Facilitando la administración de la seguridad en tu red DMZ: MatFel

Francisco Javier Díaz, Einar Felipe Lanfranco, Matías Pagano, Paula Venosa
{javierd,einar,matiasp,pvenosa}@linti.unlp.edu.ar
LINTI - Facultad de Informática - Universidad Nacional de La Plata
La Plata, B1900ASD, ARGENTINA

***Abstract*—Este documento describe el esfuerzo realizado para lograr la integración de un conjunto de aplicaciones de seguridad para redes de datos, centralizando su administración y monitoreo. Involucrando Firewall, un Sistema de Detección de Intrusiones y un Análizador de Vulnerabilidades, el resultado obtenido es una aplicación web para facilitar la gestión de seguridad que respeta los estándares actuales de desarrollo promulgados por la W3C[1].**

Se detalla en este trabajo el proceso completo de desarrollo de MatFel, relatando desde la selección de los componentes a integrar hasta las características principales del producto final.

***Index Terms*—Integración Software Libre Seguridad Privacidad Redes**

I. Motivación

El crecimiento exponencial de Internet genera que hoy en día la cantidad de servicios de red prestados por las organizaciones y el acceso a los mismos sea cada vez mayor. Ello hace que asegurar dichos servicios sea un objetivo impostergable para las organizaciones, que comienzan a entender la importancia de incluir a la seguridad como un objetivo de negocios.

Una de las dificultades que se presentan para la administración de la seguridad es que, si bien existe una gran cantidad de herramientas para implementar soluciones de seguridad, cada una de ellas se concentra en un tipo específico de problema o en un determinado aspecto de seguridad, y tanto la información que genera como su administración es independiente de las otras herramientas. Otras, como OSSIM[2], si bien son un ejemplo de integración de herramientas de seguridad y monitoreo, ofrecen solo la posibilidad de observar el estado tanto de la disponibilidad como de la seguridad de la red, pero no brindan un mecanismo que permita responder ante un evento adverso, y mucho menos de manera simple y sin la necesidad de tener que utilizar herramientas adicionales.

Además muchos de estos productos no son open source, con lo que no resulta posible adaptarlos o mejorarlos, quedando los cosumidores dependientes del fabricante. Sumándose a esto que el usuario tiene que asumir el costo de licencia si se trata de un producto comercial.

II. Introducción

El desarrollo de MatFel comenzó en el marco de la realización del trabajo de fin de carrera de la Licenciatura en Informática de la Facultad de Informática de la Universidad Nacional de La Plata[3], de los actuales egresados Einar Felipe Lanfranco y Matías Pagano. Y si bien la tesina fue presentada a fines de 2010, actualmente se continúa con el desarrollo, atravesando hoy en día el proceso de publicación como software libre, teniendo como meta inicial su publicación en Sourceforge[4].

Desde el inicio la idea fue integrar un conjunto de herramientas para seguridad en redes, las cuales debían estar distribuidas como software libre. Y aunque el objetivo planteado originalmente para la tesina era realizar un simple prototipo, el desarrollo fue avanzando hasta llegar a convertirse en un complejo sistema web con mucha funcionalidad.

El código de MatFel está desarrollado con Perl[5] como lenguaje principal pero utilizando el framework de desarrollo Catalyst, por lo que las adaptaciones, correciones y expansiones son relativamente sencillas de realizarse.

III. Componentes a Integrar

Si bien se había definido que los componentes a integrar eran,como se mencionó anteriormente un Firewall, un IDS y un Analizador de vulnerabilidades, había que elegir que implementación de cada uno se iba a utilizar y sobre qué plataforma iban a ejecutarse. Para ello se establecieron una serie de pautas que nos permitieran seleccionar entre las múltiples opciones disponibles.

A. Criterios de Selección

Los criterios establecidos fueron los siguientes:

- La licencia con la que se distribuye el software
- La comunidad que lo utiliza
- La documentación existente
- Las pruebas de integración realizadas
- La facilidad de instalación
- El idioma
- Nuestra experiencia previa en el uso de los mismos

1) La licencia: La licencia[6] bajo la que se distribuye el software a utilizar fue el único criterio ineludible de aplicar, ya que si se elegían programas cuya licencia implique algún costo para su uso o genere alguna prohibición de uso y/o lectura y/o modificación del código fuente hubiera significado un escollo importante.

Si la desventaja sólo fuera de costo sería tal vez un problema eludible, ya que pagando la licencia el problema se soluciona, pero sigue sin cumplirse uno de los objetivos de la presente

implementación, que es un costo de licenciamiento de cero pesos.

Ahora, si existiera algún impedimento de uso, lectura o modificación el problema sería más importante. Si es de uso, como ser por ejemplo que no esté permitido usar la salida del producto en otro sistema se hubiera complicado mucho la integración, pero si la restricción fuera de lectura o modificación de los fuentes sería aún peor. El escenario ideal para nuestro trabajo es que la licencia que tienen los productos respeten la mayor cantidad de puntos que establece la Fundación del software libre[7] así como las libertades básicas.

2) La comunidad y la documentación: Tanto la comunidad de desarrolladores como la comunidad de usuarios que usan cada uno de los productos es muy importante para el proceso de selección.

La comunidad de desarrolladores es importante porque representa el indicador del soporte y la actualidad que tiene un sistema, cuanto más desarrolladores haya detrás más fácil será conseguir la solución a los problemas que aparezcan y más difícil será que el desarrollo se estanque o "muera" al quedarse sin gente detrás que lo continúe. ¿De qué nos sirve un analizador de vulnerabilidades si nadie desarrolla nuevos plugins para detectar vulnerabilidades?

La comunidad de usuarios que hay detrás del software es también muy importante por varios factores. Se puede citar por ejemplo que representan un indicador de la aceptación del producto, o que al haber más usuarios se genera más documentación ya que hay más consultas y por lo tanto más respuestas, o que muchas veces en la comunidad del software libre los mismos usuarios de los productos se convierten de alguna manera en parte del equipo de desarrollo al aportar ideas, documentación, testing, problemas y/o soluciones.

3) Las pruebas de integración realizadas: Se realizaron pruebas de integración. La posibilidad de integración de cada uno de los componentes observada en dichas pruebas fue un factor determinante para saber si era factible de ser elegido o debía ser descartado de inmediato. Para poder integrar software se requiere que lo que produce un programa pueda ser usado por el otro. Por este motivo es que si no se podía lograr que la salida de un producto X fuera interpretado de manera de poder utilizarse en MatFel, X era dejado de lado.

4) La facilidad de instalación: Como resultaba deseable que una vez terminado el desarrollo, se pudiera hacer puestas en producción con cierta facilidad, otro de los criterios importantes a tener en cuenta es cuán difícil es poder instalar y configurar el producto candidato a integrar. Esto se tuvo en cuenta tanto en la instalación de un producto por separado como para evaluar su convivencia con el resto del sistema, ya que suele encontrarse software que aislado funciona sin mayores problemas pero cuando tiene que convivir con otras aplicaciones interfiere drásticamente con el resto.

IV. Experiencia previa propia

Al momento de elegir tanto las herramientas que se integraron como las herramientas que se usaron para el desarrollo se le dió un peso significativo a la experiencia previa de los desarrolladores de la tesina. Se priorizó utilizar los ambientes de trabajo, lenguajes de desarrollo y aplicaciones con los que ellos trabaja habitualmente y se sienten más cómodos, dado que su uso minimiza la complejidad de resolver determinados problemas.

Para comprender de qué se habla cuando se cita la experiencia se incluye a continuación un resumen de su historia en relación con el software libre y en el desarrollo web:

"Si bien ambos venimos del mundo privativo y comercial del software, nuestra educación universitaria se basó en un altísimo porcentaje en la utilización de productos Microsoft como base (nos animaríamos a decir que un 98%). Gracias a nuestras primeras experiencias laborales nos tuvimos que introducir en el mundo de GNU/Linux, primero usándolo en servidores y de a poquito, con el paso del tiempo, reemplazando todo lo privativo que usábamos tanto en el trabajo como en nuestros hogares, hasta reemplazarlo hoy casi en su totalidad, incluso en nuestros celulares hemos realizado algunas pruebas con Android[8].

En el camino, más allá de ser usuarios de una gran variedad de productos de libres todos los días, nos hemos vuelto desarrolladores de proyectos de software libre, participando de proyectos como Lihuen GNU/Linux[9] y KOHA-UNLP[10]. También nos hemos vuelto predicadores, asistimos a charlas en colegios y en conferencias, tratamos de difundir 'la palabra' a nuestros amigos, familiares y compañeros de trabajo, e incentivamos el uso de software libre en las cátedras que participamos y tenemos injerencia."

Lo expresado en los últimos párrafos explica el por qué se eligió software libre para el presente desarrollo teniendo en cuenta la experiencia que se posee en este área.

V. Los elegidos

A. El sistema base

Utilizando como primer criterio de selección la licencia de distribución, las opciones se redujeron a algún sistema GNU/Linux o alguno de la rama derivada de BSD[11].

A la hora de elegir entre ellas se analizó la cantidad de aplicaciones que desarrolladas para cada plataforma y se llegó a la conclusión de que existe una cantidad de aplicaciones similar para ambas, y que en general mucho del software GNU desarrollado para Linux está portado a BSD y viceversa, o sea que la mayoría del software libre disponible lo podemos correr tanto en BSD como en GNU/Linux.

Luego se tuvieron en cuenta otros criterios de elección como el idioma, la comunidad que los soporta y la facilidad de instalación. Al evaluar dichos criterios se determinó que las dos opciones no presentaban diferencia debido a que las mismas se distribuyen en diversos idiomas, la facilidad de instalación es similar y cuentan con una enorme comunidad.

Dado este resultado al aplicar los criterios descriptos se decidió tener en cuenta la amplia experiencia personal en GNU/Linux, y seleccionar entonces Debian GNU/Linux[12][1] para el servidor que albergará el desarrollo, y alguna distribución basada en ella, pero orientada a Workstation, para

[1]Debian GNU/Linux: una de las mayores distribuciones de GNU/Linux, http://www.debian.org

utilizar como estación de desarrollo, como por ejemplo Lihuen GNU/Linux[2].

B. El Firewall

Al ser GNU/Linux el ambiente escogido se decidió utilizar el firewall NETFILTER/IPTABLES para el desarrollo, ya que el mismo es un software disponible en prácticamente todas las distribuciones de Linux actuales.

Netfilter[13] es un framework disponible en el kernel Linux que permite interceptar y manipular paquetes de red. Dicho framework permite realizar el manejo de paquetes en diferentes estados del procesamiento.

El componente más popular construído sobre **Netfilter** es **IPtables**, una herramientas de firewall que permite no solamente filtrar paquetes, sino también realizar traducción de direcciones de red (NAT) para IPv4 o mantener registros de log.

C. El Analizador de Vulnerabilidades

A la hora de seleccionar un analizador de vulnerabilidades se tuvo en cuenta también en primer lugar que la plataforma de fuera trabajo GNU/Linux, qué problemas de seguridad la herramienta permite detectar, su funcionalidad y su tipo de licencia.

Se analizaron Nessus y OpenVAS como posibles alternativas:

Nessus[14] es un programa que corre en diversos sistemas operativos cuyo objetivo es detectar las vulnerabilidades potenciales en los sistemas escaneados. Fue el escaneador de vulnerabilidades más popular hasta que cerraron el código en 2005 y quitaron la versión gratis en 2008. Aunque aún está disponible una versión limitada, la licencia permite su uso sólo en redes hogareñas. A pesar de esto, Nessus sigue siendo el mejor escaneador de vulnerabilidades en los sistemas UNIX y está entre los mejores que corren en Windows.

OpenVAS[15] (Open Vulnerability Assessment System) es una herramienta para el análisis de seguridad de una red. Se deriva del proyecto Nessus que se ha convertido en un producto propietario. Todos los productos OpenVAS son libres con licencia GNU General Public License (GNU GPL).

Se seleccionó finalmente OpenVAS por ser el más desarrollado y aceptado en la comunidad de entre los escaneadores de vulnerabilidades existentes y que mantiene la licencia libre.

D. El IDS

Entre los IDS más reconocidos nos encontramos con dos alternativas: BRO-IDS y Snort.

Bro es open source, permite ejecutarse con recursos de hardware mínimos para monitorear redes de altos volúmenes de tráfico, funciona correctamente sobre GNU Linux y presenta facilidades para interrelacionarse directamente con el firewall o con cualquier otro software con el que se quiera relacionar. Si bien Bro tiene la posibilidad de utilizar las reglas de Snort importándolas directamente, tiene una gran desventaja: se necesita un usuario avanzado que esté permanentemente manipulando las políticas para ser efectivo, y su instalación no es del todo automática.

Por su parte Snort, con la excepción de la necesidad de licencia para obtener las reglas más nuevas disponibles, es también software libre. El mismo funciona correctamente sobre GNU/Linux, es muy simple de instalar sobre Debian y presenta facilidades para generar salidas que después puedan procesarse fácilmente con nuestro sistema.

La elección de IDS no resultó tan natural como las decisiones anteriores ya que ambos cumplen con los requisitos planteados y si bien la idea es preparar a el sistema para que pueda configurarse con cualquiera de las dos opciones, en un primer paso se decidió que el sistema se integre con la alertas detectadas por Snort tomando como principal factor de valoración la facilidad de instalación y los plugins existentes para Snort como Barnyard2 que van a simplificar el desarrollo de nuestra solución.

VI. Integración

Ya descriptas en la sección anterior los razones de cada elección vamos a resumir y diremos que concretamente se integraron los siguientes productos:

- El firewall de GNU/Linux Iptables
- El IDS Snort con su helper Barnyard2
- El analizador de vulnerabilidades OpenVAS

Permitiendo a partir del sistema resultante que, de forma descentralizada, múltiples usuarios puedan:

- Hacer un seguimiento del estado en que se encuentran sus servidores.
- Solicitar habilitaciones y denegaciones de permisos para el tráfico desde y hacia sus servidores.
- Visualizar si existe tráfico sospechoso saliendo o llegando a sus equipos.

Toda esta gestión se realiza a través de una cómoda y simple interfaz web a la que el usuario del sistema puede acceder utilizando el navegador web que desee. En forma gráfica puede observarse en la figura 1.

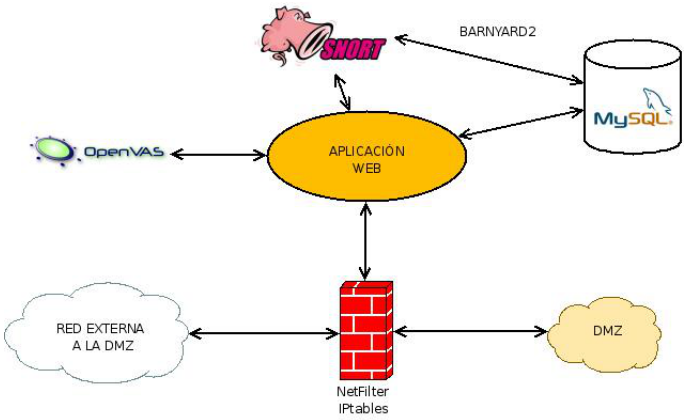


Fig. 1. Integración de las componentes

[2]Lihuen GNU/Linux: distribución GNU/Linux basada en Debian desarrollada en la Facultad de Informática de la UNLP, http://www.lihuen.info.unlp.edu.ar

VII. Cómo funciona internamente MatFel

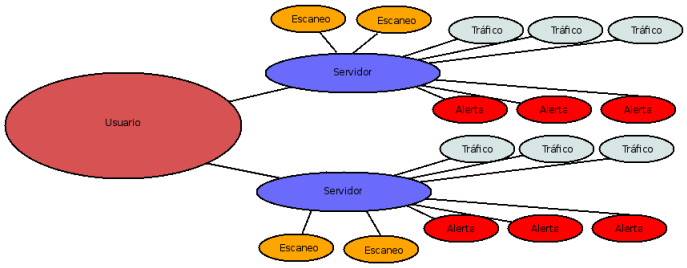
Para describir cómo funciona MatFel empezaremos describiendo los objetos internos que se manejan y la relación entre ellos.

Matfel tiene usuarios registrados en el sistema, cada uno con un rol particular.

Donde cada usuario tiene un conjunto de servidores asociados, en los cuales se alojan los servicios que ese usuario controla o administra.

Cada servidor tiene asociado una serie de elementos:

- Solicitudes de escaneos periódicos en busca de vulnerabilidades.
- Reportes de vulnerabilidades a partir de que se realizan los escaneos.
- Permisos de tráfico, por ejemplo permitir tráfico al puerto TCP/80.
- Bloqueos explícitos de tráfico, por ejemplo prohibir el tráfico al puerto TCP/80. Generalmente generados a partir de las alertas o de los reportes de vulnerabilidades.
- Alertas relacionadas a su dirección IP de entre las detectadas por el IDS.



En estos objetos presentados y su interrelación es que MatFel basa su funcionamiento. Entre las funciones más importantes podemos nombrar:

- Solicitar al OpenVas-Server que realice tests y guardar los resultados.
- Solicitar al Snort las alertas que se producjeron para los servidores registrados y guardar los resultados.
- Generar reglas de Iptables para que el Firewall las aplique.
- Presentar un panel de control a los usuarios que es gráfico, simple y agradable.
- Presentar a los usuarios un reporte de seguridad sobre sus equipos, incluyendo alertas, tráfico y resumen de vulnerabilidades.
- Enviar reportes generales por mail.

VIII. Características Internas de MatFel

A. El código

Como ya mencionamos lo que se planteó incialmente como un prototipo terminó siendo un desarrollo de software con cierta complejidad íntegramente desarrollado en Perl, utilizando el Framework Catalyst. En esta sección trataremos daremos detalles del trabajo que involucró el desarrollo.

El sistema resultante respeta el modelo MVC, habiendo utilizado Template Toolkit y FormFu para el View, DBIx para el Model y Catalyst Controller para el Controller.

Las principales librerías en las que se apoyó el desarrollo son:

- Catalyst::Controller[16]
- Catalyst::Controller::HTML::FormFu[17]
- Chart::OFC2 y subclases (Axis,Bar,Line,Pie)[18]
- DBIx::Class[19]
- DateTime
- Jquery[20]
- Json[21]
- Google jsapi[22]

Al tratarse de un desarrollo distribuido entre dos personas, fue necesario utilizar un mecanismo que permitiera la sincronización, dado que ir combinando el desarrollo de forma manual resulta muy complicado, implica un gran desperdicio de tiempo, y se convierte comunmente en una fuente de generación de errores.

El mecanismo elegido fue utilizar un sistema de control de versiones, concretamente optamos por Subversion[23], uno de los sistemas de versionado bajo licencia de software libre más difundidos actualmente, muy aceptado y que cuenta con clientes para múltiples plataformas.

Siendo más concretos podemos mencionar que después de más de 340 commits, el proyecto involucra aproximadamente 100 archivos, de los cuales 50 archivos son para el View (1369 líneas de código), 17 archivos son para el Controller (289 líneas de código) y 33 archivos son para el Model (1089 líneas de código). Es decir que se escribieron casi 2800 líneas de código, lo cual si bien no es un indicador que se pueda tomar para evaluar la calidad del trabajo, sirve a la hora de cuantificar el esfuerzo realizado.

B. El diagrama de clases

Se utilizó programación orientada a objetos en el desarrollo. El diagrama de clases no lo vamos a presentar en este documento por una cuestión de espacio, principalmente por el tamaño del mismo. Si lo describimos podemos decir que el diagrama involucra 57 clases definidas por nosotros y 25 clases adicionales entre las propias del framework y las auxiliares.

C. El modelo de la base de datos

Como se mencionó MatFel utiliza una base de datos sobre un motor MySQL[24] en la cual se han definido 39 tablas que manejan los distintos datos, en este esquema de tablas está tanto las que necesita el Barnyard para guardar lo que detecta Snort y que luego nosotros usamos como las tablas que directamente definimos nosotros.

Entender el esquema de base de datos de Snort fue muy complicado ya que no se encontró documentación relacionada a ello por lo cual hubo que hacer ingeniería inversa y a fuerza de prueba y error se consiguió descubrir la información necesaria.

D. El código resultante

A la hora del desarrollo, además de respetar las características mencionadas en los párrafos anteriores de suma importancia para los desarrolladores, se implementó la la

aplicación de manera tal que el código que genera (que es en realidad lo que ve el usuario final, respete los estándares de la W3C[1]. Ello se hizo tanto para el lenguaje de marcado de hipertexto como para las hojas de estilo utilizadas.

Para testear el cumplimiento de los estándares, se usó el validador que tiene disponible la W3C on line[25] que permite verificar si la aplicación desarrollada respeta los estándares o no. Como resultado de dicha evaluación se puede afirmar que el trabajo realizado respeta XHTML 1 en su rama Strict[26] y CSS en su versión 2.1 [27].

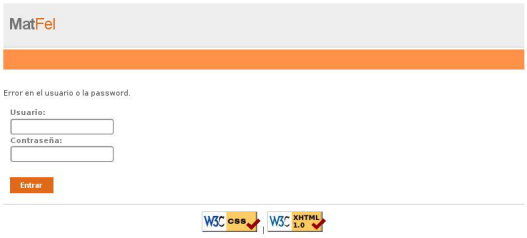


Fig. 2. Login de MatFel

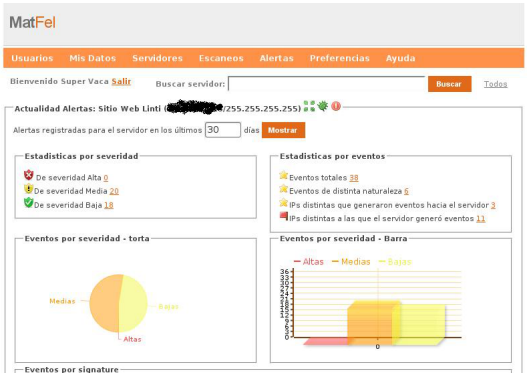


Fig. 3. Resumen de Alertas

Matfel se encuentra actualmente funcionando en la red del LINTI (Laboratorio de Investigación en Nuevas Tecnologías Informáticas) de la Facultad de Informática de la UNLP, en la cual residen alrededor de 90 servidores.

IX. Trabajo a Futuro

Mucho del trabajo planteado inicialmente como tareas a futuro en la presentación de la tesina fue finalmente realizado como parte de la misma. Más allá de eso, siempre surgen nuevas ideas como las que prensentamos a continuación que se suman a las que planeandas incialmente, como ser:

- Expandir la integración a otros productos, como por ejemplo un sensor de tráfico de red.
- Expandir la funcionalidad incorporando temas relacionados a la administración que a la seguridad en sí.
- Agregar funcionalidad como por ejemplo mediante programación de reacciones automáticas.
- Difundir MatFel.
- Solucionar bugs que se descubran en la aplicación.
- Atender los pedidos de la comunidad.
- Generar nueva documentación y mejorar la existente.
- Simplificar aún más el proceso de instalación, empaquetando la aplicación para facilitar su distribución, por ejemplo, para Debian GNU/Linux.

X. Conclusiones

En cuanto a la experiencia del desarrollo fue muy gratificante descubrir que una vez superada la curva de aprendizaje de Catalyst Framework el desarrollo se tornó algo mucho más simple que si se hubiera escrito el código directamente desde cero. Lo mismo ocurre actualmente cuando se incorpora nuevas funcionalidad o se corrigen bugs. Resultando todo en un código fuente que respeta el MVC.

En lo que se refiere a la funcionalidad para la que fue planteado el primer prototipo cubre todas las expectativas ya que simplifica enormemente el trabajo de administración y control de la seguridad de la red. Al integrar las tres herramientas en una sola aplicación web se reduce la inversión de tiempo de los recursos humanos necesaria para chequear alertas y analizar reportes, actualizar configuraciones y manejar scripts de cientos o miles de líneas para administrar un firewall complejo.

Además de las ventajas ya mencionadas encontramos otras cuestiones positivas al finalizar el desarrollo:

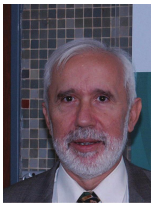
- Se reduce la cantidad de errores humanos que pueden producirse, por ejemplo nadie puede causar un error en las reglas del Firewall que ocasione que todo el sistema deje de funcionar.
- Se brinda una administración descentralizada de los filtros a los servidores, dado que cada usuario puede habilitar o deshabilitar el tráfico para sus servidores, sin la necesidad de depender del administrador del firewalls.
- Se reduce la capacitación necesaria del personal que introduce un cambio, ya que los usuarios no necesitan conocer sintaxis específica y no pueden realizar acciones que generen interferencias con el resto de los usuarios.

Bibliografía

[1] World wide web consortium (W3C). http://www.w3.org/.
[2] Open source security information management. http://www.ossim.net/.
[3] Universidad nacional de la plata - wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/UNLP.
[4] Sourceforge. http://sourceforge.net/.
[5] The perl programming language - www.perl.org. http://www.perl.org/.
[6] Licenses - GNU project - free software foundation. http://www.gnu.org/licenses/licenses.html.
[7] Free software foundation — working together for free software. http://www.fsf.org/.
[8] Android - wikipedia, la enciclopedia libre. http://es.wikipedia.org/wiki/Android.
[9] Sitio oficial de lihuen. http://lihuen.info.unlp.edu.ar/.
[10] Koha UNLP. http://koha.unlp.edu.ar/.
[11] Daemon news - best software review. http://www.daemonnews.org/200104/bsd_family.html.
[12] Debian – el sistema operativo universal. http://www.debian.org/.
[13] netfilter/iptables project homepage - the netfilter.org project. http://www.netfilter.org/.
[14] Tenable network security. http://www.nessus.org/nessus/.
[15] OpenVAS - OpenVAS - open vulnerability assessment system community site. http://www.openvas.org/.
[16] Catalyst::Controller - search.cpan.org. http://search.cpan.org/~bobtfish/Catalyst-Runtime-5.80027/lib/Catalyst/Controller.pm.

[17] Catalyst::Controller::HTML::FormFu - search.cpan.org. http://search.cpan.org/~cfranks/Catalyst-Controller-HTML-FormFu-0.06001/lib/Catalyst/Controller/HTML/FormFu.pm.
[18] Chart::OFC2 - search.cpan.org. http://search.cpan.org/dist/Chart-OFC2/lib/Chart/OFC2.pm.
[19] DBIx::Class - search.cpan.org. http://search.cpan.org/dist/DBIx-Class/lib/DBIx/Class.pm.
[20] jQuery: the write less, do more, JavaScript library. http://jquery.com/.
[21] JSON. http://www.json.org/.
[22] API de google maps - google code. http://code.google.com/intl/es-AR/apis/maps/index.html.
[23] Apache subversion. http://subversion.apache.org/.
[24] MySQL :: The world's most popular open source database. http://www.mysql.com/.
[25] The W3C markup validation service. http://validator.w3.org/.
[26] W3C XHTML2 working group home page. http://www.w3.org/MarkUp/.
[27] Cascading style sheets level 2 revision 1 CSS 2.1 specification. http://www.w3.org/TR/CSS2/.

Francisco Javier Díaz



Es Licenciado en Matemática Aplicada de la Facultad de Ciencias Exactas de la UNLP y Profesor Titular Ordinario de la Facultad de Informática de la UNLP. Es Director General, Científico y Tecnológico del Centro de Procesamiento de la Información (CeSPI) y Director del Laboratorio de Investigación de Nuevas Tecnologías Informáticas (LINTI).

Einar Lanfranco

Es Analista de Computación y Licenciado en Informática de la Facultad de Informática de la UNLP. Se desempeña Jefe de Trabajos Prácticos de la Facultad de Informática y realiza tareas de investigación en el LINTI. Es integrante de la Comisión Directiva del laboratorio. Trabaja en temas relacionados a Seguridad y Software Libre.

Matías Pagano

Es Analista de Computación y Licenciado en Informática de la Facultad de Informática de la UNLP. Se desempeña como Auxiliar Docente de la Facultad de Informática y realiza tareas de investigación en el LINTI. Trabaja en temas relacionados a Desarrollo de aplicaciones y Software Libre.

Paula Venosa

Es Analista de Computación y Licenciada en Informática de la Facultad de Informática de la UNLP. Se desempeña como Profesora Adjunta de la Facultad de Informática y realiza tareas de investigación desde el LINTI. Es integrante de la Comisión Directiva del laboratorio. Trabaja en temas relacionados a Seguridad y Auditoría.