UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

INGENIERÍA EN COMPUTACIÓN

Prevención de delitos informáticos en sistemas Linux/Debian mediante la actualización automática de vulnerabilidades del sistema utilizando OVAL

Trabajo escrito bajo la modalidad de Alto nivel
Académico para obtener el título de

INGENIERO EN COMPUTACIÓN

PRESENTA:

ERIKA GLADYS DE LEÓN GUERRERO

DIRECTOR DE TESIS

M. en E. Imelda de la Luz Flores Díaz

UNAM, México, Febrero 2011.

TABLA DE CONTENIDO

AGRADECIMIENTOS

El presente trabajo es la culminación de una etapa académica, su desarrollo a pesar de que es presentado como resultado de un único autor, ha sido elaborado con el apoyo de muchas personas a las que quiero comenzar brindando un agradecimiento.

En primer lugar, gracias a Dios y a mis padres por permitirme llegar hasta el punto en el que me encuentro, por el apoyo incondicional que me han ofrecido, por el amor con el que he sido educada y por la dedicación que han tenido hacia mi desarrollo.

Gracias a mis hermanos por la comprensión recibida y por motivarme a ser un buen ejemplo para ustedes.

Un especial agradecimiento a Inés Gervacio por alentarme para cerrar cada proyecto que comience, por su alegría y al mismo tiempo seriedad con la que me ha motivado y por enseñarme que nunca es suficiente.

Gracias a mis amigos que me han apoyado y exhortado para la culminación del presente trabajo, cada uno ha colaborado de forma distinta, con revisiones, comentarios y recomendaciones, independientemente del apoyo emocional y entusiasmo ofrecido, gracias a Edgar, Dante, Israel, Tonatiuh, Jessica, Elizabeth, Jonathan, Javier y Jordan.

Gracias a los profesores que con comentarios, revisiones y asesorías colaboraron para le mejora del documento.

Gracias a la Subdirección de Seguridad de la Información UNAM-CERT por colaborar con un eslabón para mi formación en seguridad.

Y por último, a todos aquellos que me han ayudado en algún momento de mi vida. A todos ustedes ¡Gracias!.

INTRODUCCIÓN

El presente trabajo describe el desarrollo de una aplicación que tiene como objetivo la prevención de delitos informáticos a través de la actualización de parches en sistemas Linux/Debian basándose en esquemas OVAL (Open Vulnerability and Assessment Language).

Es importante visualizar la situación actual de los delitos informáticos, por un lado se tiene el desarrollo de aplicaciones e implementación de sistemas sin tener como prioridad la seguridad de los mismos, aun es complicado justificar una inversión de sistemas de seguridad a pesar de que la experiencia marca que es preferible tomar medidas preventivas a buscar soluciones después de los incidentes.

El desarrollo consta de 4 capítulos que a grandes rasgos se describen a continuación:

En el capítulo I se describe el marco teórico sobre el cual trabaja la aplicación, presentando elementos importantes para el desarrollo de la misma y definiendo la terminología elemental empleada a lo largo del documento lo que permite el entendimiento de los capítulos restantes.

En el capítulo II se muestra la forma de trabajo de OVAL, ya que la aplicación desarrollada está basada en esquemas publicados en sus repositorios, en este capítulo, se describe la estructura y funcionamiento de partes que conforman cada esquema y resalta la necesidad de una aplicación que realice las funciones de la aplicación desarrollada en este trabajo. Al describir la forma de trabajo deja una referencia para el desarrollo de distintas aplicaciones o mejora de las existentes basadas en OVAL, los detalles de este capítulo permiten el desarrollo del capítulo III ya que contempla las bases de la aplicación.

El capítulo III describe la implementación de la aplicación esquematizando su funcionamiento y explicando su desarrollo de acuerdo al ciclo de desarrollo del software. Deja la pauta para el desarrollo del capítulo final que implementa mejoras a la aplicación.

Finalmente, el capítulo IV muestra una visión a futuro de lo que podría llegar a ser la aplicación si se continúa con su desarrollo.

CAPITULO I

CONCEPTOS TEÓRICOS

1. CONCEPTOS TEÓRICOS

1.1 TERMINOLOGÍA BÁSICA

Seguridad informática: Conjunto de métodos y herramientas destinados a proteger la información y por ende, los sistemas informáticos ante cualquier amenaza.

Amenaza: Circunstancia que presenta el potencial para causar pérdida o daño.

Integridad: Es uno de los objetivos de la seguridad, indica la consistencia de datos.

Disponibilidad: Objetivo de seguridad que permite la respuesta oportuna de algún servicio o datos en el momento que se requieran

Autenticidad: Asegura que una entidad sea quien dice ser.

Confidencialidad: Permite mantener la información fuera del alcance de personas no autorizadas.

Políticas de seguridad: Documento que define el enfoque de seguridad de una empresa, establece una serie de normas que deben seguirse al aplicar la filosofía de seguridad de la empresa.

Ataque: Explotación de alguna vulnerabilidad presente en el sistema.

Vulnerabilidad: Debilidad en el sistema de seguridad que puede ser explotada causando perdida o daño.

Parche: Solución a agujeros de seguridad encontrados en las aplicaciones y sistemas con el fin de mantener la seguridad.

CERT: Computer Emergency Response Team. Equipo de respuesta a incidentes de seguridad en cómputo se encarga de proveer el servicio de respuesta a incidentes de seguridad a sitios que han sido víctimas de algún "ataque", así como de publicar información respecto a vulnerabilidades de seguridad, alertas de la misma índole y realizar investigaciones de la amplia área del cómputo y así ayudar a mejorar la seguridad de los sitios.

MITRE: Organización encargada de investigación de nuevas tecnologías y su uso, también administra distintas organizaciones de importancia internacional.

NIST: Instituto Nacional de Estándares y Tecnología, organización encargada de realizar gran parte de los estándares que rigen las tecnologías actuales.

NSA: National Security Agency**.** Agencia de inteligencia criptológica del Gobierno de los Estados Unidos, administrada como parte del Departamento de Defensa. Es responsable de obtener y analizar información transmitida por cualquier medio de comunicación, y de garantizar la seguridad de las comunicaciones del gobierno contra otras agencias similares de otros países, y que conlleva la utilización del criptoanálisis.

XML: Extensible Markup Language. Metalenguaje basado en etiquetas que permite definir gramática de lenguajes específicos.

CVE: Common Vulnerabilities and Exposures, código asignado a una vulnerabilidad que la identifica. Esta información es publicada en Internet por la corporación MITRE.

DNS: Domain Name System. Es un sistema encargado de traducir nombres inteligibles para los humanos en identificadores binarios asociados con los equipos conectados a la red, esto con el propósito de poder localizar y direccionar estos equipos mundialmente.

MD5: Message-Digest Algorithm 5, es un algoritmo de reducción criptográfico de 128 bits. Permite proporcionar integridad a los datos, ya que obtiene un valor único que cambia si los datos son modificados.

Exploit: Pieza de software que tiene como objetivo explotar una vulnerabilidad con el fin de causar un comportamiento no deseado e imprevisto.

Bugtrack: Proyecto que permite la notificación de errores y vulnerabilidades de forma centralizada.

1.2 DELITOS INFORMÁTICOS

Con el incremento de la tecnología, se ha dado también un incremento en los delitos informáticos, es por esto que surge la necesidad de poner especial atención en cuanto a la prevención y manejo de los incidentes.

Hay dos grupos que se encuentran generalmente vinculados con los delitos informáticos, por un lado los profesionales de las tecnologías de información, responsables de la primera línea de defensa y de descubrir cuando ocurren los mismos, por otro lado los profesionales de la defensa de la ley que son responsables de clasificar los problemas legales y jurisdiccionales con la intención de llevar a los cibercriminales ante un juicio.

En esta sección se hablará de forma muy general acerca de los delitos informáticos, la intención es brindar un panorama para notar la importancia de los mismos en la actualidad.

1.2.1 Definición

Es un poco controversial la definición de delito informático ya que se tienen distintas formas de ver el mismo.

Se puede comenzar por la definición de delito en general, delito es una acción antijurídica realizada por un ser humano, tipificado, culpable y sancionado por una pena.

Davara Rodríguez[1] define el Delito informático como, "la realización de una acción que, reuniendo las características que delimitan el concepto de delito, sea llevada a cabo utilizando un elemento informático y/o telemático, o vulnerando los derechos del titular de un elemento informático, ya sea hardware o software".

[1] DAVARA RODRÍGUEZ, Miguel Ángel. "De las Autopistas de la Información a la Sociedad Virtual", Editorial Aranzadi, 1996.

Julio Téllez Valdés[2] lo define como "actitudes ilícitas en que se tienen a las computadoras como instrumento o fin".

Parker[3] define a los delitos informáticos como "todo acto intencional asociado de una manera u otra a las computadoras; en los cuales la víctima ha o habría podido sufrir una pérdida; y cuyo autor ha o habría podido obtener un beneficio"

Pero a todo esto, hay elementos en común que permiten ver que los delitos informáticos hacen referencia a delitos cometidos utilizando computadoras, las computadoras pueden verse involucrados de diferentes formas en el crimen:

- La computadora puede ser el objetivo de crimen, es decir la victima.
- La computadora puede ser usada para propósitos incidentales relacionados con un crimen, por ejemplo guardar los archivos de venta de drogas ilegales.
- La computadora puede ser la herramienta del crimen, usada para cometer el crimen.

1.2.2 Estado actual de los delitos informáticos

De acuerdo a la situación actual que vive México, existen pocas leyes encargadas de atender delitos informáticos.

[2] TELLEZ VALDÉS, Julio. "Los Delitos informáticos. Situación en México", Informática y Derecho Nº 9, 10 y 11, UNED, Centro Regional de Extremadura,Mérida, 1996.

[3] PARKER, D.B, Citado por Romeo Casabona Carlos M. Poder Informático y Seguridad Jurídica.

A pesar de vivir en un mundo rodeado de información y tecnologías y el incremento que éstas van teniendo con el paso del tiempo, no existe una legislación especializada en delitos informáticos que se encargue de vigilar la seguridad informática.

Es posible notar que al hablar de delitos informáticos se hace referencia a delitos que no tiene límites geográficos, como es el caso de otro tipo de delitos tipificados en las leyes de cada nación, debido a que la información viaja entre países, surge la necesidad de una legislación universal, especializada en delitos informáticos, la cual sea aplicada indistintamente del país en el que se cometa un crimen.

Debido a que no se tienen leyes adecuadas aun se debe tener especial cuidado con la forma en la que se protegen los sistemas ya que de no ser así se podría ser víctima de delincuentes informáticos que están tratando de hacer uso inadecuado de las tecnologías de información, como se ha expresado anteriormente existen diferentes formas de emplear una computadora para cometer delitos informáticos, el hecho de no manejar información sensible en una computadora no quiere decir que se esté libre de ser víctima de algún delito informático, el equipo manejado puede ser víctima de personas mal intencionadas, que mediante equipos victimas cometan de forma remota delitos, es debido a eso que a pesar de no tener información sensible se debe proteger el acceso y tomar en cuenta las medidas preventivas para evitar intrusiones.

El uso de las tecnologías de información es actualmente uno de los mayores recursos para todo tipo de negocios en todo tipo de situaciones, se ven en todos lados computadoras que controlan distintos sistemas, esto hace más vulnerable a la sociedad ya que un ataque a un sistema importante puede causar grandes estragos.

El crecimiento de la tecnología de información no va ligada a la implementación de medidas de seguridad que proporcionen protección, sin embargo su

crecimiento está relacionado con el aumento de los delitos informáticos. Esto hace necesario la protección individual de los sistemas.

En ocasiones puede resultar molesto implementar tantas medidas de seguridad ya que restan comodidad al personal, o puede ser algo incómodo, simplemente se puede hacer referencia a la tendencia del ser humano de oponerse al cambio, aun así es necesario que este tipo de medidas sean establecidas para evitar intrusión.

No hay que confiarse de la seguridad que viene incluida con algunos sistemas o aplicaciones, ya que no siempre es la óptima, los fabricantes suelen tener la tendencia a la funcionalidad de sistemas y no a la seguridad.

1.3 SISTEMAS DE SEGURIDAD

Los delitos informáticos se llevan a cabo debido a que las computadoras y las redes no poseen medidas de seguridad adecuadas, los cibercriminales tienen la tendencia, como sucede en el "mundo real", de elegir víctimas fáciles de atacar.

Los puntos débiles encontrados en un sistema y que son comunes entre la mayoría de los equipos, es decir vulnerabilidades detectadas en determinada aplicación, errores de configuración que dejan huecos importantes de seguridad, etc., son fácilmente atacados por individuos muchas veces con poca experiencia informática, sirviéndose de herramientas publicadas en la red o exploits difundidos, esta situación podría ser evitada en ocasiones con medidas preventivas fáciles de establecer como aplicación de parches y modificación de la configuración en los equipos.

Para mantener un sistema informático "seguro" es necesario la ejecución de todo un proceso, se debe aclarar que las medidas tomadas en este punto no aislarán al sistema completamente de un ataque, se disuade al atacante dificultando la realización del ataque, pero existiendo el tiempo y los recursos necesarios cualquier sistema puede ser comprometido.

A pesar de lo anteriormente citado es recomendable la implementación de un sistema de seguridad adecuado que disminuya el riesgo.

La implementación de un sistema de seguridad requiere un conjunto de elementos que al converger cumplirán con el objetivo del sistema: disminuir el riesgo y cumplir con los elementos de seguridad, autenticidad, confidencialidad, integridad y disponibilidad.

1.3.1 Elementos de un sistema de seguridad

Los elementos del sistema pueden variar dependiendo de muchos factores como el propósito del sistema, el sistema operativo instalado, aplicaciones presentes, datos manejados, estructura física, la arquitectura de red, los protocolos empleados, etc., No existe un conjunto de elementos estático que defina un sistema de seguridad, es debido a esto que en esta sección se describe de forma general los elementos comúnmente usados, pero se hace la aclaración de que estos elementos pueden variar entre sistemas.

Algunos de los elementos de un sistema de seguridad pueden ser los siguientes:

- *Criptografía:* Es uno de los elementos más importantes ya que permite "asegurar" distintas partes de un sistema, pudiendo ser, canales de comunicación, archivos, correo electrónico, etc. El objetivo es mantener fuera del alcance del atacante la información de forma entendible, es decir, la información es manipulada de cierta forma para permanecer ilegible ante un usuario no autorizado.

- *Sistemas antivirus*: Permite localizar distintos tipos de malware, previene infección y la propagación.

- *Firewall:* Es un tipo de filtro de comunicaciones, bloquea el acceso no autorizado de acuerdo a políticas y normas previamente establecidas. Es decir evalúa, los paquetes que cumplan con las reglas establecidas para permitir el acceso.

- *Monitor de trafico:* Permite verificar el correcto y normal funcionamiento de una red, evaluando paquetes entrantes y salientes y poniendo mayor atención en aquellos que muestran comportamiento anormal, los monitores a su vez pueden estar acompañados de alarmas que indiquen algún riesgo presente en el sistema. Existen herramientas automatizadas que realizan este trabajo, aunque se hace necesario contar con la vigilancia de un experto humano para descartar los falsos positivos.

- *Contraseñas fuertes:* Cuando el control de acceso se da mediante contraseñas la protección del sistema se encuentra basado en la fortaleza de las mismas, es necesario que se implanten políticas estrictas en cuanto a la generación, almacenamiento y cambio de contraseñas impidiendo contraseñas débiles con pocos caracteres, con palabras de diccionario y sin el uso de caracteres especiales, para esta finalidad existen herramientas automáticas que impiden asignar contraseñas débiles y periódicamente exigen el cambio de las mismas.

- *Control de acceso a datos sensibles:* Es preferible mantener múltiples mecanismos para controlar el acceso, mas hablando de datos sensibles, puede ser mediante una arquitectura de seguridad que cuente con varias capas o con distintos mecanismos de verificación de identidad, pudiendo ser algo que se tiene como es el caso de tarjetas de acceso, identificaciones, etc., algo que se sabe, pudiendo ser contraseñas o códigos de acceso o algo que se es, hablando aquí de mecanismos biométricos, lectores de huellas digitales, reconocimiento de voz, etc.

- *Instalación de actualizaciones y parches:* Es un punto muy importante para la seguridad de un equipo debido a que existen instituciones[4] que día a día encuentran nuevas vulnerabilidades e implementan las soluciones liberando parches y actualizaciones como mecanismo de prevención, gran parte de los ataques son realizados explotando las vulnerabilidades publicadas debido a que no se instalan los parches y actualizaciones a tiempo.

- *Configuración adecuada del sistema:* La mayor parte de los sistemas se basan en la funcionalidad dejando a un lado la seguridad de los datos manejados, debido a esto, las configuraciones establecidas por default no son las más adecuadas desde el punto de vista de la protección de los datos, como consecuencia es necesario adecuar la configuración del sistema priorizando la seguridad de la información.

- *Respaldos periódicos:* Se han dado grades perdidas de información por ignorar esta parte del sistema de seguridad, la pérdida de datos se puede dar debido a diversas razones entre ellas fenómenos naturales, comportamientos humanos inadecuados como venganza de empleados inconformes causando daño a la información, accidentes como daño a dispositivos de almacenamiento por derramamiento de líquidos, etc. la información se encuentra expuesta y fácilmente podría perderse, es por esto que surge la necesidad de respaldar la información constantemente, y almacenar los respaldos en distinta ubicación geográfica y con el control de acceso a los medios adecuado. Existen diversas metodologías para respaldar información, cada sistema tiene necesidades distintas por lo que el mecanismo de respaldo debe ser adecuado, este proceso puede ser de forma automatizada[5].

[4] Dentro de las que se encuentran: CERT: Computer Emergency Response Team, Bugtrack, OVAL por mencionar solo algunas.

[5] Existen distintos tipos de respaldos dependiendo de las necesidades de la institución, completos, incrementales, diferenciales, esta información puede consultarse en: http://www.backup4all.com/kb/backup-types-115.html

- *Aislamiento de datos sensibles:* Obviamente existen datos que son más sensibles que otros, es con este tipo de datos que se debe tener especial cuidado, existen distintas formas de proteger la información, entre ellas la creación de zonas especiales para este tipo de datos dejándolos fuera del alcance de personas no autorizadas. Otra solución para la protección de este tipo de información puede ser el uso de criptografía.

- *Configuración segura de navegadores:* Es un punto importante a considerar ya que existen numerosas vulnerabilidades si no se toma especial cuidado con los navegadores, como punto importante a mencionar es la existencia de inyección de código malicioso en aplicaciones web, si no se tiene cuidado con el filtrado de este tipo de código el sistema puede resultar comprometido.

- *Establecimiento de políticas de seguridad:* Las políticas de seguridad son un punto muy importante para las empresas, deben ser definidas de acuerdo a las necesidades de la organización y en base a sus objetivos. Estas políticas deben ser claras y precisas y deben ser dadas a conocer a todos los individuos de la organización, estas políticas definirán lo que se debe hacer y lo que está prohibido.

1.3.2 Medidas preventivas

A continuación se expresarán algunas medidas que se deben tomar en cuenta para mantener la seguridad del sistema, como se ha mencionado anteriormente depende de los objetivos de la organización el nivel de seguridad que se debe implementar.

- Establecer permisos de acceso individuales a los elementos de la red, restringiendo así el acceso a personas inadecuadas.
- Cifrado de los datos enviados por la red o bien almacenados en los discos para proteger los datos especialmente valiosos, importantes o confidenciales.

- Elementos de red situados en cuartos con control de acceso.
- Firewalls en los puntos de entrada de la red, sirviendo como protección perimetral.
- Protección a los servidores controlando el acceso mediante distintos mecanismos de autenticación.
- Controlar el acceso físico a los elementos del sistema.
- Tener especial cuidado con las estaciones de trabajo que disponen acceso a los servidores a través de la red.
- Proteger los dispositivos de red de acceso no autorizado, estos dispositivos incluyen hubs, switches, routers, cables de red, etc.
- Tener especial cuidado con equipos portátiles y dispositivos móviles, cuidando la información que se transporte en ellos.
- Manejar adecuadamente la información impresa, cuidando el acceso y la destrucción de la misma.
- Hacer uso de algoritmos de firmas para proporcionar autenticación.
- Hacer uso de algoritmos Hash para probar la integridad de los datos.
- Establecer distintas formas de autenticación incluyendo dispositivos biométricos.
- Que hacer uso de certificados digitales confiando así en una tercera parte verificadora.
- Establecer reglas adecuadas para los firewalls, para que se dé de forma correcta el filtrado de paquetes.
- De establecer mecanismos de detección intrusos con alarmas que indiquen la existencia de riesgos.
- Formar un equipo de respuesta a incidentes.
- Establecer y aplicar políticas de seguridad adecuadas.
- Definir puntualmente responsabilidades a cada uno de los integrantes del plan de seguridad.
- Asignar niveles de amenazas.
- Analizan puntos débiles de la organización y la red.
- Cumplir con estándares de seguridad.

- Establecer políticas adecuadas de contraseñas, manteniendo longitud y complejidad adecuada.
- Instalación de software antivirus.
- Desactivar los archivos e impresoras compartidas.
- Desactivar servicios innecesarios.
- Configurar auditoría del sistema.
- Establecer seguridad adecuada para los navegadores y correo electrónico, tener especial cuidado con métodos de incrustación de código.
- Mantener al día los parches y actualizaciones de seguridad.
- Manejar de forma adecuada las cookies.
- Exigir direcciones MAC específicas para las conexiones de red.
- Cifrar información especialmente cuando se trate de redes inalámbricas.

Esta es sólo parte de las medidas preventivas que se deben tener para evitar incidentes de seguridad.

1.4 VULNERABILIDADES

Las vulnerabilidades representan un punto muy importante a considerar ya que permiten tomar las medidas adecuadas y permiten prever posibles ataques.

Es importante realizar análisis de vulnerabilidades constantes en la búsqueda de nuevos huecos de seguridad, de esta forma se mantendrá más protegido el sistema.

1.4.1 Definición

Se puede definir vulnerabilidad como un "fallo de implementación de software, hardware o de diseño que lleva a la desprotección de sistemas de información"[6]

Es una debilidad en el sistema de información o sus componentes, ésta puede ser explotada produciendo un daño.

También puede ser definida como la presencia de debilidades, o errores en el diseño o implementación, pueden causar comportamientos inesperados e indeseables comprometiendo la seguridad del sistema, red, aplicación o protocolo involucrado.

Hay que diferenciar el concepto de vulnerabilidad del concepto de amenaza, una vulnerabilidad puede estar presente sin causar algún daño si no es explotada, la amenaza se encuentra a la espera de vulnerabilidades para poder causar daño.

Es importante localizar las vulnerabilidades presentes en un sistema y una vez localizadas a tomar medidas adecuadas para poder mitigarlas, no todas las vulnerabilidades son mitigables, algunas sólo se controlan pero seguirán presentes.

En todo proceso que involucra asegurar un sistema, es necesario la identificación de vulnerabilidades, de esta forma se podrá implementar los mecanismos necesarios para cumplir los objetivos de seguridad.

[6] Victor Oppleman. *Extreme Exploits.* Ed. ANAYA. 2005.

1.4.2 Tipos

Las vulnerabilidades se pueden clasificar definiendo en donde están presentes, se puede tener la siguiente clasificación:

- Vulnerabilidades en hardware
- Vulnerabilidades en software
- Vulnerabilidades en datos

La clasificación de vulnerabilidades está ligada a la presencia de amenazas y ataques, de acuerdo a la clasificación anterior a las amenazas y ataques que pueden afectar cada tipo de vulnerabilidad son los siguientes:

Amenazas y ataques de hardware:

- Daños con agua
- Daños con fuego
- Daños con gas
- Electrocución
- Derramar comida o agua en dispositivos
- Daños en los cables
- Daños por partículas de polvo o humo de cigarro
- Daños por maltrato físico
- Robo de dispositivos
- Daño por fenómenos naturales

Las vulnerabilidades en hardware son las más difíciles de eliminar, sin embargo es posible disminuir el riesgo limitado el acceso a personas no autorizadas, estableciendo políticas que impidan consumir alimentos y bebidas cerca de los equipos, realizar respaldos periódicos y almacenándolos en lugares seguros y en distinta ubicación geográfica.

Amenazas y ataques de software:

- Modificación
- Permitir acceso sin autorización
- Construir bombas lógicas
- Incluir caballos de Troya
- Incluir virus
- Abrir puertas traseras
- Realizar copia de datos
- Borrado
- Cambio de ubicación
- Acceso sin autorización
- Cambio de configuración
- Copia no autorizada

Las vulnerabilidades en software son el principal objetivo que cubren las definiciones OVAL, están incluidas generalmente en aplicaciones creando huecos de seguridad.

Amenazas y ataques en datos:

- Modificación

Las vulnerabilidades en los datos son de gran importancia, ya que éstos sirven de entrada a distintas aplicaciones y son el principal objetivo de protección de la seguridad, pueden provocar grandes daños y confusiones en la salida.

Los datos pueden estar de forma impresa y representan un gran peligro si no son manejados de forma adecuada.

1.4.3 Ejemplo

En esta sección se analizará una vulnerabilidad encontrada en los esquemas de definición publicados por OVAL, (esquema en el que se basa la aplicación desarrollada en este trabajo y definida en el siguiente capítulo) con la finalidad de unir varios elementos aquí definidos de una forma concreta y demostrativa.

La siguiente imagen muestra una parte del esquema de definiciones.

```
<platform>Debian GNU/Linux 4.0</platform>
<platform>Debian GNU/Linux 4.0</platform>
<product>eggdrop</product>
</afected>
<reference source="DSA"
ref_url=http://www.debian.org/security/2008/dsa-1448 ref_id="DSA-
1448"/>
<description>
It was discovered that eggdrop, an advanced IRC robot, was
vulnerable to a buffer overflow which could result in a remote user
executing arbitrary code.
</description>
```

Se muestra la plataforma y versión afectada, el producto o aplicación que genera la vulnerabilidad y la referencia hacia el parche correspondiente, también se muestra una breve descripción de la vulnerabilidad.

Dirigiéndose a la referencia se encuentra la siguiente información:

DSA-1448-1 eggdrop -- buffer overflow

Date Reported:
05 Jan 2008
Affected Packages:
eggdrop
Vulnerable:
Yes
Security database references:
In the Debian bugtracking system: Bug 427157.
In Mitre's CVE dictionary: CVE-2007-2807.

Fecha en la que fue reportada la vulnerabilidad, paquetes afectados, pero lo más importante en este caso es la referencia hacia el correspondiente CVE y hacia Bugtrack.

El CVE contiene la siguiente información:

CVE-ID	
CVE-2007-2807 (under review)	Learn more at National Vulnerability Database (NVD) • Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings
Description	
Stack-based buffer overflow in mod/server.mod/servrmsg.c in Eggdrop 1.6.18, and possibly earlier, allows user-assisted, remote IRC servers to execute arbitrary code via a long private message.	
References	
Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.	
• MISC:http://www.eggheads.org/bugzilla/show_bug.cgi?id=462 • CONFIRM:http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=427157 • DEBIAN:DSA-1448 • URL:http://www.debian.org/security/2008/dsa-1448 • DEBIAN:DSA-1826 • URL:http://www.debian.org/security/2009/dsa-1826 • FEDORA:FEDORA-2007-4305 • URL:https://www.redhat.com/archives/fedora-package-announce/2007-December/msg00336.html • FEDORA:FEDORA-2007-4325 • URL:https://www.redhat.com/archives/fedora-package-announce/2007-December/msg00348.html • GENTOO:GLSA-200709-07	

Es así como se puede obtener información más concreta sobre lo que la vulnerabilidad provoca, en este caso origina un problema llamado buffer overflow basado en la pila que se produce por una aplicación llamada Eggdrop en la versión 1.6.18, al explotar esta vulnerabilidad es posible ejecutar código a través de mensajes enviados por un canal IRC, almacenando código malicioso en segmentos de pila y ejecutándolos posteriormente.

El CVE también brinda mayor referencia para un conocimiento exacto de la vulnerabilidad.

Por otra parte la referencia que brinda el esquema de definición OVAL muestra referencia hacia bugtrack:

Debian Bug report logs - #427157

CVE-2007-2807: stack-based buffer overflow

Package: eggdrop; Maintainer for eggdrop is gpastore@debian.org (Guilherme de S. Pastore); Source for eggdrop is src:eggdrop.

Reported by: Florian Weimer <fw@deneb.enyo.de>
Date: Sat, 2 Jun 2007 07:51:01 UTC
Severity: *grave*
Tags: security
Found in version eggdrop/1.6.18-1
Fixed in version eggdrop/1.6.18-1.1
Done: Nico Golde <nion@debian.org>
Bug is archived. No further changes may be made.

En esta referencia se muestra mayor información sobre la vulnerabilidad.

1.5 PARCHES

Una de las partes más importantes dentro de este trabajo son los parches ya que permitirán actualizar el sistema evitando fallas de seguridad, por esta razón es necesario tener una idea clara del concepto de parches.

1.5.1 Definición

Un parche es una actualización de un programa utilizada para la solución de problemas o mejoras en la aplicación sin necesidad de cambiar de versión. La aplicación de los parches es dirigida hacia cualquier programa desde aplicaciones hasta el sistema operativo.

Se puede considerar como un rápido y a veces "sucio" remedio a errores y vulnerabilidades en el software.

Los parches pueden ser aplicados tanto a código ejecutable como a código fuente, por ejemplo si la aplicación ha sido distribuida, el parche puede ser publicado vía internet y cada usuario puede aplicar el parche a su correspondiente aplicación, en cambio si los productores de la aplicación quieren modificar el producto y hacer una nueva distribución, pueden recurrir a la aplicación directa en el código fuente.

En algunas fuentes de información se dice que es emplea la palabra parche por la aplicación creada por el lingüista Larry Wall (creador del lenguaje de programación Perl) llamada patch que permitía la instalación automática de actualizaciones a sistemas UNIX[7].

[7] http://en.wikipedia.org/wiki/Patch_(Unix)

1.5.2 Importancia de los parches

El empleo de parches como mecanismo de prevención de ataques es de gran importancia para la seguridad de los sistemas ya que gran parte de los ataques está basada en la explotación de las vulnerabilidades en aplicaciones y sistemas operativos que son publicadas en internet al mismo tiempo que se publican exploits que hacen uso de ellas por lo que para los individuos con malas intenciones es muy fácil identificar las aplicaciones instaladas que contienen alguna vulnerabilidad, ejecutar el exploit correspondiente y así hacer uso indebido de los sistemas.

La oportuna instalación de parches previene ataques y ayuda a mantener actualizado el sistema.

1.5.3 Tipos de parches

En General que existen tres tipos de parches de acuerdo a su uso:

Parche de seguridad

Permiten cubrir agujeros de seguridad, no modifican las funciones del programa. Este es el tipo de parche que se manejará a lo largo del desarrollo de la aplicación.

Parche de actualización

Permite la modificación algunas veces de la funcionalidad de la aplicación, añadiendo nuevas herramientas, este tipo de parches también permite mejorar la aplicación, disminuyendo tiempos, el procesamiento, aplicando algoritmos más eficientes para la realización de alguna tarea, eliminando código muerto, etc.

Parches de depuración

Permite la corrección de errores de programación ya sea porque se trata de una versión beta de la aplicación o porque una vez publicada la actualización se han encontrado errores que es necesario corregir.

Dentro de los parches de depuración también se encuentran parches que cubren agujeros de seguridad, corrigiendo algún error de programación que cause vulnerabilidad.

1.5.4 El papel de los parches dentro de la prevención de delitos informáticos

Es importante tener en cuenta la reacción que tendrá el sistema con la aplicación de un parche ya que en ocasiones resulta contradictorio el efecto de la aplicación, debido a los parches son realizados de una forma rápida puede ser que no estén bien programados o que no midan las consecuencias después de su aplicación y causen más vulnerabilidades de las que cubren.

Los parches deben aplicarse en ambientes de prueba, nunca dentro de un sistema en operación, esto se hace para medir el efecto y evitar desastres.

Los parches aplicados de manera correcta pueden prevenir ataques, pero hay que tener cuidado con la publicación de los mismos ya que pueden provenir de fuentes con intenciones maliciosas.

1.6 CVE (Common Vulnerabilities and Exposures)

Debido a que OVAL se basa en CVE para definir las vulnerabilidades dentro de sus esquemas, es necesario tener una idea clara sobre lo que es y su importancia.

1.6.1 Definición

CVE (Common Vulnerabilities and Exposures) es un identificador que permite la estandarización al referirse a las vulnerabilidades ya que les asigna una clave única que es manejada por distintas aplicaciones manteniendo interoperabilidad, las vulnerabilidades manejadas con CVE son publicadas en internet con una descripción detallada sobre su comportamiento, datos en general y ligas a paginas que hablan más de la vulnerabilidad[8].

CVE permite conocer públicamente las vulnerabilidades de seguridad, proporcionando una numeración común de configuración (CCE) identificando problemas de configuración de seguridad y exposiciones.

CVE facilita compartir datos con otras bases de datos y herramientas teniendo acceso rápidamente a la descripción de la vulnerabilidad referida.

El identificador CVE es de libre descarga permitiendo a cualquier herramienta hacer referencia a ellos.

1.6.2 Lista CVE

Al descubrir una nueva vulnerabilidad de seguridad, se le asigna un identificador candidato CVE por el que se vota para convertirse o no en una nueva entrada en la lista CVE, si es rechazado, se publican los motivos en el sitio web CVE; sí es aceptada se une a la lista CVE.

[8] http://cve.mitre.org/

La lista CVE se distribuye de forma gratuita en la página http://cve.mitre.org/cve/index.html, puede ser copiada, redistribuida, referenciada, analizada pero no modificada.

1.6.3 Importancia

CVE surge en 1999 para unificar los nombres de las vulnerabilidades, hasta ese momento la referencia hecha hacia las vulnerabilidades era manejada de forma distinta por cada herramienta, esto implica que no existían una compatibilidad.

CVE homologa los nombres de las vulnerabilidades, constituyendo un punto de referencia para el intercambio de datos con el fin de que los productos de seguridad y los servicios pueden hablar entre sí.

CVE también permite la evaluación de cobertura de herramientas y servicios proporcionando una línea base identificando las herramientas más eficaces y adecuadas a las necesidades de cada organización.

En el presente capitulo se ha definido de manera general conceptos de seguridad con el fin de introducir al lector a la temática del presente escrito, lo que permite visualizar de manera más clara el capitulo número II donde se explicará la base del sistema.

CAPITULO II

OVAL

2. OVAL

2.1 ¿QUÉ ES OVAL?

Open Vulnerability and Assessment Language es una comunidad internacional de seguridad informática creada por el US-CERT (Computer Emergency Response Team), tiene la función de publicar y difundir de forma estandarizada documentos que contienen definiciones para la evaluación de sistemas, implicando, la presencia de vulnerabilidades y configuraciones inadecuadas que causen compromiso, las aplicaciones y versiones instaladas y los parches aplicables según las características definidas; independientemente a estas publicaciones, la comunidad OVAL tiene un foro en el que se debaten temas periféricos al área y periódicamente realiza un boletín que envía de forma gratuita a sus suscriptores.

OVAL maneja un lenguaje para estandarizar la transferencia de información y permite la publicación de definiciones en repositorios administrados por la comunidad OVAL, este lenguaje está basado en XML (Extensible Markup Language) que facilita el manejo de información.

El lenguaje OVAL maneja tres principales esquemas que son los siguientes:

- Esquema de características del sistema
- Esquema de definición
- Esquema de resultados

Cada uno de los anteriores cumple un papel importante dentro del análisis de los equipos, estos esquemas son independientes entre sí, es decir se pueden utilizar separados de acuerdo a las necesidades de quien haga uso de ellos.

La recopilación de la información contenida en los diversos esquemas está dada por un conjunto de colaboradores entre los que se encuentran:

- Industria
- Academia
- Organizaciones de gobierno

- Vendedores de sistemas operativos
- Vendedores de herramientas de seguridad
- Instituciones de investigación

Este conjunto de personas es el encargado de la actualización y creación de nuevas versiones de esquemas con la finalidad de brindar a la comunidad información de utilidad para la protección de los sistemas.

La actualización de los esquemas no solo están en manos de este grupo de personas, OVAL es una comunidad abierta que permite la colaboración de cualquier persona que desee realizar aportaciones, es decir, teniendo el caso de que alguien externo al grupo de desarrollo encuentre alguna vulnerabilidad en algún producto de software puede reportarla al vendedor correspondiente, el cual a su vez hará el reporte a la comunidad OVAL para la actualización de esquemas.

El propósito de este lenguaje es poner a disposición del público elementos de seguridad que son útiles para la creación de nuevas herramientas y para la evaluación de equipos, es decir, es la base de muchas herramientas que realizan diversas funciones pero tienen sus principios en algún esquema OVAL.

Los esquemas publicados son libres y permiten a cualquier persona hacer uso de ellos ya sea del conjunto (Esquema de características del sistema, Esquema de definición, Esquema de resultados) o de alguno de ellos por separado, puede ser, para la evaluación de equipos usando el interprete OVAL o para la creación de nuevas herramientas.

OVAL maneja también un intérprete del lenguaje que permite hacer uso de los tres esquemas publicados, realizando el siguiente proceso:

- Verifica el esquema de definición correspondiente al sistema a evaluar.
- Verifica las características del sistema en análisis.
- Compara los resultados de los puntos anteriores generando reportes con los resultados obtenidos. Exponiendo las vulnerabilidades, parches y configuración existentes.

2.2 CARACTERÍSTICAS

OVAL presenta diversas características que se muestran como ventajas por lo que se ha convertido en base de muchas herramientas y por lo que la comunidad que lo integra ha ido en crecimiento. Estas son algunas características propias de OVAL:

- OVAL es soportado por una comunidad de expertos y por instituciones de gran importancia a nivel internacional, por lo que se le puede considerar confiable.
- OVAL es escrito en XML facilitando la localización y tratamiento de información.
- Permite identificar fácilmente si una vulnerabilidad se encuentra presente en un sistema.
- Interoperabilidad con otros productos que manejan lenguaje XML.
- Maneja distintas versiones en las que se realizan las actualizaciones.
- Publica los esquemas en un repositorio disponible en Internet.
- Es libre.
- Independencia entre esquemas.
- Es múltiple el uso que se puede dar a los distintos esquemas manejados.
- Debido a que la comunidad que lo integra es amplia, la actualización de los esquemas es frecuente.
- Maneja definiciones clasificadas de acuerdo a distintos criterios por lo que la cantidad de información manejada puede disminuir.

2.3 FORMA DE TRABAJO DE OVAL

El lenguaje OVAL maneja tres principales categorías:

- *Esquema de características:* Permite concentrar las características de configuración del equipo a analizar.

- *Esquema de definición:* Permite detectar el estado de la máquina, la presencia y ausencia de vulnerabilidades, parches, etc.

- *Esquema de resultados*: Permite reportar los resultados obtenidos del análisis.

El proceso estándar de evaluación trabaja de la siguiente forma:

1. Avisos y políticas de seguridad. Publicación de avisos de seguridad por parte de los vendedores e instituciones encargadas de la actualización de vulnerabilidades y amenazas. Desarrollo por parte de agencias especializadas como NIST[9] y NSA[10] de políticas de configuración.

2. Generación de definiciones. Codificación de los avisos y políticas mencionados en el paso anterior de tal forma que queden expresadas como definiciones OVAL. Estas definiciones están compuestas de dos partes, una parte principal que describe las características del formato y partes secundarias individuales que separan las diferentes plataformas de sistemas operativos y aplicaciones.

3. Coleccionar datos del equipo a evaluar. De acuerdo al esquema de características, se recopila la información necesaria del equipo al cual se le aplicara la prueba. Esta información queda plasmada en un documento con formato XML para ser compatible con OVAL. Se incluyen, parámetros de sistema operativo, características del software de aplicación instalado, y distintas características de configuración concernientes a la seguridad del equipo.

[9] Definido en terminología básica. Punto 1.1
[10] Definido en terminología básica. Punto 1.1

4. Análisis OVAL. Comparación entre las características recopiladas del sistema a evaluar definidas en el paso 3 y las definiciones codificadas en OVAL expresadas en el paso 2.

5. Resultados OVAL. Los resultados de la comparación del paso anterior definen si un equipo es vulnerable o no, mostrando los resultados en un formato especifico. En este paso se hace uso del esquema de resultados que facilita la incorporación a otras herramientas ya que lo codifica en un formato estándar.

2.4 FASES DE ANÁLISIS OVAL

A continuación se explican las principales fases para llevar a cabo un análisis completo con los esquemas OVAL, como se ha mencionado, OVAL es una base para la creación de herramientas, depende de cada programador y diseñador las fases de análisis requeridas en herramientas especificas, aquí se muestra el análisis tradicional haciendo uso de los tres esquemas.

2.4.1 Verificar configuración y estado del sistema

El esquema de características OVAL permite obtener la información necesaria del equipo a evaluar en un formato compatible con los elementos de comparación para así trabajar de forma homogénea entre esquemas y entre otras aplicaciones.

El objetivo de esta fase es conocer las propiedades del equipo a evaluar indicando puntualmente los aspectos que influyen en la seguridad y el adecuado desempeño, pueden listarse las siguientes características:

- Sistema operativo instalado
- Versión de sistema operativo
- Aplicaciones instaladas
- Versiones de las aplicaciones instaladas
- Configuración del sistema
- Parches instalados

La información obtenida en esta fase puede verse independiente como un inventario del sistema, identificando su estado y puede servir de referencia para un control de modificaciones o como base para comparaciones entre el estado anterior y actual del sistema; son distintas las aplicaciones que hacen uso de este esquema de forma separada al análisis completo que incluye los tres esquemas.

De esta fase se obtiene el punto de comparación con la información obtenida con los esquemas de definición para extraer los puntos vulnerables del sistema.

2.4.2 Análisis del sistema

Los esquemas de definición manejados por OVAL se encuentran estructurados de tal forma que pueden ser manejados desde lo general a lo específico anulando así rápidamente características que no conciernan al análisis activo.

En esta fase convergen todos los esquemas, es la fase más importante sí se realiza de forma completa el proceso de análisis por medio de los tres esquemas.

Consiste en la comparación entre el esquema de definición que se obtiene directamente del repositorio OVAL y el esquema de características obtenido a partir de la evaluación del equipo, es así como se genera el esquema de resultados que identifica detalladamente cada amenaza en el sistema describiendo el problema a través de referencias que permiten obtener ideas más claras y fundamentadas de cada vulnerabilidad o fallo en la configuración, incluyendo también referencias a la ubicación de los parches correspondientes en caso de que ya hayan sido publicados.

La comparación que se realiza en esta etapa se facilita debido a la compatibilidad entre formatos de esquemas, aunado a esto, OVAL estructura los esquemas de forma que se identifiquen fácilmente las partes aplicables, esto aumenta la velocidad de análisis ya que en caso de ocupar un esquema completo de definición que incluya todos los sistemas operativos y versiones existentes, cuenta con secciones que invalidan partes del esquema que no apliquen al test, dejando solo las características compartidas entre el esquema de definición y el esquema de características, este resultado es el que aplicando un formato específico genera el esquema de resultados.

2.4.3 Reporte de estado del sistema

OVAL brinda independencia entre esquemas, esto da una gran flexibilidad a los creadores de herramientas ya que permite integración y combinación de esquemas; el Reporte del estado del sistema es definido en formato estándar XML, este documento contiene el resultado de la comparación entre el esquema de configuración y de definición, manifiesta el estado de configuración del sistema evaluado, mencionando las vulnerabilidades encontradas, los parches aplicados, las configuraciones que causan conflictos, etc.

Diversas aplicaciones pueden tomar como entrada este esquema para interpretar y tomar las medidas necesarias para mitigar las amenazas encontradas y los conflictos de configuración.

Es a partir del esquema de resultados que se presentan los datos al usuario convirtiéndolos a formato HTML o algún formato que permita la visualización de los resultados de forma clara.

2.5 ESQUEMAS MANEJADOS

El lenguaje OVAL maneja tres distintas categorías: características del sistema, definición y resultados, cada una definida en un esquema distinto.

En general los esquemas están conformados de forma jerárquica, manejando un esquema principal y múltiples esquemas componentes que separan los test describiendo distintos tipos de software, distintos sistemas operativos, etc, esta forma jerárquica de estructura facilita añadir nuevos esquemas componentes sin necesidad de modificar el esquema principal, también facilita la modificación o eliminación, otra ventaja de esta estructura es la elección de elementos en el análisis, ya que se encuentra perfectamente clasificado, es posible elegir entre aplicaciones o sistemas operativos, de esta forma se reduce considerablemente el proceso de análisis.

Para la modificación de esquemas se cuenta con un moderador OVAL que se trata de un individuo o de una organización, su tarea es proporcionar una guía imparcial para la modificación, este personaje generalmente es un integrante de la corporación MITRE[11].

El proceso de modificación de un esquema OVAL es el siguiente:

1. Se realizan sugerencias y comentarios por parte de la comunidad OVAL al moderador, exponiendo las modificaciones a realizar.
2. Se determinan las sugerencias y comentarios válidos que pueden ser considerados en la nueva versión.
3. La comunidad OVAL propone modificaciones, eliminaciones o elementos a añadir en la versión propuesta. Esta fase es coordinada por el moderador.
4. El moderador verifica que los elementos del lenguaje sean válidos.
5. Se publica en el repositorio OVAL la versión oficial del esquema.

[11] Definido en terminología básica. Punto 1.1

Para facilitar la modificación de esquemas componentes, los esquemas OVAL manejan dos números enteros en sus versiones, el primero especifica la versión del esquema principal y el segundo la versión de los esquemas secundarios, un esquema principal es válido hasta que existen anulaciones de esquemas componentes, mientras sólo se añadan, se modifica sólo la segunda parte de la versión, es decir la correspondiente a los esquemas componentes.

2.5.1 Esquema de características del sistema

Representa la información de configuración del sistema a evaluar. Incluye parámetros del sistema operativo, características del software de aplicación instalado y características de configuración que influyen en la seguridad del sistema.

El propósito de este esquema es proporcionar una base de datos con información del sistema que sirve de referencia para ser comparado con el esquema de definición.

2.5.1.1 Descripción del esquema de características del sistema

Almacena en formato XML las características de configuración del sistema a evaluar, incluyendo, características de las aplicaciones instaladas, sistema operativo instalado, parámetros de configuración del sistema, etc.

2.5.1.2 Estructura del esquema de características del sistema

Consiste de cuatro secciones distintas:

- Generador
- Información del sistema
- Objetos coleccionados
- Datos del sistema

Generador <generator>

Permite definir la herramienta de creación, la versión y la fecha y hora en que el documento fue generado.

Información del sistema <sys_info>

Contiene detalles sobre la arquitectura del sistema operativo instalado y sobre las interfaces de red.

Objetos coleccionados <collected_objects>

Muestra la correspondencia entre los objetos contenidos en el documento de definición y los existentes en el sistema. Cada objeto de esta sección consta de los siguientes elementos:

- Identificador de objeto
- Un apuntador de referencia al valor asociado de la sección de system_data
- Una bandera que muestra la situación de los objetos del sistema. El conjunto de posibles banderas es el siguiente:
- complete: cada instancia de los objetos fue referenciada en el archivo de características.
- does not exist: El objeto no puede ser encontrado en el sistema.
- error: Un error ocurrió mientras se realizaba la operación.
- incomplete: Sólo un subconjunto de los objetos buscados fue encontrado.
- not collected: No se intentó obtener información del sistema.
- not applicable: El objeto no es aplicable al sistema evaluado.

Datos del sistema <system_data>

Contiene información sobre las características del sistema reunidas. Está compuesto por un identificador y una bandera de estatus.

2.5.1.3 Resultados del esquema de características del sistema

De acuerdo a la estructura manejada por el esquema de características, se facilita el manejo de la información, el esquema de características presenta toda la información de los elementos contenidos en el equipo, es una especie de inventario realizado con un formato especial para facilitar el análisis y presentar compatibilidad.

2.5.2 Esquema de definición

Este esquema cumple tres funciones principales:

- *Definición de vulnerabilidades:* Identificación y descripción de vulnerabilidades presentes en un sistema.
- *Definición de parches:* Identifica los parches aplicables a un sistema.
- *Definición de configuración:* Determina las características de configuración que representan una amenaza para el sistema.

2.5.2.1 Descripción del esquema de definición

El esquema de definición permite la construcción de sentencias lógicas de la siguiente forma:

¿Está instalado el sistema operativo X?

¿Y el servicio Y se encuentra habilitado?

¿Y se permite al usuario Z usar el servicio Y?

Cada pregunta es representada mediante parámetros de configuración específicos y un valor asociado.

El conjunto de estas preguntas conforman un estado de máquina que sirve como punto de partida para la evaluación de diversos sistemas.

2.5.2.2 Estructura del esquema de definición

El documento que conforma el esquema de definición consta de seis distintas secciones:

- Generador
- Definiciones
- Pruebas
- Objetos
- Estados
- Variables

La información contenida en las pruebas, objetos, estados y variables puede ser expresada a través de múltiples definiciones.

Generador

En esta parte se especifica la herramienta usada para generar el documento, la versión del esquema y la hora en que fue creado en su totalidad el documento XML.

Definiciones

Aquí se incluyen todas las definiciones que conforman el documento. Cada definición se distingue por un identificador global único con la siguiente estructura:

Nombre de la organización DNS: Identificador de tipo: identificador de valor

Nombre de la organización DNS: Provee una fuente de información a cerca de la definición específica.

Identificador de tipo: Identifica el tipo de definición. Los identificadores pueden ser los siguientes:

def - definición

obj - objeto

ste – estado

tst – prueba

var – variable

Identificador de valor: Es un valor único que identifica la definición, está dado por la combinación del nombre de organización DNS y el identificador de tipo.

Cada definición puede estar dividida en dos secciones:

- <metadata>
- <criteria>

La sección <metadata> describe la información de cabecera asociada con la definición, está compuesta por:

- Titulo corto
- Información de plataforma afectada
- Descripción
- Referencias para mayor información

La sección <criteria> provee del medio para conformar una combinación lógica y anidada de las distintas evaluaciones y permite también asociar otras definiciones, esta sección está compuesta por:

- Referencia al nivel correspondiente de la evaluación
- Bandera de negación para controlar las respuestas del test [verdadero, falso]
- Comentario que describe la intención de la evaluación

Existe una sección opcional definida como <notes>, el objetivo de esta sección es proveer de una documentación más detallada a cerca de aspectos técnicos de la definición.

Pruebas

Aquí están incluidos todos los niveles de evaluaciones individuales necesarios para las definiciones.

La estructura suele ser muy similar entre las distintas pruebas. Incluyen los siguientes elementos:

- *Identificador único:* Maneja el mismo formato que las definiciones, manejadas anteriormente. Es decir, *Nombre de la organización DNS: Identificador de tipo: identificador de valor.*
- *Versión:* Permite administrar las modificaciones.
- *Comentario:* Breve descripción del propósito de las pruebas.
- *Verificación:* Relaciona el objeto de referencia y el estado de referencia. Pueden ser los siguientes valores:
    - *All:* Todos los objetos identificadores deben coincidir con el estado.
    - *At least one:* Por lo menos un objeto identificador debe coincidir con el estado.
    - *Only one:* Solamente un objeto debe coincidir con el estado.
    - *None exist:* Ningún objeto debe coincidir con el estado.
- *Objeto de referencia:* Permite la comparación para la verificación.
- *Estado de referencia:* Permite la comparación para la verificación.

Objetos

La sección de objetos contiene todos los elementos del sistema, por cada objeto, existe un esquema que detalla las propiedades importantes para la prueba.

Los objetos contienen dos elementos importantes:

<behavior> Controla la forma en que los datos son generados por el objeto.

<filter> Es usado para limitar o restringir el número de objetos de comparación a coleccionar.

Estados

La sección de estados complementa la sección de objetos. Representa el valor contra el cual el objeto del sistema actual será comparado. Define los valores específicos de interés y las operaciones a realizar cuando se realice una comparación.

Variables

En ocasiones no todos los valores de las definiciones son conocidos en el momento en el que la definición es escrita. A veces los valores pueden variar dependiendo de las políticas de la organización o de la configuración específica de cada equipo, debido a esto, existe una sección de variables que representan valores que pueden ser modificados en tiempo de ejecución.

En OVAL existen tres tipos de variables:

- Externas: Son obtenidos de una fuente externa.
- Locales: Son obtenidos de objetos en el mismo sistema.
- Constantes: Representan valores estáticos usados en las definiciones.

2.5.2.3 Resultados del esquema de definición

Al estructurar de la forma mencionada en la sección anterior se obtiene un esquema de definición global que contiene todas las características y elementos a ser evaluados, este esquema de definición se encuentra publicado en el repositorio OVAL y es frecuentemente actualizado incrementando el número de objetos a evaluar.

2.5.3 Esquema de resultados

Permite almacenar los resultados de la evaluación del sistema. Este esquema contiene la comparación de las características de configuración del sistema con el esquema de definición. Para la interpretación, de estos resultados existen diversas aplicaciones que ayudan al análisis de las medidas necesarias para mitigar riesgos, por ejemplo, aplicar parches, alterar las características de configuración del sistema, limitar el acceso a determinada área del sistema, etc.

2.5.3.1 Descripción del esquema de resultados

Almacena la salida del proceso de comparación entre el esquema de definición y el esquema de características, sirve como fuente de alimentación de distintas aplicaciones con el fin de interpretar los datos y proporcionar soluciones a las fallas de seguridad.

2.5.3.2 Estructura del esquema de resultados

El esquema de resultados consta de cuatro distintas secciones:

- Generador
- Directivas
- Definiciones OVAL
- Resultados

Generador <generator>

Proporciona información sobre la generación del documento, la herramienta usada en el proceso, la versión del esquema y la fecha de creación.

Directivas <directives>

Existe información contenida en el esquema de resultados que no siempre resulta necesaria, por tal motivo, gran parte de los elementos incluidos son configurados como opcionales. Las directivas permiten definir el resultado de los elementos incluidos en el esquema, esta característica brinda eficiencia para los consumidores de los resultados ya que permite la fácil localización de elementos.

Las definiciones existentes se listan a continuación:

definition_true: Reporta definiciones con resultado de evaluación verdadero.

definition_false: Reporta definiciones con resultado de evaluación falso.

definition_unknown: Reporta definiciones con resultado de evaluación desconocida.

definition_error: Reporta definiciones con resultado de evaluación definido como error.

definition_not_evaluated: Reporta definiciones no evaluadas.

definition_not_applicable: Reporta definiciones que no aplican al sistema evaluado.

Cada directiva tiene asociados dos elementos:

- ***Reported****(verdadero/falso) :* Bandera que permite distinguir si la directiva ha sido incluida o asociada en un documento de resultados.
- ***Content****(thin/full)*: En esta parte se describe la directiva.
    - *Thin:* Indica que sólo se provee un mínimo de información sobre la directiva.
    - *Full:* Indica que la información provista es muy detallada.

Definiciones OVAL <oval_definitions>

Indica el contenido de definiciones OVAL que fueron analizadas con el fin de generar el documento de resultados, y eliminando la necesidad de la presencia de esquema de definiciones al momento de analizar los resultados, esta parte del esquema opcional.

Resultados <results>

Contiene el resultado de análisis, está compuesto por elementos del sistema contiene los siguientes elementos:

- *Definitions*: Resultado de análisis de las definiciones OVAL.
- *Test:* Resultado de análisis de cada evaluación considerada.
- *OVAL_system_characteristics:* Es una copia del documento de características del sistema.

2.5.3.3 Resultados del esquema de resultados

El resultado de este esquema sirve de entrada a diversas herramientas proporcionando un reporte de las vulnerabilidades y huecos de seguridad encontrados, hay que aclarar que a pesar de que se encuentren las referencias para obtener más información sobre la vulnerabilidad, esto no implica que el esquema proporcione automáticamente soluciones al problema dado, de esta tarea se encargan herramientas con “adopción” OVAL.

2.6 REPOSITORIO OVAL

Las distintas definiciones manejadas por OVAL son publicadas en repositorios mantenidos por la corporación MITRE, por un conjunto de expertos compuesto por distintas instituciones y por usuarios voluntarios que reportan anomalías detectadas.

Por medio de los repositorios se puede analizar, discutir, almacenar y publicar o difundir las distintas definiciones OVAL.

En la actualidad existen un total de 7369 definiciones distribuidas entre distintos sistemas operativos como se muestra en la figura 2.1:

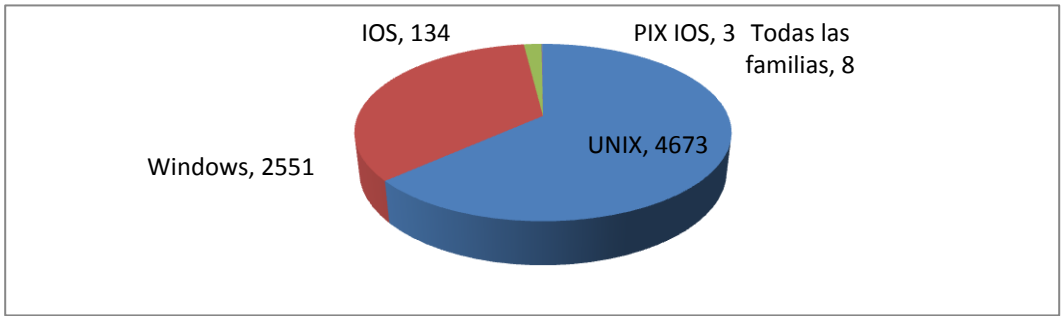


Figura 2.1 Distribución entre sistemas operativos de esquemas OVAL.
Fuente: http://oval.mitre.org/repository/

En los repositorios se publican las actualizaciones a las definiciones, ya sea que se eliminen, se modifiquen o se creen nuevas.

Dentro del repositorio es posible encontrar distintas clases de definiciones las cuales se muestran a continuación:

- *Vulnerability*. Evalúa y determina la presencia de vulnerabilidades.
- *Patch*. Evalúa donde un determinado parche puede ser aplicado.
- *Inventory*. Verifica las aplicaciones instaladas en el sistema.
- *Compliance*. Evalúa y determina el estado de configuración del sistema.
- *Micellaneous*. Aquí entran definiciones que están fuera de las definiciones anteriores.

Dentro de la clasificación anterior el repositorio realiza una clasificación más para las definiciones, que es la siguiente:

- *Por familia:* Separa entre Windows y UNIX.
- *Por plataforma:* Separa entre los distintos sistemas operativos y sus versiones.
- *Por volumen:* Contiene todas las definiciones de la clase en un solo archivo, se encuentran en formato XML ya sea comprimido o no.

Cada documento que se encuentra en el repositorio, tiene publicado el Hash MD5 para verificar integridad de los datos e impedir modificación en los esquemas, la fecha de actualización y el tamaño del archivo.

Existe también un espacio donde se publican únicamente las definiciones modificadas o añadidas, permitiendo actualizar los esquemas de forma manual o recurriendo directamente al esquema completo actualizado.

El repositorio cuenta con un buscador de definiciones, donde es posible especificar distintas características para agilizar la búsqueda, características como:

- Clase
- Plataforma
- Producto
- Contribuidor
- Organización
- Familia
- Fuente de referencia
- Número de referencia

Finalmente el repositorio cuenta con una parte de instrucciones para ser miembro del foro del repositorio OVAL y de esta forma realizar contribuciones ya sea añadir definiciones o modificar las existentes.

2.7 ALCANCES OVAL

Debido a la independencia de esquemas y a la facilidad de manejo de información es posible llevar a cabo distintas acciones o actividades con la combinación de esquemas o trabajando con ellos de forma separada.

Entre las actividades que es posible realizar mediante estos esquemas se encuentran las siguientes:

Evaluación de vulnerabilidades: Permite examinar los parámetros que identifican la presencia de una vulnerabilidad.

Administración de parches: Después de identificar las vulnerabilidades presentes en un sistema, el esquema presenta la referencia al sitio de descarga de parches. Independientemente de eso, una de las clases de definiciones es específicamente parches por lo que es posible identificar si un parche es aplicable a un sistema.

Administración de configuración: Después de analizar el estado de configuración de un sistema se realiza la comparación con el listado de configuración optima haciendo un reporte de los resultados.

Inventarios: Permite identificar las aplicaciones existentes en un sistema, sus características y su versión.

Auditoria: Permite registrar el estado de cierto sistema identificando las características de configuración, de esta forma es posible tener posteriormente un punto de comparación entre cambios que haya sufrido el sistema.

Las acciones definidas anteriormente son solamente un ejemplo de lo que es posible realizar mediante las definiciones OVAL depende de cada programador o de las necesidades de las organizaciones el uso que se dé y la combinación con otros elementos para nutrir las herramientas creadas.

2.8 PRODUCTOS COMPATIBLES CON OVAL

Se define como "adopción OVAL" al hecho de tomar los esquemas publicados por la comunidad OVAL y crear herramientas que basen su funcionamiento en las definiciones establecidas.

Se debe recordar que OVAL es libre y gratuito por lo que cualquier programador puede emplearlos como herramienta e incluirlos en la creación de sus aplicaciones.

Ejemplos de este tipo de programas hay muchos, es importante observar que grandes organizaciones se encuentran involucradas con esta comunidad y han creado a partir de OVAL distintas aplicaciones, algunas en estado beta otras ya disponibles, organizaciones como Hewlett-Packard, la comunidad Red Hat, U.S. Army CERDEC y muchas más. A continuación se muestra la tabla 2.1 con algunos ejemplos de estas aplicaciones.

Nombre	Creador	Funciones
Armadillo	U.S. Army CERDEC	Realiza el inventario de software, parches, configuraciones y vulnerabilidades en redes.
CA Total Defense	CA, Inc.	Mitiga y reporta riesgos que pueden surgir debido a vulnerabilidades y a configuraciones erróneas.
eSCAPe - Enhanced SCAP Editor	G2, Inc.	Adoptan tecnología OVAL y SCAP, permiten crear contenidos sin necesidad de conocer el lenguaje OVAL y XML.
HP Client Automation	Hewlett-Packard	Detección de vulnerabilidades en tiempo real.
HP Live Network	Hewlett-Packard	Ofrece actualización diaria, para mantener protegidos los entornos de red.
Red Hat security advisories	Red Hat	Repositorio de definiciones.
SUSE Linux Enterpris OVAL Information	Novell, Inc.	Repositorio de definiciones.
Xacta IA Manager HostInfo	Telos Corporation	Evaluador de definiciones.

Tabla 2.1 Aplicaciones basadas en OVAL

CAPITULO III

DESARROLLO DE LA APLICACIÓN

3. DESARROLLO DE LA APLICACIÓN

En el capitulo anterior se ha mostrado detalladamente la forma de trabajo de OVAL, esto se hizo con el fin de comprender la estructura de la aplicación en este trabajo desarrollada. A continuación, el presente capítulo, mostrará el desarrollo de la aplicación.

3.1 ANÁLISIS

Esta etapa es la primera en el desarrollo de cualquier aplicación, aquí se define lo que se quiere lograr con la implementación de determinado software, que es lo que se espera, cuáles son los objetivos, las características, la forma esperada de funcionamiento; esta etapa es muy importante hablando de software comercial ya que es aquí donde se define lo que el cliente espera de la aplicación, es importante porque es la base para la creación de software funcional y de calidad ya que si no se sabe qué es lo que se busca, la aplicación puede resultar ambigua.

En esta etapa también se realiza una evaluación de la factibilidad de desarrollo de la aplicación, en este caso estamos hablando solamente de software libre comenzando desde el sistema operativo Debian GNU/Linux, los esquemas OVAL y el lenguaje de programación (Perl), por lo que se declara totalmente factible su desarrollo. No es lo mismo en el caso de software comercial, ya que se deben evaluar aspectos como personal que desarrollará la aplicación, equipos utilizados en el desarrollo, el impacto que puede tener la nueva aplicación en el sistema, los riesgos que acompañan su implantación, la capacitación de personal para el uso de la nueva aplicación, etc. Aquí sólo se considerará el desarrollo como una aplicación independiente dejando como responsabilidad del usuario le evaluación de costos que su uso implique.

3.1.1 Objetivos

Los objetivos de la creación de una aplicación que actualice de manera automática vulnerabilidades del sistema son los siguientes:

- Reducir el tiempo en la aplicación de parches.
- Disminuir los riesgos presentes en el sistema.
- Disminuir vulnerabilidades presentes en el sistema.
- Facilitar al administrador la evaluación y corrección del sistema.
- Disminuir el costo en aplicación de parches.
- Automatizar la labor del administrador al instalar parches.
- Mantener un registro de los cambios que sufre el sistema al instalar parches.
- Mantener el sistema actualizado de acuerdo a la publicación de parches OVAL.

La actualización constante de los esquemas OVAL permite obtener los parches adecuados en el momento que son publicados, debido a la relación que tiene OVAL con los proveedores de productos e instituciones de publicación de parches.

3.1.2 Requisitos

El desarrollo de la aplicación requiere de los siguientes requisitos:

- Debido a que la aplicación se realizará en lenguaje de programación Perl, es necesario contar con un intérprete de Perl.
- Necesario contar con un editor de texto en el cual se desarrollará la aplicación.

- El sistema operativo al cual se aplica es exclusivamente Debian, por lo que es necesario contar con equipos con el sistema instalado para la realización de pruebas.

- Se debe contar con conexión a internet para distintas consultas y descargas.

- El espacio en disco duro debe ser de aproximadamente 100 Mb.

- La memoria RAM debe ser mínimo de 256 Mb.

3.1.3 Características esperadas

Se espera que la aplicación pueda realizar de manera automática un inventario del sistema para de esta forma determinar el estado del sistema y aplicar los parches correspondientes.

Es necesario que se manifiesten las siguientes características:

- Sistema operativo presente en el equipo a evaluar

- Versión del sistema operativo

- Aplicaciones instaladas

Estas características son importantes ya que son manejadas por el esquema OVAL con el cual se trabajará.

La aplicación debe descargar de forma automática el esquema OVAL que será utilizado para referenciar los parches, esta descarga debe ser cada que se ejecute la aplicación ya que los esquemas tienen constantes modificaciones y actualizaciones.

La aplicación debe evaluar el sistema e identificar que parches son aplicables.

En base a los parches identificados como aplicables, debe realizar una descarga de los mismos.

En caso de que los parches se encuentren comprimidos o dentro de un paquete, la aplicación debe convertirlos a un formato manejable, extrayéndolos de forma adecuada.

Una vez extraídos los parches debe instalarlos de forma adecuada o notificar en una bitácora la causa por la que no se ha instalado correctamente. Toda acción debe ser registrada en bitácora.

3.1.4 Delimitación del campo de aplicación

La aplicación será diseñada para trabajar con sistemas operativos Debian, en realidad esta aplicación es una muestra de trabajo con esquemas OVAL, puede extenderse a otros sistemas pero la aplicación será exclusiva para Debian ya que es uno de los sistemas más estables y de mayor uso, entre las ventajas que posee este sistema se encuentran las siguientes:

- ***Instalación sencilla:*** Contando con distintas opciones de instalación (CD, DOS, discos flexibles, directamente de la red).

- ***Increíble cantidad de software:*** Incluye más de 18733 elementos de software diferentes.

- ***Paquetes bien integrados:*** Debian sobrepasa a todas las otras distribuciones en lo bien integrados que están sus paquetes. Presenta facilidad para encontrar cada paquete ya que están localizados en un mismo sitio.

- ***Código fuente:*** Hay cientos de herramientas y lenguajes de desarrollo, además de millones de líneas de código fuente en el sistema base. Todo el software en la distribución principal es conforme al criterio de las Directrices de Software Libre de Debian (DFSG). Esto significa que se puede usar libremente este código para estudiarlo o para incorporarlo a un nuevo proyecto de software libre. También hay una buena cantidad de herramientas y código apropiado para el uso en proyectos propios.

- ***Actualizaciones fáciles:*** Facilita la actualización a nuevas versiones de Debian debido al sistema de empaquetamiento, solo se debe ejecutar apt-get update ; apt-get dist-upgrade (o aptitude update ; aptitude dist-upgrade, según la versión) y se realizará la actualización desde un CD en cuestión de minutos o se puede configurar apt para utilizar alguno de los trescientos espejos de Debian y actualizarlo desde la red.

- ***Sistema de seguimiento de errores:*** El sistema de seguimiento de errores de Debian es público. Donde se envían informes de errores y se buscan soluciones que son publicadas para su pronta solución. Este sistema permite que Debian responda a los problemas rápida y honestamente.

- ***Estabilidad:*** Existen muchos casos de máquinas que trabajan durante más de un año seguido sin reiniciarse. De la misma forma, hay equipos que tan sólo son reinicializados debido a un fallo en el suministro de corriente o a una actualización del hardware.

- ***Rápido y ligero en memoria:*** Otros sistemas operativos pueden ser rápidos en una o dos áreas, pero, estando basado en GNU/Linux, Debian es ligero. El software para Windows se ejecuta bajo GNU/Linux usando un emulador a veces más rápido que en su ambiente original.

- ***Controladores:*** Los controladores para la mayoría del hardware están escrito por usuarios de GNU/Linux, no por el fabricante, lo que permite que continúe el soporte mucho después de que el fabricante haya detenido su producción.

- ***Seguridad del sistema:*** Debian y la comunidad del software libre procuran vehementemente que los problemas de seguridad sean solucionados rápidamente, publicando las soluciones. La disponibilidad del código fuente permite que la seguridad en Debian se evalúe de forma abierta, lo que evita que se implementen modelos de seguridad pobres. Además, la mayoría de los proyectos de software libre tienen sistemas de revisión por terceras partes, que, como primera medida, evitan que se introduzcan en el sistema problemas de seguridad potenciales.

- ***Software de seguridad:*** Debian tiene paquetes del famoso software GPG (y PGP) que permite enviar correo entre usuarios preservando su privacidad. Además, SSH permite crear conexiones seguras a otras máquinas que tengan SSH instalado.

3.1.5 Funcionalidades

La aplicación facilitará distintas tareas, y tendrá las siguientes funcionalidades:

- Instalador automático de parches
- Verificador del estado del sistema
- Herramienta para centralizar parches
- Herramienta que permite mantener actualizada una bitácora que registre cada acción en la administración de parches
- En general un administrador de parches.

3.1.6 Comportamiento esperado

Se espera que la aplicación realice sus actividades de forma automática, podría manejarse, en caso de que el administrador lo crea pertinente, la ejecución al inicio del sistema de esta forma la actualización de parches iría de la mano con el inicio del sistema, esto puede representar un problema para sistemas que permanecen encendidos durante meses, por lo que esta cuestión se deja en manos del administrador.

Existirán casos en los que la aplicación no puede instalar el parche, para estos casos, la aplicación debe indicar en una bitácora que el parche no ha sido instalado, de esta forma el administrador tendrá control sobre el sistema actuando de forma inmediata en caso de que la vulnerabilidad que no ha sido cubierta por el parche represente un gran riesgo para el sistema.

Toda la actividad llevada a cabo por la aplicación, será registrada en una bitácora para mantener el control sobre los parches.

3.2 DISEÑO

En esta parte del desarrollo de software, se establecerá la forma en la cual se resolverán y satisfarán las características planteadas durante la fase de análisis, en esta etapa se pretende establecer la forma de trabajo de la aplicación para lograr los objetivos planteados.

Se definirá el lenguaje de programación más adecuado de acuerdo a las características de la aplicación, se definirá también el algoritmo del sistema y su correspondiente diagrama de flujo y finalmente se establecerá la interfaz con el usuario.

En teoría en esta parte queda resuelto la forma en la cual va trabajar la aplicación, esta parte es muy importante para todos los desarrolladores de software, actualmente muchos desarrolladores se saltan este paso yendo directamente a la codificación, lo cual es inadecuado ya que se debe establecer un análisis completo de cómo la aplicación va a trabajar, en esta etapa se suelen encontrar distintos errores que pueden ser resueltos adecuadamente.

3.2.1 Definición del lenguaje de programación

Se ha elegido como lenguaje de programación Perl debido a las siguientes características que se han considerado como ventaja frente a otros lenguajes:

- Perl ha sido desarrollado bajo la idea de un lenguaje de programación con sintaxis lo más parecida al lenguaje natural de las personas, esto con el fin de hacerlo fácil de aprender, de entender y de usar, así facilitar la labor de los programadores.

- Originalmente fue desarrollado para ser un lenguaje de manipulación de texto lo que es una gran ventaja para la aplicación ya que se requiere gran manejo de texto.

- El lenguaje Perl posee una sintaxis especial para el manejo de las expresiones regulares.

- Perl es software libre (bajo licencia GNU y licencia artística), lo que significa que no es necesario pagar para obtenerlo.

- Perl es un eficiente, completo y fácil de usar.

- Debido a que Perl toma características de otros lenguajes de programación (Lisp, C, AWK, etc.), se puede asegurar que la estructura resultante del lenguaje es lo suficientemente robusta y confiable, algo que ya se ha demostrado en la cantidad de aplicaciones en las que se utiliza Perl.

- Existen distintos módulos Perl que facilitan el manejo de datos, en el caso específico de la aplicación a implementar, maneja módulos que facilitan el manejo de archivos xml, el formato en el cual se encuentran los esquemas OVAL a ser utilizados.

- Permite el manejo directo de comandos, esta característica es muy importante ya que gran parte de las tareas a realizar son mediante comandos directos de Linux.

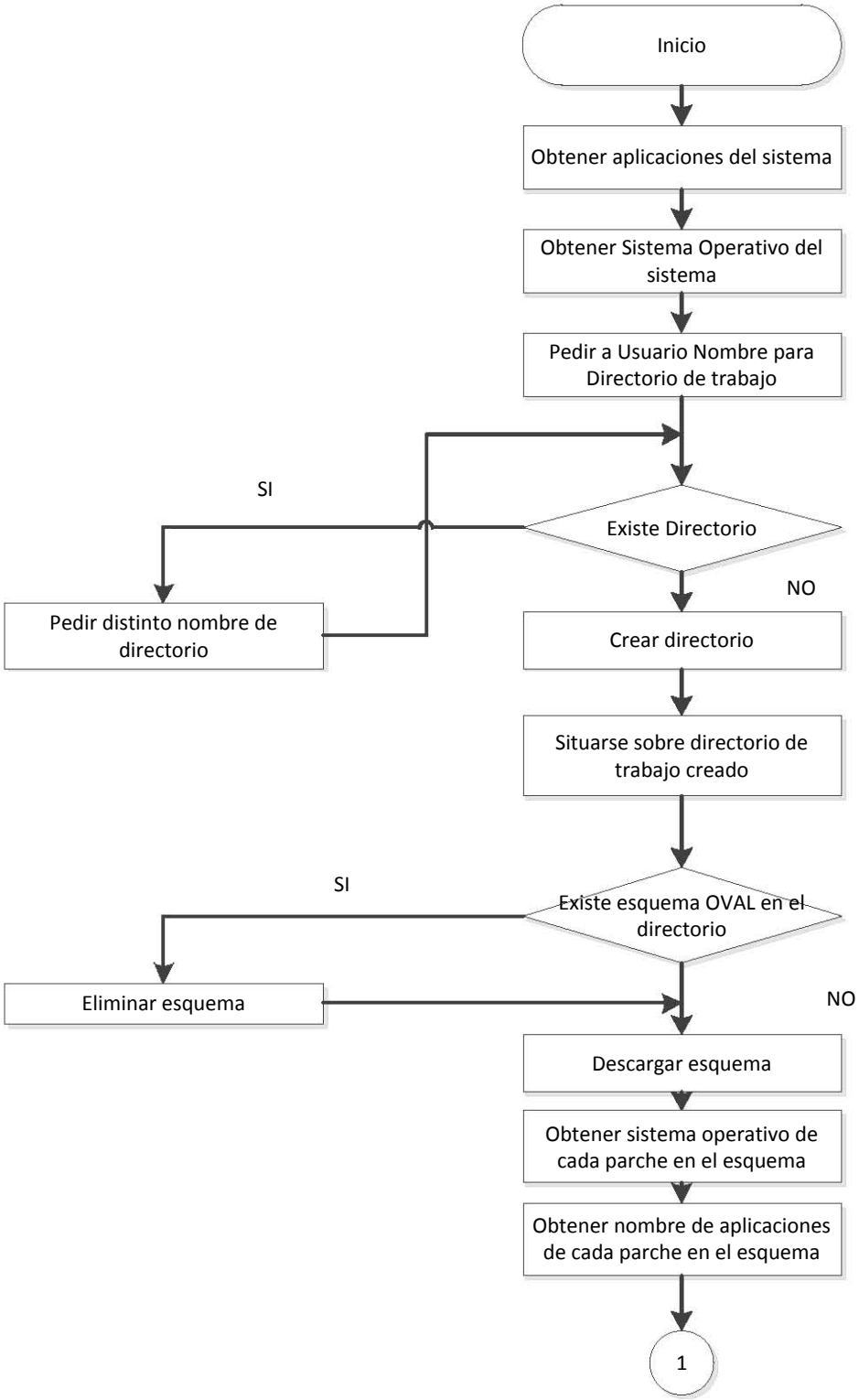
3.2.2 Algoritmo del sistema

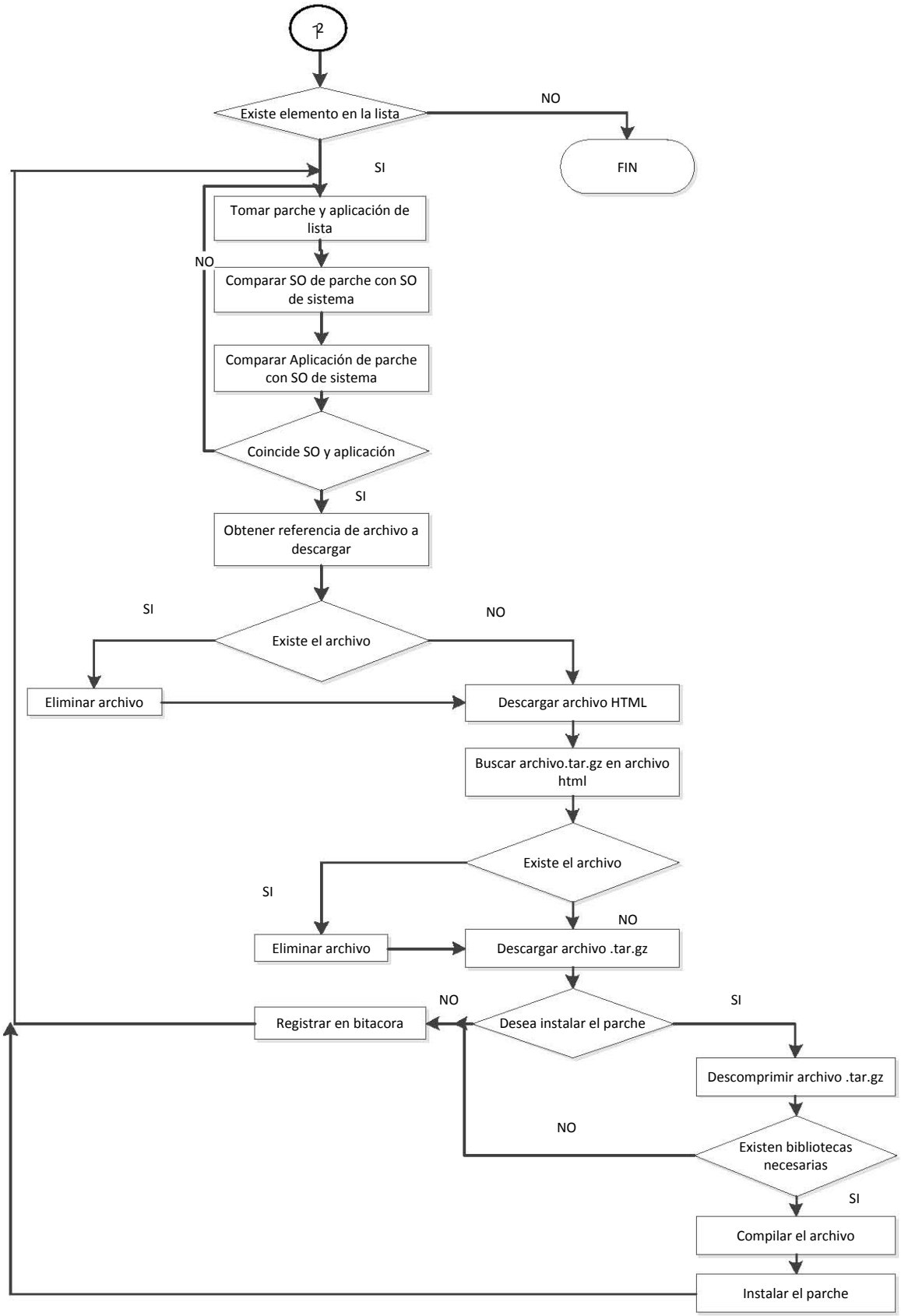
Se mostrará el algoritmo que describirá el proceso que realizará la aplicación.

1. Obtener aplicaciones instaladas en el sistema donde se ejecute la aplicación
2. Obtener el sistema operativo donde se ejecute la aplicación
3. Pedir a usuario nombre de directorio para ejecutar la aplicación
4. Verificar si existe el directorio dado por el usuario
5. Si existe el directorio pedir el nombre de otro directorio en el que se trabajará
6. Si no existe, crear directorio de trabajo
7. Situarse sobre directorio de trabajo creado
8. Verificar existencia del esquema OVAL en el directorio de trabajo
9. Si existe el esquema OVAL en el directorio de trabajo, eliminarlo
10. Descargar de internet el esquema OVAL
11. Obtener sistema operativo de cada parche en el esquema OVAL
12. Obtener nombre de aplicaciones de cada parche en el esquema OVAL
13. Realizar comparación del sistema operativo del parche con el sistema operativo del equipo a evaluar

14. Realizar comparación del nombre de la aplicación del parche con las aplicaciones existentes en el equipo a evaluar
15. Si no coincide sistema operativo y aplicaciones continuar evaluando los siguientes parches para ver si aplican al sistema evaluado
16. Si coincide sistema operativo y aplicaciones obtener la referencia del archivo a descargar.
17. Evaluar si existe archivo con el nombre de referencia a descargar
18. Si existe archivo con el nombre de referencia a descargar, eliminarlo
19. Descargar la pagina html a la que se hace referencia en el esquema OVAL
20. Buscar en la pagina descargada el archivo .tar.gz correspondiente al sistema evaluado
21. Verificar que no exista archivo con el mismo nombre
22. Si existe, eliminarlo
23. Descargar el archivo
24. Preguntar si se desea instalar el parche
25. Si la respuesta es negativa, registrar en la bitácora y continuar evaluando los otros parches
26. Si la respuesta es positiva descomprimir el archivo .tar.gz
27. Evaluar que existen las librerías necesarias para la instalación del parche
28. Compilar el archivo
29. Instalar el parche
30. Continuar con cada parche hasta terminar el esquema

3.2.3 Diagrama de flujo




2
Existe elemento en la lista
NO
FIN
SI
Tomar parche y aplicación de lista
NO
Comparar SO de parche con SO de sistema
Comparar Aplicación de parche con SO de sistema
Coincide SO y aplicación
SI
Obtener referencia de archivo a descargar
SI
Existe el archivo
NO
Eliminar archivo
Descargar archivo HTML
Buscar archivo.tar.gz en archivo html
Existe el archivo
SI
NO
Eliminar archivo
Descargar archivo .tar.gz
NO
SI
Registrar en bitacora
Desea instalar el parche
Descomprimir archivo .tar.gz
NO
Existen bibliotecas necesarias
SI
Compilar el archivo
Instalar el parche

3.2.4 Establecimiento de interfaz

Esta aplicación es sólo el inicio de lo que podría ser un administrador de parches, por el momento la interfaz con el usuario será vía comandos, manejada desde una consola como un comando más.

Perl puede manejar elementos gráficos, pero la intención de esta aplicación es la administración de parches, los usuarios a los cuales está dirigido son generalmente administradores que tienen noción del uso de comandos vía consola, el hecho de proveerla con una interfaz gráfica puede requerir mayor número de recursos y alentar el proceso en general.

Se pretende que la aplicación sea lo más automática posible por lo que es innecesario establecer una interfaz gráfica por el momento ya que lo único que se necesita es la ejecución, independientemente de eso, la única interacción que tiene con el usuario es al momento de preguntar sobre la instalación de cada parche, de ahí en fuera el sistema trabaja de forma autónoma.

3.3 IMPLEMENTACIÓN

Una vez que se ha diseñado la aplicación, teniendo en cuenta el algoritmo del sistema y el diagrama de flujo se codificará de acuerdo al lenguaje de programación elegido. Esta parte del desarrollo es de las finales ya que previamente se debe considerar la solución de forma general tomando en cuenta los requerimientos establecidos en la fase de análisis, los objetivos, las características, y el funcionamiento esperado, esto es independiente del lenguaje de programación empleado, el hecho de llevar a cabo un desarrollo por etapas ayuda al programador a desarrollar una aplicación de forma más ordenada y limpia, eliminando errores en las etapas adecuadas independientemente de la fase de pruebas, es necesario que el programador posea la base sobre la cual va programar es decir el resultado de la etapa de diseño.

En esta etapa se realizará la codificación analizando entradas y salidas de los elementos principales de la aplicación, al terminar esta etapa la aplicación queda lista para pruebas.

3.3.1 Codificación

En esta parte del documento se llevará a cabo la explicación del código fuente, se procederá incrustando fragmentos de código al mismo tiempo que se realiza una explicación de los mismos.

Encabezados

En la primera parte del script se colocará un indicador del lenguaje en el cual está escrito el programa y los módulos que serán empleados durante la ejecución, en este caso se incluyen dos módulos que facilitan el manejo de documentos XML.

```
#!/usr/bin/perl
use XML::XPath;
use XML::XPath::XMLParser;
```

Subrutinas de impresión

Se crean dos subrutinas para facilitar la impresión en pantalla y bitácora. Estas subrutinas serán llamadas cada vez que sea necesario registrar las acciones realizadas.

```
#########Impresion en bitacora y pantalla#########################
sub impri
{
        my ($tex)= @_;
        my ($s, $m, $h)=localtime;              #Obtiene hora actual
        print "\b"x79, " "x79, "\b"x79,
                sprintf("[%02d:%02d:%02d]", $h,$m,$s),
                "\t", $tex, "\n";
        impri_log($tex);                        #Llama funcion de escritura en bitacora
}
#########Imprime solo en bitacora ##############################
sub impri_log
{
        $f_log = "/var/log/Ovpatch.log";        #Define archivo de bitacora
        my ($tex)= @_;
        open FLOG, ">> $f_log"
             or die "No es posible abrir archivo de bitacora";
        print FLOG "[", scalar(localtime), "] \t $tex\n";
        close FLOG;
}
```

Análisis del sistema

Como primer punto se realizará el parseo del sistema en el cual se ejecuta el script obteniendo las características.

Comenzando por las aplicaciones instaladas, mediante el comando **`dpkg --get-selections`**, almacenando el resultado en un arreglo y guardando solo la parte que será útil para seleccionar los parches que serán descargados e instalados. El resultado será almacenado en el arreglo **`@aplicaciones3`**.

```
#########Obtener informacion del sistema#########################
###############Aplicaciones instaladas##########################
@aplicaciones=`dpkg --get-selections`;        #Obtiene aplicaciones instaladas
impri("* Obteniendo informacion del sistema [Aplicaciones]...\n");
my $poss=0;
foreach $apli (@aplicaciones)
{
    @aplicaciones2=split(/\t/,$apli);
    $aplicaciones3[$poss]=$aplicaciones2[0];  #Arreglo con todas las apli
    $poss++;
}
```

Se identificará el sistema operativo mediante el comando **`head -n1 /etc/issue`** y se almacenará la parte útil en la variable **`$so2`**.

```
#################Sistema operativo###################################
my $so=`head -n1 /etc/issue`;                    #Obtiene version de SO
impri ("* Obteniendo informacion del sistema [Sistema Operativo]...\n");
my @so1=split(' ',$so);
my $so2= $so1[0]." ".$so1[1]." ".$so1[2];        #Contiene sistema operativo
```

Preparar entorno

Para mantener un orden, se creara un nuevo directorio sobre el cual se trabajará, el usuario debe proporcionar el nombre del nuevo directorio al momento de insertar el comando, se comprobará que el directorio no exista para no sobrescribir un directorio con el mismo nombre.

En caso de manipulación de este directorio, se comprobará que no exista en el directorio el esquema OVAL con el cual se va trabajar, si se del caso de que esquema exista se borrará para que sea actualizado.

Mediante estas acciones, se puede comenzar a trabajar en un ambiente preparado para los propósitos de la aplicación.

```
#################Crear entorno de trabajo#############################
my $archivo_xml="unix.xml";
my $nombre_dir=$ARGV[0];                 #Asigna a var parametro de usuario

while(-e $nombre_dir)                    #si existe el directorio pide otro
{
   impri ("El directorio sobre el que desea trabajar ya existe. Especifique otro:\n");
   $nombre_dir=<STDIN>;                  #Se asigna a la variable la entrada del teclado
   chop($nombre_dir);                    #Se elimina el retorno de carro
}

`mkdir $nombre_dir`;                     #Crea directorio de trabajo
impri( "* Creando directorio $nombre_dir...\n");
chdir($nombre_dir);                      #Cambia de directorio de trabajo
$diract=`pwd`;
chop ($diract);
impri( "* Cambio de directorio de trabajo a $diract...\n");
if(-e $archivo_xml)                      #si existe archivo xml lo borra
{
   impri( "* El esquema ya existe, se actualizara...\n");
   my $borra=`rm $archivo_xml`;
}
```

Descarga de esquema

Se descargará el esquema de la página oficial oval, mediante el comando wget. Esto permite trabajar con un esquema actualizado.

```
##################Descarga el esquema##########################################
impri( "* Descargando el esquema...\n");
`wget -nv http://oval.mitre.org/rep-data/org.mitre.oval/p/family/unix.xml`;
```

Análisis del esquema

Ya que se tiene el esquema OVAL, se analizará para obtener las características importantes para descartar parches que no apliquen al sistema, el análisis del archivo xml será mediante un modulo de Perl especial para trabajar con archivos de este tipo, existen distintos módulos que facilitan esta tarea, se ha elegido éste debido a que permite manejar los datos de forma cotidiana, por medio de un esquema jerárquico, trabajando con rutas, existen palabras clave que serán tomadas como referencia, en este caso “definition” que es la palabra que delimita cada definición, se obtendrá cada elemento dentro de esos límites y se almacenará en el arreglo @producto.

Una vez que se tiene un arreglo se manipulará para obtener la aplicación, y el sistema operativo.

Debido a que existen definiciones que son aplicadas a dos sistemas operativos al mismo tiempo, existe un arreglo para sistema operativo uno y dos.

Se almacenara la información de la siguiente forma:

@plataform2= sistema operativo 1
@plataform4= sistema operativo 2
@product2=Nombre de la aplicación
@title2=Nombre de la vulnerabilidad
@description=descripción de la vulnerabilidad
@reference2=URL de parches

```
##################Analisis del esquema################################################
impri( "* Analizando el esquema...\n");
my $cont=0;
my $arbol= XML::XPath->new(filename =>$archivo_xml);                #Genera un nuevo objeto
my $nodos= $arbol->find('oval_definitions/definitions/definition'); #Sigue ruta
my $contador=0;
foreach my $elemento ($nodos->get_nodelist)                         #para cada elemento en ruta
{
   $producto[$cont]=XML::XPath::XMLParser::as_string($elemento);    #Almacena en arreglo
   $cont++;
}
impri( "* Comparando esquema con caracteristicas del sistema...\n");
foreach $pr (@producto)
{
    my @linea=split(/\n/,$pr);                      #Separa por renglones
    my @plataforma1,@s2,@s3,@s4,@s5,@s6,@pl2=();    #Limpia arreglos
    my $count=0;
    foreach my $i(@linea)                           #Para cada linea
    {
       if ($i =~ /<platform>/)                      #Si la linea contiene la etiqueta plataforma
       {
           $pl2[$count]=$i;                         #Guarda en un arreglo plataforma
           $count++;
       }
    }
    #Separa los elementos
    my @platform1=split(/<platform>/,$pl2[0]);
    my @platform2=split(/<\/platform>/,$platform1[1]);  #Sistema operativo 1
    my @platform3=split(/<platform>/,$pl2[1]);
    my @platform4=split(/<\/platform>/,$platform3[1]);  #Sistema operativo 2
    my @product1=split(/<product>/,$pr);
    my @product2=split(/<\/product>/,$product1[1]);     #Nombre de la aplicacion
    my @title1=split(/<title>/,$pr);
    my @title2=split(/<\/title>/,$title1[1]);           #Nombre de la vulnerabilidad
    my @description1=split(/<description>/,$pr);
    my @description2=split(/<\/description>/,$description1[1]); #Descripcion de vulnerabilidad
    my @reference1=split(/<reference/,$pr);             #URL de parches
    my @reference2=split("/",$reference1[1]);
```

Comparaciones

Debido a que se maneja más de un sistema operativo en algunas definiciones, se emplea una bandera para indicar si la comparación da resultado satisfactorio en alguno de los casos.

```
##################Comparaciones#######################################################
##################Sistema operativo###################################################
   my $eqos=0;
   my $eqapl=0;
   if ($platform2[0] eq $so2 or $platform4[0] eq $so2) #comparando plataformas
   {
        $eqos=1;                                       #Activa bandera
   }
```

Se compararán las aplicaciones almacenadas en el arreglo que mantiene las aplicaciones instaladas en el sistema y las obtenidas del esquema OVAL, señalando con una bandera en caso de coincidir.

```
###############Aplicaciones#####################################################
foreach $elemento (@aplicaciones3)                    #Comparando aplicaciones
{
    if ($product2[0] eq $elemento)
    {
        $eqapl=1;                                     #Activa bandera
    }
}
```

Obtener URL del parche

En caso de que coincida tanto el sistema operativo como la aplicación instalada en el sistema se obtendrá la URL que contiene la referencia al parche y se almacenará en el arreglo @referencias.

```
###############Elementos que apliquen#########################################
if ($eqos eq 1 and $eqapl eq 1)                       #Si ambas banderas activadas
{
     $referencias[$contador]=$reference2[3];          #Almacena en arreglo referencias
     $contador++;
}
```

Obtiene el nombre del archivo

Basándose en la URL, obtiene solo el nombre de la pagina a la que hace referencia, por lo general es html, los nombres de los archivos son almacenados en el arreglo @nom_arch1.

```
##################Obteniendo nombre de paquetes#################################
my $contt=0;
foreach $elem (@referencias)
{
    my @nom_arch=split(/\//,$elem);
    $nom_arch1[$contt]=$nom_arch[$#nom_arch];
    $contt++;
}
```

Descarga de archivos con referencias a los parches

Una vez que se tiene en un arreglo todos los parches aplicables, se descargarán las páginas que contienen la referencia directa al parche y se almacenaran en el directorio activo, es decir, el directorio creado para la ejecución de la aplicación.

Primero se verifica que no exista el archivo, en caso de que así sea, se eliminará para ser actualizado.

```
##################Descarga de referencias#######################################
impri( "* Descargando referencias...\n");
foreach $elem (@referencias)
{
    foreach $elem2 (@nom_arch1)
    {
       if(-e $elem2 and $elem =~ $elem2)     #si existe archivo lo borra
       {
           impri( "* La referencia ya existe, se actualizara...\n");
           my $borra2=`rm $elem2`;
           break;
       }
    }
    `wget -nv $elem`;
}
```

Llamada a subrutina

Se hará referencia a una subrutina que se explicarán a continuación. Se envía como parámetro el nombre del archivo html que contiene la referencia al parche.

```
foreach $elemento (@nom_arch1)
{
     &html($elemento);
}
```

Lectura del archivo HTML

Se realizara la lectura del archivo de referencia mediante un descriptor llamado HTML, cada renglón será almacenado como elemento del arreglo @leehtml para su posterior manipulación.

```
open(HTML, $_[0]);                          #Abre archivo de referencia
$descriptor, @leehtml, @paq=();
while (my $descriptor= <HTML>)              #Para cada linea del archivo
{
        $leehtml[$contador2]=$descriptor;   #Almacena en un arreglo cada linea
        $contador2++;
```

Búsqueda de parche adecuado al sistema operativo

Dentro del archivo se realizará la búsqueda del patrón almacenado en la variable $so2, es decir el sistema operativo sobre el cual se ejecuta la aplicación, esto se realiza con el fin de localizar el parche adecuado al sistema, una vez encontrado, se almacena la posición en la que se ha encontrado en la variable $poss2

```
while (my $descriptor= <HTML>)              #Para cada linea del archivo
{
        $leehtml[$contador2]=$descriptor;   #Almacena en un arreglo cada linea
        $contador2++;
        if($descriptor =~ /$so2/)           #Se busca linea de interes
        {
             my $poss2=$contador2;          #Asigna a variable posicion
        }
}
```

Se realizará la búsqueda del archivo tar.gz que corresponda al sistema operativo en base a la posición almacenada en la variable $poss2, se hará uso de la bandera $encontrado para detener la búsqueda en caso de que se haya encontrado el paquete buscado.

```
foreach $linea (@leehtml)                   #Analiza en el arreglo
{
   if ($encontrado eq 0)                    #Si aun no se ha encontrado la liga buscada
   {
     $cont3++;                              #Contador de lineas
     @paq=();                               #Limpia el arreglo
     if($cont3>$poss2 and $linea =~ /.tar.gz/)  #Elige linea con tar.gz despues de poss2
     {
```

Una vez encontrado el paquete, se obtiene el nombre de la aplicación mediante el uso de la función de perl “split” que separa elementos y se almacena en la variable $desco1.

```
my @paq=split(/"/,$linea);          #Filtra solo parte util
$paquete= $paq[1];

my @patch=split(/\//,$paquete);     #Obtiene nombre de parche descargados
my $patch1=$patch[$#patch];

@desco=split(/\//,$patch1);
$desco1=$desco[$#desco-1];                  #Nombre de aplicacion
```

Sí el parche ya existía en la carpeta, se elimina para ser reemplazado por el nuevo, de esta forma se asegura que el paquete estará actualizado.

```
if(-e $desco1)                      #si existe archivo lo borra
{
     impri( "* El parche ya existe, se actualizara...\n");
     my $borra3=`rm $desco1`;
}
```

Descarga de parche

Una vez que se tiene la URL del paquete a descargar, se descarga mediante wget, con el valor de la variable $paquete previamente almacenado.

```
impri( "* Descargando parche $paquete...\n");
`wget -nv $paquete`;  #Descarga parche
```

Preguntar sobre instalación de parches

Una vez que se ha descargado el parche almacenado en un archivo .tar.gz se preguntará si se desea instalar, a pesar de presentarse como una herramienta automática, es necesaria esta parte ya que los parches no siempre representan una solución adecuada ante ciertas vulnerabilidades ya que en muchos casos es una implementación exprés ante algún problema y no se toman en cuanta

ciertos factores que pueden afectar el sistema mas allá de lo que lo pueden proteger, es por esta razón que se pregunta por cada parche antes de instalarlo ya que distintos documentos entre ellos el publicado por el NIST, SP800-40 hablan de toda una metodología que se debe seguir antes de la instalación de parches, tomando en cuenta lo citado en el documento, se deja al administrador y al sistema de implementación de parches la metodología previa a la instalación, ya que depende de los elementos y fines de cada sistema la decisión a tomar.

Se realiza la evaluación de la respuesta del usuario con el fin de evitar respuestas inválidas que puedan provocar errores en la herramienta.

```
while ($ban1 eq 0)
{
   impri( "Desea instalar el parche $patch [Si/No]\n");
   $respuesta1=<STDIN>;
   chop($respuesta1);              #Se elimina el retorno de carro
   impri ("Respuesta de usuario: $respuesta1\n");
   if ($respuesta1 ne "Si" and $respuesta1 ne "si" and $respuesta1 ne "No" and
       $respuesta1 ne "no")
   {
      impri( "Respuesta incorrecta. Ingrese Si [si] o No [no]\n");
   }
   else
   {
      $ban1=1;
   }
}
```

Instalación del parche

En caso de que la respuesta sea afirmativa, se instalará el parche, para lo cual es necesario descomprimir el parche, colocarse en el directorio descomprimido y realizar la compilación e instalación del parche con los comandos, ./configure, make y make install. Es necesario especificar que la instalación tendrá éxito en caso de que no se requieran elementos extra para la instalación ya que la herramienta realiza la instalación por default sin tomar en cuenta casos específicos de cada aplicación, aun así en ambos casos, éxito o fracaso en la instalación quedará registrado en un log que será de ayuda para el administrador y para el sistema de administración de parches en caso de existir en la organización.

```
if ($respuesta1 eq "Si" or $respuesta1 eq "si")
{
    @desco=split(/\//,$patch1);
    $desco1=$desco[$#desco-1];            #Nombre de aplicacion

        @descomprime=`tar -xvzf $patch1`;#Descomprime parche
        @dirdes=split(/\//,$descomprime[0]);
        $directoriodesco=$dirdes[0];
        #$borra=`rm $patch1`;                    #Borra comprimido
        chdir($directoriodesco);
        $p=`pwd`;
        print "Cambio de directorio a $p\n";
        print "* Instalando parche...\n";
        print "* Verificando dependencias para configuracion...\n";
        $conf=`./configure`;
        print "* Compilando...\n";
        $make1=`make`;
        print "* Instalando ejecutables...\n";
        $make2=`make install`;
}
```

El código completo se presenta a continuación:

```
#!/usr/bin/perl
use XML::XPath;
use XML::XPath::XMLParser;

#########Impresion en bitacora y pantalla#######################
sub impri
{
    my ($tex)= @_;
    my ($s, $m, $h)=localtime;      #Obtiene hora actual
    print "\b"x79, " "x79, "\b"x79,
         sprintf("[%02d:%02d:%02d]", $h,$m,$s),
         "\t", $tex, "\n";
    impri_log($tex);            #Llama funcion de bitacora
}
#########Imprime solo en bitacora ############################
sub impri_log
{
    $f_log = "/var/log/Ovpatch.log";     #Define bitacora
    my ($tex)= @_;
    open FLOG, ">> $f_log"
         or die "No es posible abrir archivo de bitacora";
    print FLOG "[", scalar(localtime), "] \t $tex\n";
```

```
    close FLOG;
}
#########Iniciar bitacora#####################################
impri_log("-----------\t   Iniciando Ovpatch \t ---------\n");

#########Obtener informacion del sistema#######################
###############Aplicaciones instaladas#########################
@aplicaciones=`dpkg --get-selections`;  #Obtiene ap instaladas
impri("* Obteniendo informacion del sistema
        [Aplicaciones]...\n");
my $poss=0;
foreach $apli (@aplicaciones)
{
    @aplicaciones2=split(/\t/,$apli);
    $aplicaciones3[$poss]=$aplicaciones2[0];  #Todas las apli
    $poss++;
}

#################Sistema operativo##############################
my $so=`head -n1 /etc/issue`;            #Obtiene version de SO
impri("* Obteniendo informacion del sistema [Sistema
        Operativo]...\n");
my @so1=split(' ',$so);
my $so2= $so1[0]." ".$so1[1]." ".$so1[2];#Contiene SO

#################Crear entorno de trabajo#####################
my $archivo_xml="unix.xml";
my $nombre_dir=$ARGV[0];    #Asigna a var parametro de usuario

while(-e $nombre_dir)       #si existe el directorio pide otro
{
   impri("El directorio sobre el que desea trabajar ya existe.
            Especifique otro:\n");
   $nombre_dir=<STDIN>;      #Se asigna a lect teclado
   chop($nombre_dir);        #Se elimina el retorno de carro
}

`mkdir $nombre_dir`;        #Crea directorio de trabajo
impri( "* Creando directorio $nombre_dir...\n");
chdir($nombre_dir);          #Cambia de directorio de trabajo
$diract=`pwd`;
chop ($diract);
impri( "* Cambio de directorio de trabajo a $diract...\n");
if(-e $archivo_xml)         #si existe archivo xml lo borra
{
```

```
   impri( "* El esquema ya existe, se actualizara...\n");
   my $borra=`rm $archivo_xml`;
}

##################Descarga el esquema############################
impri( "* Descargando el esquema...\n");
`wget -nv http://oval.mitre.org/rep-
        data/org.mitre.oval/p/family/unix.xml`;
##################Analisis del esquema###########################
impri( "* Analizando el esquema...\n");
my $cont=0;
my $arbol= XML::XPath->new(filename =>$archivo_xml); #n objeto
#Sigue ruta
my $nodos= $arbol-
>find('oval_definitions/definitions/definition');
my $contador=0;
foreach my $elemento ($nodos->get_nodelist)
{

 #Almacena en arreglo
 $producto[$cont]=XML::XPath::XMLParser::as_string($elemento);
 $cont++;
}
impri( "* Comparando esquema con caracteristicas del
        sistema...\n");
foreach $pr (@producto)
{
    my @linea=split(/\n/,$pr);          #Separa por renglones
    my @plataforma1,@s2,@s3,@s4,@s5,@s6,@pl2=();   #Limpia
    my $count=0;
    foreach my $i(@linea)               #Para cada linea
    {
       if ($i =~ /<platform>/) #Si la línea tiene plataforma
       {
           $pl2[$count]=$i;          #Guarda en un arreglo
           $count++;
       }
    }
    #Separa los elementos
    my @platform1=split(/<platform>/,$pl2[0]);
    my @platform2=split(/<\/platform>/,$platform1[1]);  #SO 1
    my @platform3=split(/<platform>/,$pl2[1]);
    my @platform4=split(/<\/platform>/,$platform3[1]);  #SO 2
    my @product1=split(/<product>/,$pr);
    my @product2=split(/<\/product>/,$product1[1]);     #Aplic
```

```
   my @title1=split(/<title>/,$pr);
   my @title2=split(/<\/title>/,$title1[1]);  #vulnerabilidad
   my @description1=split(/<description>/,$pr);
   my @description2=split(/<\/description>/,$description1[1]);
   my @reference1=split(/<reference/,$pr);    #URL de parches
   my @reference2=split(/"/,$reference1[1]);

##################Comparaciones#################################
##################Sistema operativo#############################
   my $eqos=0;
   my $eqapl=0;
   if ($platform2[0] eq $so2 or $platform4[0] eq $so2) #cmp
   {
       $eqos=1;                          #Activa bandera
   }

###############Aplicaciones#####################################
   foreach $elemento (@aplicaciones3)         #Cmp aplicaciones
   {
       if ($product2[0] eq $elemento)
       {
           $eqapl=1;                         #Activa bandera
       }
   }
   ###############Elementos que apliquen#######################
   if ($eqos eq 1 and $eqapl eq 1)  #Si ambas banderas 1
   {
        $referencias[$contador]=$reference2[3];  #Almacena
        $contador++;
   }
}
##################Obteniendo nombre de paquetes###############
my $contt=0;
foreach $elem (@referencias)
{
    my @nom_arch=split(/\//,$elem);
    $nom_arch1[$contt]=$nom_arch[$#nom_arch];
    $contt++;
}
##################Descarga de referencias#####################
impri( "* Descargando referencias...\n");
foreach $elem (@referencias)
{
    foreach $elem2 (@nom_arch1)
    {
```

```
        if(-e $elem2 and $elem =~ $elem2)     #si existe borra
        {
            impri( "* La referencia ya existe, se
                  actualizara...\n");
            my $borra2=`rm $elem2`;
            break;
        }
    }
    `wget -nv $elem`;
}
##################Parseo de referencias#######################
impri( "* Analizando referencias y descargando parches...\n");
foreach $elemento (@nom_arch1)
{
    &html($elemento);
}

sub html
{
     my $contador2=0;
     my $flag=0;
     my $cont3=0;
     my $encontrado=0;

     open(HTML, $_[0]);              #Abre archivo de referencia
     $descriptor, @leehtml, @paq=();
     while (my $descriptor= <HTML>)      #Para cada linea
     {
            $leehtml[$contador2]=$descriptor;   #Almacena
            $contador2++;
            if($descriptor =~ /$so2/)    #Se busca linea
            {
             my $poss2=$contador2;  #Asigna posicion
            }
     }
     foreach $linea (@leehtml)      #Analiza en el arreglo
     {
        if ($encontrado eq 0)  #Si aun no se ha encontrado
        {
            $cont3++;                      #Contador de lineas
            @paq=();                       #Limpia el arreglo
            #Elige linea con tar.gz despues de poss2
            if($cont3>$poss2 and $linea =~ /.tar.gz/)
            {
               my @paq=split(/"/,$linea); #Filtra parte util
```

```
$paquete= $paq[1];

my @patch=split(/\//,$paquete);   #Obtiene nombre
my $patch1=$patch[$#patch];

@desco=split(/\//,$patch1);
$desco1=$desco[$#desco-1];       #Aplicacion

if(-e $desco1)     #si existe archivo lo borra
{
    impri( "* El parche ya existe, se
           actualizara...\n");
    my $borra3=`rm $desco1`;
}
impri( "* Descargando parche $paquete...\n");
`wget -nv $paquete`;  #Descarga parche
my $ban1=0;

while ($ban1 eq 0)
{
      impri( "Desea instalar el parche $patch
           [Si/No]\n");
      $respuesta1=<STDIN>;
      chop($respuesta1); #Se elimina el r de carro
      impri ("Respuesta de usuario:
             $respuesta1\n");
       if ($respuesta1 ne "Si" and $respuesta1 ne
           "si" and $respuesta1 ne "No" and
           $respuesta1 ne "no")
      {
          impri( "Respuesta incorrecta. Ingrese Si
                [si] o No [no]\n");
      }
      else
      {
         $ban1=1;
       }
 }
 if($respuesta1 eq "Si" or $respuesta1 eq "si")
 {
       @desco=split(/\//,$patch1);
       $desco1=$desco[$#desco-1];     #Aplicacion
       @descomprime=`tar -xvzf $patch1`;#Descompr
       @dirdes=split(/\//,$descomprime[0]);
       $directoriodesco=$dirdes[0];
```

```
                chdir($directoriodesco);
                $p=`pwd`;
                impri( "Cambio de directorio a $p\n");
                impri( "* Instalando parche...\n");
                impri( "* Verificando dependencias para
                      configuracion...\n");
                $conf=`./configure`;
                impri("$conf\n");
                impri( "* Compilando...\n");
                $make1=`make &> tmp`;
                `cat tmp>>/var/log/Ovpatch.log`;
                my $impr2=`more tmp`;
                print "$impr2\n";
                impri( "* Instalando ejecutables...\n");
                $make2=`make install &> tmp2`;
                `more tmp2>>/var/log/Ovpatch.log`;
                my $impr=`more tmp2`;
                print "$impr\n";
                `rm tmp tmp2`;
            }
            else
            {
                break;
            }
             chdir($diract);
             $p2=`pwd`;
            $encontrado=1;              #Activa bandera
        }                              #Cierra if .tar.gz
    }                                  #Cierra if encontrado
    else
    {
        break;                      #Sale una vez encontrada linea
    }
  }                              #Cierra foreach
}                                #Cierra sub
```

3.3.2 Entradas

Al definir el sistema como automático, realiza la mayor parte de los procesos sin interacción con el usuario.

La aplicación es presentada como un comando que se ejecuta de la siguiente forma:

```
debian:/home/debian/Desktop# ./Ovpatch.pl directorio_parches
```

Donde se indica el nombre de la aplicación y se da como parámetro el nombre del directorio que se creará para la ejecución.

Se ha decidido indicar el nombre del directorio de trabajo para proporcionar al usuario cierto control sobre la localización de los archivos descargados.

En caso de indicar un nombre de directorio que ya exista, se solicitará al usuario indicar un nuevo nombre para el directorio:

```
[18:09:39]      El directorio sobre el que desea trabajar ya existe. Especifique otro:
```

Una vez que el usuario ejecuta la aplicación, no realiza interacción con él, hasta el momento de indicar al sistema si el parche será instalado o no, como se ha mencionado anteriormente, es importante que el administrador decida sobre la instalación del parche ya que debe haber el proceso previo de evaluación para medir el impacto que tendrá su aplicación.

```
[17:14:03]      Desea instalar el parche  [Si/No]
```

La interacción del usuario para decidir sobre la instalación de un parche será iterativa de acuerdo al número de parches que coincidan con el sistema.

Independientemente de lo ya citado, no es necesario proporcionar al sistema más información.

3.3.3 Salidas

Una vez que se ha ejecutado la aplicación, se obtendrá como resultado la instalación de parches que hayan cumplido con las características del sistema.

Es importante mencionar que debido a la diversidad de requerimientos del sistema previos a la instalación de cada parche, algunos de los parches no serán instalados dejando únicamente registro en una bitácora para su posterior revisión por parte del administrador.

Cada operación que realice la aplicación será registrada en una bitácora almacenada en el mismo directorio creado para la ejecución de la aplicación.

Al finalizar la ejecución, se tendrá un directorio con los parches aplicables al sistema, podrá ser útil como un repositorio de parches y también, en caso de que debido a los requerimientos del sistema no se haya instalado el parche, el administrador puede recurrir a este directorio para instalar los parches de forma manual. Este directorio también puede ser útil para evitar la descarga continua del mismo parche para distintos sistemas, ya que en caso de contar con sistemas con las mismas características no es necesario realizar la descarga repetidas ocasiones.

Es posible visualizar el proceso conforme va ejecutándose la aplicación, obteniendo en pantalla un resultado como el mostrado en la figura 3.1

```
debian:/home/debian/Desktop# perl Ovpatch.pl directorio_parches
[18:12:56]      * Obteniendo informacion del sistema [Aplicaciones]...

[18:12:56]      * Obteniendo informacion del sistema [Sistema Operativo]...

[18:12:56]      * Creando directorio directorio_parches...

[18:12:56]      * Cambio de directorio de trabajo a /home/debian/Desktop/directorio_parches...

[18:12:56]      * Descargando el esquema...

2011-02-08 18:15:08 URL:http://oval.mitre.org/rep-data/org.mitre.oval/p/family/unix.xml [6824418/
6824418] -> "unix.xml" [1]
[18:15:08]      * Analizando el esquema...
```

Figura 3.1 Resultado de ejecución de la aplicación

3.4 PRUEBAS

Esta fase en teoría es la última, pero el desarrollo de una aplicación en realidad es un ciclo constante para tener un adecuado mantenimiento.

Una vez que se tiene la implementación de la aplicación, se realizarán un conjunto de pruebas para verificar el correcto funcionamiento.

El tipo de pruebas que se realizará serán más que nada funcionales, por el momento se dejará a un lado la parte de rendimiento y comportamiento en ambientes como un gran número de maquinas ya que para este tipo de ambientes es necesario implementar otro tipo de soluciones, como sería el caso de un servidor de parches.

Esta fase, es en concreto una etapa de pruebas específicas ya que durante todo el desarrollo de la aplicación se han llevado a cabo distintos tipos de pruebas para evaluar que la aplicación sea desarrollada de forma adecuada de acuerdo con los objetivos planteados.

3.4.1 Criterios de revisión

Una vez concluida la aplicación, se realizarán distintas pruebas dentro de las que están incluidos en los siguientes aspectos:

- Directorio creado correctamente
- Si el directorio existe, solicitud de nuevo nombre
- Verificación del sistema operativo coincidente con el almacenado
- Descarga del esquema de forma adecuada
- Correspondencia del esquema de acuerdo características del sistema
- Verificación de aplicaciones, coincidencia con las aplicaciones instaladas en el sistema
- En caso de existencia de archivo, eliminación
- Descarga de archivo html
- Creación adecuada de bitácoras
- Descompresión adecuada de archivos
- Inserción de valores válidos

Dentro de las pruebas se verificará la respuesta del sistema ante distintos escenarios, se realizará la evaluación de los puntos anteriormente mencionados verificando el sistema de cómo respuesta la salida esperada.

3.4.2 Validación

De acuerdo a los criterios de revisión las pruebas se realizaron, identificando los puntos vulnerables y corrigiéndolos, estos fueron los resultados.

- Directorio creado correctamente: Se verificó que una vez pasado este punto en el programa, se haya creado correctamente, el directorio, con el nombre y en la localización correcta.

- Si el directorio existe, solicitud de nuevo nombre: Se introdujo el nombre de un directorio existente, verificando que el sistema pidiera al usuario un nuevo nombre de directorio.

- Verificación del sistema operativo coincidente con el almacenado: Se verificó que la variable que contenía la versión del sistema operativo tuviera el valor correcto en base al sistema en evaluación.

- Descarga del esquema de forma adecuada: Una vez que se ha pasado la parte del proceso en la que se realiza la descarga del esquema, se verifica, que el esquema sea el mismo que el publicado en internet por OVAL.
- Correspondencia del esquema de acuerdo características del sistema: Se verifica que el esquema descargado coincida con el sistema a evaluar, es decir Debian.

- Verificación de aplicaciones, coincidencia con las aplicaciones instaladas en el sistema: verificar que las aplicaciones instaladas en el sistema, correspondan a las almacenadas por el arreglo que maneja las aplicaciones en el programa.

- En caso de existencia de archivo, eliminación: se realiza la prueba, insertando archivos existentes y verificando que realmente se eliminen y muestre el mensaje de eliminación en pantalla al usuario.

- Descarga de archivo html: se verifica que los archivos html descargados se crean los correspondientes a la vulnerabilidad específica, haciendo la correspondencia con el abierto por algún navegador.

- Creación adecuada de bitácoras: se verifica que las bitácoras sean correctamente creadas y que contengan las acciones realizadas por el programa.

- Descompresión adecuada de archivos: se verifica que la carpeta creada de descompresión contenga los archivos correctos.

- Inserción de valores válidos: se inserta al sistema valores no válidos observando su comportamiento.

CAPITULO IV

EVOLUCIÓN

4. EVOLUCIÓN

4.1 CAMPO DE APLICACIÓN

Hasta el momento, la aplicación ha sido limitada a sistemas Debian, existe gran similitud entre algunas distribuciones Linux sin embargo hay diferencias en cuanto a comandos, lo que origina que el campo de aplicación esté limitado a un sistema especifico.

Presenta límites debido a que la programación hace uso de comandos específicos de sistemas Debian para realizar distintas tareas durante su ejecución, como descarga de paquetes, obtención de información del sistema, instalación de aplicaciones, etc.

A pesar de lo mencionado anteriormente, se debe tomar en cuenta que la base será la misma para cualquier sistema Linux, bastará con la creación de subrutinas en la parte de divergencias entre sistemas para poder ampliar su aplicación.

Otro punto fundamental para aumentar el campo de aplicación, son los esquemas OVAL, existe una división por sistemas operativos en los repositorios OVAL, en este caso, debido a que el campo de aplicación está estrictamente definido para Debian, se ha realizado la descarga únicamente de este esquema. Para ampliar el campo de aplicación se puede recurrir a dos opciones:

- Existe un esquema general que incluye todos los sistemas operativos, esta podría representar la mejor opción en caso de ser aplicable a todos los sistemas, sin embargo puede representar un bajo rendimiento en varios aspectos si solo se ocupa parte del esquema, debido al tamaño por lo que aumentaría el tiempo tanto en descargas como en el análisis para encontrar los elementos útiles.

- Otra opción podría representar el trabajo sobre cada esquema de acuerdo al sistema aplicable, aumentaría la eficiencia si se trata solo de algunos sistemas con los cuales trabajará.

4.2 COMUNICACIÓN CON HERRAMIENTAS COMPATIBLES

Otro paso hacia la evolución de la aplicación es la comunicación que pueda tener con otras herramientas.

Hasta el momento, puede ser tomado como un repositorio de parches, evitando la saturación del ancho de banda de la red, evitando también que todas las maquinas presentes en el sistema realicen la descarga de cada parche necesario, es importante en este punto mantener el servidor de parches o repositorio de parches con un control de acceso adecuado ya que en caso de manipular los parches e inyectarles malware se realizaría una infección de nivel de red ya que cada parche en el repositorio será distribuido en cada máquina. De acuerdo con el documento publicado por el NIST de la administración de parches, es necesaria la realización de una evaluación previa a la instalación de los parches, evaluando el nivel de riesgo ante la aplicación.

Una opción más es el manejo de OVALDI una herramienta que interpreta los esquemas OVAL y arroja resultados en base al sistema evaluado, para que realice la evaluación del equipo automáticamente, dando como resultado un archivo en formato HTML con los resultados obtenidos, sería posible tomar como base ese programa manteniendo mayor compatibilidad entre sistemas y partiendo de él realizar la descarga e instalación de parches.

La aplicación desarrollada hasta el momento solamente cuenta con una interfaz a modo comando, para lograr su desarrollo y mejora podría implementarse una interfaz gráfica de ayuda al administrador, ofreciendo disponibilidad a usuarios menos experimentados. O podría, mediante el uso de alguna herramienta, mejorar la presentación.

Otro punto importante es la entrega de resultados, podría implementarse mediante un módulo de Perl la creación de documentos HTML para una mejor visualización.

4.3 PERSPECTIVAS

La éste punto incluye la visión a futuro que se tiene de la aplicación, existen algunas mejoras independientemente de las mencionadas anteriormente que pueden realizarse a la aplicación.

Se puede comenzar mencionando la parte de la instalación previa de bibliotecas necesarias para la instalación del parche, sería posible verificar previamente lo necesario para la correcta instalación de cada parche, y en base a eso realiza la instalación automática de cada biblioteca. Este punto es un poco complicado debido a la diversidad en cuanto a instalación se refiere de cada biblioteca, pero sería posible formar patrones y en base a la instalación de distintos parches.

Otro punto que ya se ha mencionado anteriormente pero que es de suma importancia para la fácil visualización de datos y resultados es la parte de la interfaz, es necesario que se muestren los datos de una manera más clara para que de esta forma sean fácilmente filtrados por cada administrador, esta parte incluye no solamente la interfaz para la ejecución del herramienta, también incluye la parte de resultados.

Podría agregarse una funcionalidad más que es empleada en distintas herramientas que hacen búsqueda de vulnerabilidades en los sistemas, y es la parte de envío mediante correo electrónico de alertas al administrador. Aquí podría hacerse una clasificación de vulnerabilidades enviando solamente las más urgentes o peligrosas.

Otra parte que es importante y que por el momento se ha resuelto mediante la eliminación y descarga de los archivos cada vez que se ejecuta la aplicación, es la parte de integridad de los datos, existe una firma MD5 disponible vía internet, en versiones futuras, se podrían realizar una verificación de integridad mediante MD5.

CONCLUSIONES

Con el término del presente trabajo se ha logrado la creación de una aplicación que actualiza de manera automática vulnerabilidades en sistemas operativos Linux/Debian.

Se ha descrito la forma de trabajo de OVAL para así tener una visión más clara sobre lo que la aplicación realizará y la forma en la que trabajará.

Se realizaron pruebas identificando los problemas de la aplicación y resolviéndolos de acuerdo a los objetivos iniciales.

Con la creación de la aplicación:

- Se ha logrado la reducción de tiempo en la aplicación de parches.
- Se disminuyeron los riesgos presentes en el sistema.
- Se disminuyeron las vulnerabilidades presentes en el sistema.
- Se ha facilitado al administrador la evaluación y corrección del sistema.
- Se ha logrado la disminución de costos en aplicación de parches.
- Se ha automatizado la labor del administrador al instalar parches.
- Se mantiene un registro de los cambios que sufre el sistema al instalar parches.
- Se mantiene el sistema actualizado de acuerdo a la publicación de parches OVAL.

Aún hay muchos detalles que resolver, la creación de esta herramienta es una muestra de cómo los administradores de acuerdo a necesidades, pueden crear aplicaciones para facilitar sus tareas, esta herramienta puede mutar para convertirse en una herramienta más completa, ya que está desarrollada en un lenguaje de programación con gran poder.

Se exhorta al lector al desarrollo de herramientas que faciliten las tareas cotidianas.

REFERENCIAS

- [R1] DAVARA RODRÍGUEZ, Miguel Ángel. *De las Autopistas de la Información a la Sociedad Virtual*. Editorial Aranzadi, 1996.

- [R2] TELLEZ VALDÉS, Julio. *Los Delitos informáticos. Situación en México*. Informática y Derecho Nº 9, 10 y 11, UNED, Centro Regional de Extremadura, Mérida, 1996.

- [R3] ROMEO CASABONA, Carlos María. *Los llamados delitos informáticos*. Revista de Informática y Derecho, UNED, Centro Regional de Extremadura, Mérida, 1995.

- [R4] ACURIO DEL PINO, Santiago. *Delitos Informáticos Generalidades.*

- [R5] PFLEEGER, Charles. *Security in Computing.* Estados Unidos de América. Editorial Prentice Hall. Segunda Edición. 2000.

- [R6] OPPLEMAN, Victor. *Extreme Exploits*. Madrid. Editorial ANAYA Multimedia. 2005.

- [R7] MITRE History. http://www.mitre.org. 2009.

- [R8] OVAL Repository. http://oval.mitre.org/.

- [R9] DEBRA Littlejohn Shinder. *Scene of the Cybercrime: Computer Forensics Handbook.* Estados Unidos de América. Editorial Syngress Shinder books. 2002.
- [R10] DEBRA Littlejohn Shinder. *Prevención y detección de delitos informáticos.* España. Editorial ANAYA Multimedia. 2002.

- [R11] *Introduction to the OVAL Lenguage*, Version 5.0 http://oval.mitre.org/about/documents.html

- [R12] *OVAL Language Requirements*, Version 5.0 http://oval.mitre.org/about/documents.html

- [R13] *OVAL Design Document*, Version 5.0 http://oval.mitre.org/about/documents.html

- [R14] PETER Mell. *NIST Publication 800-40 Creating a Patch and Vulnerability Management Program. Recommendations of the National Institute of Standards and Technology (NIST).* Version 2.0. 2005.