar contraseñas en el sistema.	passwd
Mostrar el estado de los procesos que están corriendo en el sistema.	gnome-system-monitor
Editar procesos del sistema.	gnome-system-monitor
Mostrar el procesamiento de los recursos del sistema.	gnome-system-monitor
Mostrar los archivos del sistema.	nautilus

Mostrar los sucesos que han ocurrido en el sistema.	syslog
Administrar archivos de los sucesos que han ocurrido en el sistema.	systemd
Administrar correo electrónico.	thunderbird
Conectar la estación de trabajo a un escritorio remoto.	openssh-client
Enviar mensajes de manera instantánea.	empathy
Conectar a través de la web.	firefox
Integrar funcionalidades de administración de contactos con as relacionadas con los servicios de mensajería.	empathy
Gestionar hojas de cálculo.	calc
Gestionar presentaciones.	impress
Gestionar texto.	writter
Gestionar documentos con formato pdf.	evince
Grabar CD/ DVD	Brasero
Reproducir vídeos.	totem
Reproducir música.	rhythmbox
Organizar música.	rhythmbox
Grabar sonidos.	gnome-sound-recorder
Obtener fotos y vídeos desde una *webcam*.	cheese
Agrupar configuraciones del sistema.	gnome-control-center
Configurar la apariencia del sistema.	gnome-control-center
Configurar opciones de brillo.	gnome-control-center
Apagar el sistema	initscripts
Configurar idioma de teclado.	gnome-control-center
Configurar teclado.	gnome-control-center
Configurar cursor.	gnome-control-center
Mostrar diccionario	aspell
Configurar opciones de privacidad	zeitgeist
Mostrar diagnósticos de privacidad	zeitgeist

Conectar a *Bluethooth.*	gnome-bluetooth
Gestionar *drivers* del sistema.	xserver-xorg-input-all
Editar los perfiles de colores del sistema.	libcolord1
Cambiar tamaño del monitor del sistema.	libxrandr2
Gestionar servicio de red.	net-tools
Configurar servicios de impresión.	system-config-printer-gnome
Configurar opciones de sonido del sistema.	alsa-base
Configurar fecha y hora del sistema.	libtimedate-perl
Mostrar detalles del sistema	lshw
Mostrar opciones configuración para discapacitados.	gnome-system-monitor

Tabla 3: Relación entre requisitos funcionales de Nova y las aplicaciones y bibliotecas que necesita el sistema para su funcionamiento.

Tabla 4: Aplicaciones y bibliotecas reducidas de la distribución cubana de GNU/Linux Nova.	
acpid	brltty
acpi-support	cabextract
activity-log-manager-common	cmap-adobe-japan1
activity-log-manager-control-center	command-not-found
adium-theme-ubuntu	command-not-found-data
adobe-flash-properties-gtk	crda
adobe-flashplugin	cryptsetup-bin
aisleriot	dhcp3-client
alacarte	diffutils
apparmor	folks-common
apparmor-utils	fonts-droid
apport	fonts-horai-umefont
apport-gtk	fonts-liberation
apport-symptoms	fonts-unfonts-core
apt-transport-https	foomatic-db-engine
apt-xapian-index	freepats

augeas-lenses	friendly-recovery
avahi-autoipd	geoip-database
avahi-daemon	gettext
avahi-utils	gettext-base
bamfdaemon	ginn
binfmt-support	gir1.2-appindicator3-0.1
bluez-alsa	gir1.2-atspi-2.0
bluez-cups	gir1.2-dbusmenu-gtk-0.4
bluez-gstreamer	gir1.2-gtk-2.0
gir1.2-gtksource-3.0	guile-1.8-libs
gir1.2-gudev-1.0	gvfs-backends
gir1.2-launchpad-integration-3.0	gvfs-fuse
gir1.2-notify-0.7	hplip
gir1.2-panelapplet-4.0	hplip-data
gir1.2-timezonemap-1.0	hunspell-en-ca
gir1.2-wnck-3.0	hunspell-en-us
gnome-accessibility-themes	hyphen-en-us
gnome-applets	icoutils
gnome-applets-data	imagemagick
gnome-exe-thumbnailer	imagemagick-common
gnome-screensaver	irqbalance
gnome-session-fallback	iw
gnome-terminal	jockey-common
gnome-terminal-data	jockey-gtk
gnome-user-guide	krb5-locales
gnome-user-share	libapparmor-perl
grub-pc	libapparmor1
grub-pc-bin	libart-2.0-2
grub2-common	libaugeas0

gstreamer0.10-crystalhd	libavcodec53
gstreamer0.10-ffmpeg	libavformat53
gstreamer0.10-fluendo-mp3	libbamf0
gstreamer0.10-plugins-base-apps	libbrlapi0.5
gstreamer0.10-tools	libcapi20-3
gtk3-engines-unico	libcdio-cdda1
libcdio-paranoia1	libgif4
libcdt4	libglib-perl
libclass-factory-util-perl	libgnome-menu2
libclass-load-perl	libgpgme11
libclass-singleton-perl	libgphoto2-l10n
libcrypt-passwdmd5-perl	libgraph4
libcryptsetup4	libgtk2-perl
libcrystalhd3	libgtksourceview-3.0-0
libdaemon0	libgtksourceview-3.0-common
libdata-optlist-perl	libgvc5
libdatetime-format-builder-perl	libhpmud0
libdatetime-format-iso8601-perl	libhtml-form-perl
libdatetime-format-strptime-perl	libhtml-format-perl
libdatetime-locale-perl	libhtml-parser-perl
libdatetime-perl	libhtml-tagset-perl
libdatetime-timezone-perl	libhtml-tree-perl
libencode-locale-perl	libhttp-cookies-perl
libexttextcat-data	libhttp-daemon-perl
libexttextcat0	libhttp-date-perl
libfile-basedir-perl	libhttp-message-perl
libfile-desktopentry-perl	libhttp-negotíate-perl
libfile-listing-perl	libilmbase6
libfile-mimeinfo-perl	libindicate-gtk3

libfont-afm-perl	libindicate5
libgc1c2	libio-socket-inet6-perl
libgettextpo0	libio-socket-ssl-perl
libjs-jquery	libodbc1
liblist-moreutils-perl	liboil0.3
liblockfile-bin	libopencore-amrwb0
liblockfile1	libopenexr6
liblouis-data	libpackage-deprecationmanager-perl
liblouis2	libpackage-stash-perl
liblqr-1-0	libpackage-stash-xs-perl
liblwp-mediatypes-perl	libpango-perl
liblwp-protocol-https-perl	libparams-classify-perl
liblzo2-2	libparams-util-perl
libmad0	libparams-validate-perl
libmagickcore4	libpathplan4
libmagickcore4-extra	libpostproc52
libmagickwand4	libproxy1-plugin-gsettings
libmailtools-perl	libproxy1-plugin-networkmanager
libmath-round-perl	libpth20
libmeanwhile1	libquvi-scripts
libmetacity-private0	librpc-xml-perl
libminiupnpc8	libsocket6-perl
libmodule-runtime-perl	libss2
libmpg123-0	libsub-install-perl
libnet-http-perl	libterm-readkey-perl
libnet-ssleay-perl	libtry-tiny-perl
libnetpbm10	libunistring0
libnewt0.52	liburi-perl
libnss-mdns	libva1

libwww-perl	unixodbc
libwww-robotrules-perl	vino
libxml-parser-perl	w3m
migration-theme	wamerican
mobile-broadband-provider-info	wbritish
modemmanager	wine
mtools	wine-gecko1.4
notify-osd-icons	wine1.4
odbcinst	wine1.4-common
odbcinst1debian2	wine1.4-i386
packagekit-backend-aptcc	winetricks
plymouth-label	wspanish
plymouth-theme-nova-2013	xz-lzma
pppconfig	ttf-liberation
pppoeconf	ttf-umefont
rtkit	ttf-unfonts-core
syslinux-legacy	ttf-wqy-microhei
tar -tasksel	ubuntu-cloud-keyring
tasksel-data	ubuntu-docs
ttf-droid	

Tabla 4: Aplicaciones y bibliotecas reducidas de la distribución cubana de GNU/Linux Nova.

Tomando en cuenta la relación de vulnerabilidades que contienen algunas de las bibliotecas anteriormente tratadas, fueron detectadas 69 vulnerabilidades en el período 2013-2015, en la tabla 5 se muestran ejemplos de algunas de estas (28) (29) (30) (31) (32) (33) (34) (35) (36) (37) (38) (39) (40) (41) (42) (43).

Tabla 5: Ejemplos de algunas vulnerabilidades que podrían haber sido prevenidas en el período 2013-2015.		
Año	**Cantidad de vulnerabilidades**	**Algunas Vulnerabilidades**
2013	27	CVE-2012-6607, CVE-2012-0787, CVE-2012-0786, CVE-2013-4854, CVE-2013-3919, CVE-2013-2266, CVE-2012-5689, CVE-2013-4298, CVE-2011-4969, CVE-2013-1415, CVE-2002-2443, CVE-2013-4391, CVE-2013-4327, CVE-2013-5745.
2014	32	CVE-2014-1419, CVE-2014-0484, CVE-2014-6273, CVE-2014-0490, CVE-2014-0489, CVE-2014-8680, CVE-2014-8500, CVE-2014-3859, CVE-2014-3214, CVE-2014-0591, CVE-2011-2198, CVE-2014-3564, CVE-2014-1236, CVE-2014-0978, CVE-2014-4345, CVE-2014-4344, CVE-2014-4343, CVE-2014-5271, CVE-2012-5580, CVE-2014-3158.
2015	10	CVE-2015-1349, CVE-2014-1949, CVE-2014-9421, CVE-2014-5355, CVE-2014-5352, CVE-2015-3395, CVE-2015-3310.
Total	69	

Tabla 5: Vulnerabilidades período 2013-2015 en aplicaciones y bibliotecas innecesarias para el funcionamiento de la distribución cubana de GNU/Linux Nova.

3.2 Reducción de cantidad de líneas de código.

La estructura modular de la mayoría de las aplicaciones de la distribución cubana de GNU/Linux Nova permite la compilación alternativa de líneas de código. Tomando como ejemplo el Kernel en la configuración de los ordenadores ASUS, distribuidos por la ensambladora de *hardware* cubana GEDEME, el sistema tiene instalados por defecto 3541 *drivers* de los 76 que realmente necesita para funcionar. De haberse compilado el Kernel solo con las funcionalidades que necesita se hubiesen podido eliminar de antemano gran número de vulnerabilidades. Un ejemplo de esto se presenta en las tabla 6, en la cual se hace referencia a 13 vulnerabilidades en el período 2013-2015 relacionadas con los módulos del Kernel que manejan IPv6[16], las cuales no hubiesen estado presentes en la distribución cubana de GNU/Linux

16 Versión 6 del protocolo Internet *Protocol*, definida en el RFC 2460 y diseñada para reemplazar a Internet *Protocol* versión 4, que actualmente está implementado en la mayoría de los dispositivos que acceden a Internet.

Nova de haberse compilado el Kernel sin estos módulos.

Tabla 6: Vulnerabilidades que podrían haber sido evitadas en el Kernel de la distribución cubana de GNU/Linux Nova en el período 2013-2015 (44) (45) (46).				
Año	**CVE ID**	**Tipo de vulnerabilidad**	**Nivel**	**Vulnerable**
2013	CVE-2013-0343	Dos +Info	3.2	SI
	CVE-2013-2232	DoS	4.9	SI
	CVE-2013-6431	Dos	4.7	SI
	CVE-2013-4125	DoS	5.4	SI
	CVE-2013-4162	DoS	4.7	SI
	CVE-2013-4163	DoS	4.7	SI
	CVE-2013-4387	DoS Overflow Mem. Corr.	6.1	SI
	CVE-2013-4470	DoS +Priv Mem. Corr.	6.9	SI
	CVE-2013-4563	DoS	7.1	SI
	CVE-2013-6431	DoS	4.7	SI
2014	CVE-2014-2309	DoS	6.1	SI
	CVE-2014-7207	DoS	4.9	SI
2015	CVE-2015-2922	+Info	3.3	SI
Total	13			

Tabla 6: Vulnerabilidades que podrían haber sido evitadas en el Kernel de la distribución cubana de GNU/Linux Nova en el período 2013-2015.

Otra forma de reducir líneas de código es modificando el código fuente de la aplicación o biblioteca con el objetivo de refinarlo y reducir la cantidad de líneas de código innecesarias. Esto ha sido demostrado por el proyecto Libressl. LibreSSL es un *fork* de OpenSSL y una versión de la biblioteca *Transport Layer Security (TLS*)[17], creado por OpenBSD y desarrollado con el objetivo de modernizar el código base, mejorar la seguridad y aplicar mejores prácticas en el proceso de desarrollo (48). LibreSSL ha eliminado una serie de características de OpenSSL que puede causar problemas de compilación para el *software* en

17 Protocolo que garantiza la seguridad e integridad de los datos entre aplicaciones cliente/servidor que se comunican a través de Internet.

el cual se basan ellos (47).

El desarrollo primario ocurre en el interior del árbol de fuentes de OpenBSD con la atención habitual por la cual el proyecto es conocido. Sobre una base regular, el código es reempaquetado para el uso portátil por otros sistemas operativos. (Linux, FreeBSD, Windows, etcétera) (48).

LibreSSL está compuesta por cuatro partes (48):

1. La utilidad *OpenSSL*, que provee herramientas para administrar certificados llaves, etcétera.
2. Libcrypto: una biblioteca de los fundamentos de criptografía.
3. Libssl: una biblioteca TLS, compatible con OpenSSL.
4. Libtls: una nueva biblioteca TLS, diseñada para que sea fácil desarrollar pruebas para todas las aplicaciones.

La tabla 7 representa una relación entre la cantidad de vulnerabilidades detectadas en LibreSSL y OpenSSL desde la liberación de la primera versión de LibreSSL (47). Mientras que en las tablas 8 y 9 se presenta una relación entre las vulnerabilidades detectadas en LibreSSL y OpenSSL en el período 2014-2015.

Tabla 7: Cantidad de vulnerabilidades detectadas en LibreSSL y OpenSSL desde la versión inicial de LibreSSL.

Severidad	**Vulnerabilidades LibreSSL**	**Vulnerabilidades OpenSSL**
Alta	0	4
Media (Moderada)	12	22
Baja	7	9
Total	19	35

Tabla 7: Cantidad de vulnerabilidades detectadas en LibreSSL y OpenSSL desde la versión inicial de LibreSSL.

Tabla 8: Relación entre las vulnerabilidades detectadas en LibreSSL y OpenSSL en el año 2014.					
Fecha	**Referencia CVE**	**Severidad**	**LibreSSL**	**OpenSSL**	**Lineas de Código LibreSSL/OpenSSL**
06-08-2014	CVE-2014-3508	Media	Parcialmente vulnerable	Vulnerable	240532/308096
	CVE-2014-5139	Media	No vulnerable	Vulnerable	240532/308096
	CVE-2014-3512	Alta	No vulnerable	Vulnerable	240532/308096
15-10-2014	CVE-2014-3513	Media	No vulnerable	Vulnerable	240375/308096
	CVE-2014-3567	Alta	No vulnerable	Vulnerable	240375/308096
Total	5				

Tabla 8: Relación entre vulnerabilidades detectadas en LibreSSL y OpenSSL en el año 2014 .

Tabla 9: Relación entre las vulnerabilidades detectadas en LibreSSL y OpenSSL en el año 2015.					
Fecha	**Referencia CVE**	**Severidad**	**LibreSSL**	**OpenSSL**	**Líneas de Código LibreSSL/OpenSSL**
08-01-2015	CVE-2014-8275	Baja	Corregido en la 2.1.4	Vulnerable	251221/308258
	CVE-2014-3572	Baja	Corregido en la 2.1.4	Vulnerable	251221/308258
	CVE-2014-3570	Baja	Corregido en la 2.1.4	Vulnerable	251221/308258
	CVE-2015-0205	Baja	Corregido en la 2.1.4	Vulnerable	251221/308258
	CVE-2015-0206	Media	Corregido en la 2.1.3	Vulnerable	251221/308258
	CVE-2014-3571	Media	No vulnerable	Vulnerable	251221/308258
	CVE-2014-3569	Baja	No vulnerable	Vulnerable	251221/308258
	CVE-2015-0204	Baja	No vulnerable	Vulnerable	251221/308258
19-03-2015	CVE-2015-0209	Baja	No vulnerable	Vulnerable	258815/308258
	CVE-2015-0291	Alta	Código no presente	Vulnerable	258815/308258
	CVE-2015-0204	Alta	Corregido en 2.1.2	Vulnerable	258815/308258
	CVE-2015-0207	Media	No vulnerable	Vulnerable	258815/308258
	CVE-2015-0208	Media	Código no presente	Vulnerable	258815/308258

	CVE-2015-0290	Media	Código no presente	Vulnerable	258815/308258
	CVE-2015-0292	Media	Corregido en 2.0.0	Vulnerable	258815/308258
	CVE-2015-0293	Media	Código no presente	Vulnerable	258815/308258
	CVE-2015-0285[18]	Baja	No vulnerable	Vulnerable	258815/308258
	CVE-2015-1787	Media	Corregido en 2.0.1	Vulnerable	258815/308258
Total	18				

Tabla 9: Relación entre vulnerabilidades detectadas en LibreSSL y OpenSSL en el año 2015.

Del análisis de LibreSSL con respecto a OpenSSL, se demuestra que mediante la reducción y refinación del código fuente, es posible reducir considerablemente la probabilidad de aparición de vulnerabilidades. Mediante esta técnica, LibreSSL logró evitar 15 vulnerabilidades que afectaron a OpenSSL y pudo corregir otras 8 vulnerabilidades antes de que fueran detectadas y corregidas en OpenSSL.

3.3. Caso de estudio distribución para soporte del Sistema Integrado de Gestión Estadística.

En los últimos 3 años la distribución cubana de GNU/Linux Nova se ha utilizado como el sistema operativo sobre el cual se instalan las aplicaciones desarrolladas por la UCI para la administración central del estado. Ejemplo de ello han sido el sistema dedicado a las elecciones “Nova Elecciones”, el Sistema de Gestión Fiscal (SIGF) y el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos (SITPC).

A continuación se exponen las características del sistema que se utiliza para el Sistema Integrado de Gestión Estadística (SIGE) que se despliega actualmente en todos los tribunales de la isla basado en la distribución cubana de GNU/Linux Nova 4.0.

Tabla 10: Relación entre SPE del sistema base del SIGE el número de vulnerabilidades que contiene este en el período 2010-2015.

Aplicaciones y bibliotecas	2010	2011	2012	2013	2014	2015
apt		1	3	1	8	

18 Esta vulnerabilidad no puede ocurrir por el diseño del generador de números pseudoaleatorios (PRNG) en LibreSSL.

bind9	9	6	7	5	5	1
krb5	11	11	4	7	8	6
libpng	3	7	3		4	2
OpenSSL	12	14	16	4	24	20
perl	1	3	4	2	2	
python	7	2	5	2	6	
php	35	35	22	13	32	26
postgresql	7	1	9	6	9	1
linux kernel	124	83	115	189	133	40
Total	209	163	188	229	231	96

Tabla 10: SPE que contiene el sistema que se utiliza como base para el SIGE y número de vulnerabilidades que contienen las mismas en el período 2010-2015.

De las 174 aplicaciones y bibliotecas que componen el sistema que se utiliza como base para el SIGE, 10 son SPE (tabla 10). Si asumimos que el *software* desarrollado en la UCI no contiene vulnerabilidades la fórmula para el cálculo de la probabilidad de errores de este sistema quedaría de la forma:

$$P_{Vul} = \frac{10}{F_{NOVA} + F_{SIGE}}$$

Como prueba de concepto se plantea que el desarrollo de esta aplicación utilizando otras tecnologías no propensas a errores, por ejemplo nodejs[19], permitiría la inclusión de menos SPE reduciendo la probabilidad de aparición de vulnerabilidades en el sistema. Para demostrar esto se desarrolló un *script* (Anexo 2, Anexo 3) que genera un sistema operativo funcional solo utilizando las aplicaciones que se reflejan en la tabla 11, y al cual nos referiremos desde ahora como NMEPLBB por las aplicaciones y las bibliotecas que el mismo posee.

19 Entorno de programación en la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basada en el motor V8 de Google.

Tabla 13: Relación entre las aplicaciones del sistema de prueba NMEPLBB y el número de vulnerabilidades que contiene este en el período 2010-2015*.						
Aplicaciones y bibliotecas	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**
nodejs *		2	2	6	4	5
mongoDB				4	2	1
eglibc				1	1	
pcre						
linux kernel *	124	83	115	189	133	40
busybox				1		
boost				1		
Total	124	85	117	202	140	46

Tabla 11: Aplicaciones que contiene el sistema NMEPLBB y vulnerabilidades que contienen las mismas en el período 2010-2015.

Como se puede apreciar el sistema NMEPLBB posee solamente 2 SPE por lo que la fórmula para el mismo quedaría de la siguiente forma:

$$P_{Vul} = \frac{2}{F_{NOVA} + F_{SIGE}}$$

En la tabla 12, se realiza una comparación entre el sistema utilizado como base del SIGE y el sistema de prueba NMEPLBB. Al aplicar la fórmula a la distribución utilizada como base del SIGE, el resultado de la misma denota que la probabilidad de existencia de vulnerabilidades en el mismo es mayor que al aplicar la fórmula en el sistema de prueba NMEPLBB, el cual solamente contiene dos SPE. Tomando en cuenta que un sistema posee en el mejor de los casos tantas funcionalidades como bibliotecas y aplicaciones posee, en ambos sistemas se mantiene la misma cantidad de funcionalidades.

Por lo tanto, se puede decir que se redujo la probabilidad de aparición de vulnerabilidades en el sistema generado frente a la distribución utilizada como base del SIGE. Además se redujeron 402 vulnerabilidades

* *Software* con Propensión a Errores que contiene el sistema de prueba NMEPLBB.

en el sistema de prueba NMEPLBB con respecto la distribución utilizada como base del SIGE en el período 2010-2015.

Tabla 12: Tabla comparativa entre la probabilidad de aparición de vulnerabilidades en el Sistema Integral de Gestión Estadística y el sistema NMEPLBB.

Sistema	Cantidad de SPE en los sistemas	SPE	Cantidad de vulnerabilidades en el período 2010-2015	Probabilidad de aparición de vulnerabilidades
SIGE	10	apt	1116	$P_{Vul} = \frac{10}{F_{NOVA} + F_{SIGE}}$
		bind9		
		krb5		
		libpng		
		OpenSSL		
		perl		
		python		
		php		
		postgresql		
		linux kernel		
NMEPLB	2	nodejs	714	$P_{Vul} = \frac{2}{F_{NOVA} + F_{SIGE}}$
		linux kernel		
Cantidad de vulnerabilidades reducidas: 402				

Tabla 12: Comparación entre la probabilidad de aparición de vulnerabilidades en el Sistema Integral de Gestión Estadística y el sistema NMEPLBB.

Conclusiones Parciales.

Al concluir el capítulo 3 de la presente investigación se pudo reducir la cantidad de vulnerabilidades en la distribución cubana de GNU/Linux Nova reduciendo la cantidad de aplicaciones y bibliotecas que la misma utiliza. También se demostró que es posible reducir la cantidad de vulnerabilidades en un *software* reduciendo la cantidad de líneas de código innecesario que el mismo presenta. Además, se pudo reducir la probabilidad de aparición de vulnerabilidades en una distribución reduciendo la cantidad de *software*

con propensión a errores que se utiliza en el funcionamiento de la misma.

Conclusiones.

Al finalizar la presente investigación se puede concluir que:

1. Los análisis estático y dinámico de código no ofrecen una solución completa para la reducción de vulnerabilidades en una distribución FOSS.
2. La corrección de vulnerabilidades en *software* de terceros es comúnmente más costosa que la implementación de las funcionalidades que provee este.
3. Existe en el mundo del FOSS aplicaciones que son más propensas a la aparición de errores y la mayoría de estas son desarrolladas por comunidades con poco o ningún financiamiento.
4. La cantidad de *software* propenso a errores incluido en una distribución de FOSS es directamente proporcional a la probabilidad de aparición de errores en esta.
5. Se pueden reducir errores en las aplicaciones ya existentes reduciendo las líneas de código en ella que no son válidas para el negocio en que se utilicen.
6. Se pueden reducir los errores de una distribución FOSS reduciendo las bibliotecas y aplicaciones en ellas que no son válidas para el negocio en que se utilicen.
7. Se pueden desarrollar sistemas a la medida cuya probabilidad de aparición de errores tienda a 0.
8. Se pueden cambiar las bibliotecas y aplicaciones existentes en un sistema por homólogas con menos propensión a errores.

Por lo anterior se puede afirmar que el objetivo de proponer vías para disminuir la probabilidad de aparición de vulnerabilidades en la distribución cubana de GNU/Linux Nova ha sido cumplido.

Recomendaciones.

En la presente investigación se plantea el desarrollo de aplicaciones y sistemas propios. Esto trae como consecuencia directa, la necesidad de crear nuestras propias funcionalidades para los sistemas programados. Debido a lo planteado anteriormente, se recomienda el estudio de la aparición de vulnerabilidades en un sistema desde el punto de vista de la probabilidad de aparición de vulnerabilidades en las funcionalidades implementadas en el país.

Referencias Bibliográficas.

1. CHÁVEZ FRÍAS, Hugo, 2004, Decreto 3390. [online]. 23 December 2004. [Accessed 21 May 2015]. Available from: http://www.softwarelibre.gob.ve/images/stories/leyes/decreto3390softwarelibre.pdf

2. CORREA DELGADO, Rafael, 2008, Decreto 1014. [online]. 10 April 2008. [Accessed 21 May 2015]. Available from: http://www.administracionpublica.gob.ec/wp-content/uploads/downloads/2014/06/DecretoEjecutivo1014.pdf

3. MORALES AMAYA, Evo, [no date], Decreto supremo No. 1793. [online]. [Accessed 26 May 2015]. Available from: http://www.redipd.org/legislacion/common/legislacion/Bolivia/DS_1793_Telecomunicaciones.pdf

4. MUJICA, José, BONOMI, Eduardo, PORTO, Luis, POLGAR, Jorge, FERNÁNDEZ HUIDOBRO, Eleuterio, EHRLICH, Ricardo, PINTADO, Enrique, BAYARDI, José, MUÑIZ, Susana, BENECH, Enzo, KECHICHIAN, Liliam, LEJTREGER, Raquel and OLESKER, Daniel, 2015, Decreto N° 44/015 [online]. 6 February 2015. [Accessed 26 May 2015]. Available from: http://www.impo.com.uy/bases/decretos/44-2015

5. Revelations | Courage Snowden, [no date]. [online], [Accessed 27 May 2015]. Available from: https://edwardsnowden.com/es/revelations/

6. ¿Cuál es la definicion de Vulnerabilidad? [online]. [Accessed 18 June 2015]. Available from: http://www.alegsa.com.ar/Dic/vulnerabilidad.php

7. Rebelion. El Software Comunitario es la unión del Software Libre y el socialismo científico. [online]. [Accessed 18 June 2015]. Available from: http://www.rebelion.org/noticia.php?id=104875

8. Definición de Sistema operativo?» Concepto en Definición ABC. [online]. [Accessed 18 June 2015]. Available from: http://www.definicionabc.com/tecnologia/sistema-operativo.php

9. Distribución Linux - Wikipedia, la enciclopedia libre. [online]. [Accessed 18 June 2015]. Available from: https://es.wikipedia.org/wiki/Distribuci%C3%B3n_Linux

10. The Critical Security Controls for Effective Cyber Defense [online]. SANS. [Accessed 12 June 2015]. Available from: https://www.sans.org/media/critical-security-controls/CSC-5.pdf

11. EXPÓSITO, Raul, [no date], ¿Qué es el Análisis Estático del Código? [online].

[Accessed 21 May 2015]. Available from: http://raulexposito.com/documentos/analisis-estatico-codigo/
12. Static code analysis, [no date]. [online], [Accessed 22 May 2015]. Available from: http://www.viva64.com/en/t/0046/
13. LLUNA, Eduardo, 2011, Análisis estático de código en el ciclo de desarrollo de software de seguridad crítica. [online]. 2011. Vol. Revista Española de Innovación, Calidad e Inge niería del Software. [Accessed 21 May 2015]. Available from: http://www.redalyc.org/pdf/922/92222551004.pdf
14. Analisis dinamico, [no date]. [online], [Accessed 21 May 2015]. Available from: http://mit.ocw.universia.net/6.170/6.170/f01/pdf/lecture-10.pdf
15. How to Prevent the next Heartbleed, [no date]. [online], [Accessed 22 May 2015]. Available from: http://www.dwheeler.com/essays/heartbleed.html
16. What is zero-day exploit? - Definition from WhatIs.com, [no date]. [online], [Accessed 26 May 2015]. Available from: http://searchsecurity.techtarget.com/definition/zero-day-exploit
17. BALL, Thomas, [no date], The Concept of Dynamic Analysis [online]. Bell Laboratories, Lucent Technologies. [Accessed 21 May 2015]. Available from: http://research.microsoft.com/en-us/um/people/tball/papers/fse-concept.pdf
18. Valgrind: About, [no date]. [online], [Accessed 28 May 2015]. Available from: http://valgrind.org/info/about.html
19. Using Valgrind to Find Memory Leaks - Cprogramming.com. [online]. [Accessed 9 June 2015]. Available from: http://www.cprogramming.com/debugging/valgrind.html
20. Wheeler: How to Prevent the next Heartbleed [LWN.net], [no date]. [online], [Accessed 28 May 2015]. Available from: http://lwn.net/Articles/596890/
21. How to Detect a Zero-Day Threat | Seculert Blog on Breach Detection, [no date]. [online], [Accessed 27 May 2015]. Available from: http://www.seculert.com/blog/2013/03/how-to-detect-zero-day-threat.html
22. OpenSSL OpenSSL?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 9 June 2015]. Available from: http://www.cvedetails.com/product/383/OpenSSL-OpenSSL.html?vendor_id=217

23. Linux Linux Kernel?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 9 June 2015]. Available from: http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

24. Redhat OpenSSL?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 9 June 2015]. Available from: http://www.cvedetails.com/product/3966/Redhat-OpenSSL.html?vendor_id=25

25. Busybox Busybox?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 9 June 2015]. Available from: http://www.cvedetails.com/product/7452/Busybox-Busybox.html?vendor_id=4282

26. COBB, Michael. S7260713: Integrating Vulnerability Management in the Development Process. Reporte. Dark Reading, 2013.

27. MACHÍN CASTILLO, Jorge Luis and CORSO BOUZA, Anay, 2010, *Descripción del proceso de construcción del sistema operativo base de la distribución cubana de GNU/Linux Nova.* La Habana : Universidad de las Ciencias Informáticas.

28. Canonical Acpi-support?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/28172/Canonical-Acpi-support.html?

29. Debian APT?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/17236/Debian-APT.html?vendor_id=23

30. Augeas Augeas?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/26487/Augeas-Augeas.html?vendor_id=12963

31. ISC Bind?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/144/ISC-Bind.html?vendor_id=64

32. Gnome Gnome-terminal?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/2855/Gnome-Gnome-terminal.html?vendor_id=283

33. GNU Gpgme?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/10513/GNU-Gpgme.html?vendor_id=72
34. Graphviz?: Products and vulnerabilities. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/vendor/3872/Graphviz.html
35. GTK Gtk+?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/1134/GTK-Gtk-.html?vendor_id=666
36. Imagemagick?: Products and vulnerabilities. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/vendor/1749/Imagemagick.html
37. Jquery Jquery?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/11031/Jquery-Jquery.html?vendor_id=6538
38. MIT Kerberos?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/61/MIT-Kerberos.html?vendor_id=42
39. Libav Libav?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/20944/Libav-Libav.html?vendor_id=11428
40. Libproxy Project Libproxy?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/23542/Libproxy-Project-Libproxy.html?vendor_id=12472
41. Samba PPP?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/2091/Samba-PPP.html?vendor_id=102
42. Ubuntu Developers Systemd?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/26277/Ubuntu-Developers-Systemd.html?vendor_id=12896
43. David King Vino?: CVE security vulnerabilities, versions and detailed reports. [online]. [Accessed 23 June 2015]. Available from: http://www.cvedetails.com/product/20740/David-King-Vino.html?vendor_id=11398
44. Linux Linux Kernel?: List of security vulnerabilities, [no date]. [online], [Accessed 5 June 2015]. Available from: http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2013/Linux-

Linux-Kernel.html

45. Linux Linux Kernel?: List of security vulnerabilities, [no date]. [online], [Accessed 3 June 2015]. Available from: http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2014/Linux-Linux-Kernel.html

46. Linux Linux Kernel?: List of security vulnerabilities, [no date]. [online], [Accessed 3 June 2015]. Available from: http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2015/Linux-Linux-Kernel.html

47. LibreSSL - FreeBSD Wiki, [no date]. [online], [Accessed 1 June 2015]. Available from: https://wiki.freebsd.org/LibreSSL

48. LibreSSL, [no date]. [online], [Accessed 1 June 2015]. Available from: http://www.libressl.org/

Anexos.

Aplicaciones		
Stage 1	**Stage 2**	**Stage 3**
app-admin/perl-cleaner	sys-apps/baselayout	app-arch/bzip2
app-admin/python-updater	app-arch/bzip2	app-arch/cpio
dev-lang/perl	app-arch/cpio	app-arch/gzip
dev-lang/python	app-arch/tar	app-arch/tar
app-shells/bash	app-shells/bash	app-arch/xz-utils
app-editors/nano	dev-lang/perl	app-i18n/man-pages-es
net-misc/rsync	dev-lang/python	app-shells/bash
net-misc/wget	net-misc/iputils	dev-perl/Locale-gettext
sys-kernel/linux-headers	net-misc/rsync	dev-util/pkgconfig
app-arch/bzip2	net-misc/wget	net-misc/iputils
app-arch/tar	sys-apps/coreutils	net-misc/openssh
app-arch/gzip	sys-apps/debianutils	net-misc/rsync
app-misc/pax-utils	sys-apps/diffutils	net-misc/wget
dev-libs/libxml2	sys-apps/file	sys-apps/baselayout
dev-libs/expat	sys-apps/findutils	sys-apps/busybox
dev-libs/popt	sys-apps/gawk	sys-apps/coreutils
virtual/editor	sys-apps/grep	sys-apps/diffutils
virtual/init	sys-apps/kbd	sys-apps/file
virtual/libiconv	sys-apps/net-tools	sys-apps/findutils
sys-devel/libtool	sys-apps/portage-2.0.51.22	sys-apps/gawk
sys-devel/automake	sys-process/procps	sys-apps/grep
sys-devel/autoconf	sys-process/psmisc	sys-apps/help2man
sys-devel/libperl	sys-apps/sed	sys-apps/kbd
sys-devel/gcc	sys-apps/shadow	sys-apps/man
sys-devel/gettext	sys-apps/texinfo	sys-apps/man-pages

Anexo 1: Aplicaciones en el sistema base de Nova.

```
#!/bin/bash
#set -x
VERSION=v0.12.4
WORKING_DIR=/tmp/system

#Getting node.js
#wget http://nodejs.org/dist/$VERSION/node-$VERSION-linux-x64.tar.gz

#creating file system structure
mkdir -pv $WORKING_DIR/{bin,etc,lib,sbin,proc,mnt,usr,sys,dev}
mkdir -pv $WORKING_DIR/dev/pts

#creating lib64 folder link
ln -svf lib $WORKING_DIR/lib64
ln -svf /bin $WORKING_DIR/usr/
ln -svf /sbin $WORKING_DIR/usr/

#Taken fron node.js README.md
tar --strip-components 1 -xzf node-$VERSION-linux-x64.tar.gz -C $WORKING_DIR

#Getting libraries needed by all binaries
for i in $(find $WORKING_DIR -type d -iname bin)
do
    for j in $(ls $i)
    do
        if [ ! -z "$(file $i/$j | grep 'ELF 64-bit LSB executable')" ]; then
            for k in $(ldd $i/$j | awk '{print $1}' );
            do
                LIB=$(find {,/usr}/lib{64,} | grep $k | head -1)
                cp -v $LIB $WORKING_DIR/lib64/
            done
        fi
    done
#Getting kernel modules for current system
KERNEL_VERSION=$(uname -r)

for i in $(lsmod | awk '{print $1}')
do
    MOD="$(find /lib/modules/$KERNEL_VERSION | grep /$i.ko)"
    DIR=$WORKING_DIR/"$(dirname $MOD)" 2> /dev/null
    mkdir -pv $DIR 2> /dev/null
    cp $MOD $DIR 2> /dev/null
done

cp /lib/modules/$KERNEL_VERSION/modules.* $WORKING_DIR/lib/modules/$KERNEL_VERSION/

#Getting busybox
#Uncomment this to get from internet site
#wget http://www.busybox.net/downloads/binaries/1.21.1/busybox-x86_64 -o $WORKING_DIR/bin/busybox

cp -v busybox $WORKING_DIR/bin/busybox
chmod +x $WORKING_DIR/bin/busybox

#Networking condiguration to be used on the new system's init script
NETWORK_INTERFACE=$(ip route get 8.8.8.8 | grep -Po '(?<=dev\s)[^\s]*')
NETWORK_IP=$(ip route get 8.8.8.8 | awk 'NR==1 {print $NF}')
NETWORK_MASK=$(ifconfig $NETWORK_INTERFACE | grep -Po '^(254|252|248|240|224|192|128)\.0\.0\.0|255\.(254|252|
248|240|224|192|128|0)\.0\.0|255\.255\.(254|252|248|240|224|192|128|0)\.0|255\.255\.255\.(254|252|248|240|224|
192|128|0)')
NETWORK_ROUTE=$(route | grep default | awk '{print $2}')

#Creating basic dev nodes
mknod -m 600 $WORKING_DIR/dev/console c 5 1
mknod -m 666 $WORKING_DIR/dev/null c 1 3
```

Anexo 2: Script que genera el sistema NMEPLBB.

```
#Getting hosts nameservers
cp /etc/resolv.conf $WORKING_DIR/etc

#Creating system's init script

cat > $WORKING_DIR/init << EOF
#!/bin/busybox sh

if test x"\$HAS_CTTY" != x"Yes"; then
        # initialise /proc and /sys
        busybox mount -t proc proc /proc
        busybox mount -t sysfs sys /sys
        # let busybox install all applets as symlinks
        busybox --install -s
        # spawn shells on tty 2 and 3 if debug or installer
        if test -n "\$DEBUG"; then
                # ensure they can open a controlling tty
                mknod /dev/tty c 5 0
                # create device nodes then spawn on them
                mknod /dev/tty2 c 4 2 && openvt
                mknod /dev/tty3 c 4 3 && openvt
        fi
        echo 0 0 0 0 > /proc/sys/kernel/printk
        # initialise /dev (first time)
        mkdir -p /dev/block
        echo /sbin/mdev > /proc/sys/kernel/hotplug
        mdev -s
        # re-run this script with a controlling tty
        exec env HAS_CTTY=Yes setsid cttyhack /bin/sh "\$0" "\$@"
fi

find /lib/modules/$KERNEL_VERSION | grep ko$ |
while read MOD;do
    modprobe \$MOD &
done
#Mount root partition
mount \$ROOT /mnt -o rw

ifconfig $NETWORK_INTERFACE $NETWORK_IP netmask $NETWORK_MASK up
route add default gw $NETWORK_ROUTE

#Run node server
cd \$(dirname \$INDEX)
node \$(basename \$INDEX)
EOF

chmod +x $WORKING_DIR/init

#Creating system
DATE=$(date +%s)
cd $WORKING_DIR
find . -print | cpio -o -H newc | gzip -9 > ../node-system-$DATE.cpio.gz

echo "Job done ensure to add the following lines to your grub configuration"
echo "linux /boot/vmlinuz-$KERNEL_VERSION ROOT=/dev/sda1 INDEX=/my/node/site/server.js"
echo "initrd /boot/node-system-$DATE.cpio.gz"

rm -rf $WORKING_DIR
```

Anexo 3: Script que genera el sistema NMEPLBB.

Glosario de Términos.

A

***Android*:** Sistema operativo desarrollado por Google y basado en el núcleo de Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o *tablets*: y también para relojes inteligentes, televisores, automóviles.

Aplicación informática: En informática, una aplicación es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo. Esto lo diferencia principalmente a otros tipos de programas, como los sistemas operativos.

B

Biblioteca de *software*: En informática, una biblioteca (del inglés *library*) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca. A diferencia de un programa ejecutable, el comportamiento que implementa una biblioteca no espera ser utilizado de forma autónoma, sino que su fin es ser utilizada por otros programas, independientes y de forma simultánea.

D

Distribución Linux: Una distribución Linux (coloquialmente llamada distro), es una distribución de *software* basada en el núcleo de Linux que incluye determinados paquetes de *software* para satisfacer las necesidades de un grupo específico de usuarios, dando así origen a ediciones domésticas, empresariales y para servidores. Por lo general están compuestas, total o mayoritariamente, de *software* libre, aunque a menudo incorporan aplicaciones o controladores propietarios.

F

***Fork*:** Una bifurcación (*fork* en inglés), en el ámbito del desarrollo de *software*, es la creación de un proyecto en una dirección distinta de la principal u oficial tomando el código fuente del proyecto ya existente. Comúnmente se utiliza el término inglés. Como resultado de la bifurcación se pueden llegar a generar proyectos diferentes que cubren necesidades distintas aunque similares. El término también puede ser usado para representar la ramificación de cualquier trabajo.

K

Kernel: En informática, un núcleo o kernel**,** es un *software* que constituye una parte fundamental del sistema operativo, y se define como la parte que se ejecuta en modo privilegiado (conocido también como modo núcleo). Es el principal responsable de facilitar a los distintos programas acceso seguro al *hardware* de la computadora o en forma básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.

S

***Software* libre:** Es la denominación de *software* que respeta la libertad de los usuarios que adquirieron el producto y, por lo tanto, una vez obtenido el mismo, puede ser usado, copiado, estudiado, modificado y redistribuido libremente de varias formas.