1.5.3 Script de directorios: se programará el script que recorre de forma recursiva y jerárquica los directorios que establece el usuario para analizar cada uno de los archivos que se encuentren bajo los mismos.

1.5.4 Script de archivos: se desarrollará el script que tiene como objetivo el procesamiento de un archivo determinado por el usuario para buscar en él comandos, sentencias y shells que son comúnmente utilizados por backdoors.

1.5.5 Realización de pruebas de software: se realizarán pruebas para determinar la funcionalidad de cada módulo, de la integración de las capas del software y cada componente de las mismas.

1.5.6 Revisión del sprint - retrospectiva: en estas dos reuniones el equipo muestra el trabajo realizado en esta instancia con la participación del cliente si es requerido y, en privado, determinan las mejoras que deberían haber para los futuros sprints.

1.6 Reunión de planificación del sprint: reunión entre los miembros del equipo, Scrum Master y Product Owner donde se establecen criterios para el sprint que comenzará y se genera el sprint backlog. En este espacio se analizan las historias de usuario que se desarrollarán, se discute el estimado en términos de puntos de complejidad y se hace una distribución inicial de las tareas entre los miembros del equipo.

1.6.1 Validación de requisitos: el cliente deberá verificar que los requisitos que ha planteado en un principio estén reflejados en el software producido en su versión final. Cabe aclarar que se requiere de aprobaciones parciales por parte del cliente al finalizar cada sprint.

1.6.2 Validación de software: el cliente debe aprobar que el software en su totalidad satisface sus necesidades y pretensiones. Esto incluye el diseño de interfaces y la facilidad de uso.

1.6.3: Elaboración de manual de usuario: se creará un manual de usuario que contendrá instrucciones para el correcto uso del software. Esto incluye información sobre el producto, alcance, explicaciones de cada módulo y un paso a paso de cómo debe proceder el usuario.

1.6.4 Realización de pruebas de performance: se determinarán los tiempos de CPU, ciclos, uso de memoria RAM y espacio de almacenamiento que requiere el software en distintos ambientes (modificando entornos de escritorio, configuraciones de hardware y distribuciones GNU/Linux). Se desea conocer el desempeño del programa en ambientes realistas de ejecución.

1.6.5 Realización de pruebas de portabilidad: se determinará la capacidad de adaptación del software frente a diferentes distribuciones de GNU/Linux, entornos de escritorio y gestores de ventanas, como así también frente a la ausencia de estos últimos.

1.6.6 Validación de documentación: el cliente deberá aprobar la documentación generada, tanto para el mantenimiento del programa como para su instalación y utilización.

1.6.7 Preparación de evento de presentación: se realizarán las gestiones necesarias para presentar oficialmente el software al cliente y dar por terminado el proyecto. Incluye la implementación del programa en los servidores del cliente y la incorporación del paquete a los repositorios comunitarios.

1.6.8 Revisión del sprint - retrospectiva: en estas dos reuniones el equipo muestra el trabajo realizado en esta instancia con la participación del cliente si es requerido y, en privado, determinan las mejoras que deberían haber para los futuros sprints.

Product Backlog

Sprint	ID	Historia de usuario	Valor
1	1.1.1	Conformación del equipo de desarrollo	5
	1.1.2	Adquisición del hardware	5
	1.1.3	Instalación y configuración de SO	5
	1.1.4	Investigación de .tar.gz en Pacman	5
	1.1.5	Investigación del análisis efectivo en .tar.	8
	1.1.6	Revisión del sprint - retrospectiva	0
			Subtotal: 28
2	1.2.1	Investigación de llamadas en logs de redes	5
	1.2.2	Investigación de puertos usados por backdoors	5
	1.2.3	Investigar sentencias de backdoors	10
	1.2.4	Definición de pila de tecnologías	5
	1.2.5	Instalación de software de desarrollo	3
	1.2.6	Revisión del sprint - retrospectiva	0
			Subtotal: 28
3	1.3.1	Definición de estructura básica del programa	8
	1.3.2	Elaboración de documentación para el desarrollo	8
	1.3.3	Determinación de normas de calidad	3
	1.3.4	Desarrollo de interfaz con el usuario	10
	1.3.5	Revisión del sprint - retrospectiva	0
			Subtotal: 29
4	1.4.1	Script de puertos	8
	1.4.2	Script de procesos	8
	1.4.3	Determinación de dependencias requeridas	5
	1.4.4	Revisión de documentación de los lenguajes	3
	1.4.5	Desarrollo capa de datos	10
	1.4.6	Revisión del sprint - retrospectiva	0
			Subtotal: 34
5	1.5.1	Script de binarios	3
	1.5.2	Script de empaquetados	8
	1.5.3	Script de directorios	8
	1.5.4	Script de archivos	8
	1.5.5	Realización de pruebas de software	5
	1.5.6	Revisión del sprint - retrospectiva	0
			Subtotal: 32

6	1.6.1	Validación de requisitos	5
	1.6.2	Validación de software	5
	1.6.3	Elaboración de manual de usuario	3
	1.6.4	Realización de pruebas de performance	5
	1.6.5	Realización de pruebas de portabilidad	5
	1.6.6	Validación de documentación	5
	1.6.7	Preparación de evento de presentación	3
	1.6.8	Revisión del sprint - retrospectiva	0
			Subtotal: 31
Total:			**182**

Tabla 7. Product Backlog

Diagrama de Gantt

El diagrama de Gantt es una herramienta de planificación para proyectos en donde se establecen fechas y vínculos entre las mismas. La principal utilidad es poder realizar el seguimiento y control de la ejecución de las tareas. Además, permite visualizar el calendario general del proyecto con la fecha prevista de finalización y el impacto que tiene en dicha fecha el retraso que puede haber al desarrollar las tareas.[29]

Las siguientes imágenes proceden del programa *GanttProject* de uso libre y gratuito.

Nombre	Fecha de inicio	Fecha de fin	Duración
Preparación	1/01/16	11/01/16	7
Adquisición de hardware	1/01/16	5/01/16	3
Instalación y configuración de SO	6/01/16	11/01/16	4
Conformación del equipo de desarrollo	1/01/16	8/01/16	6
Investigación	12/01/16	1/02/16	15
Investigar .tar.gz en Pacman	12/01/16	14/01/16	3
Investigar el análisis efectivo en .tar.gz	15/01/16	19/01/16	3
Investigar llamadas de logs de redes	13/01/16	20/01/16	6
Investigar puertos usados por backdoors	14/01/16	21/01/16	6
Investigar sentencias de backdoors	20/01/16	27/01/16	6
Definir pila de tecnologías	28/01/16	1/02/16	3
Pila Tecnológica	2/02/16	12/02/16	9
Instalar software de desarrollo	2/02/16	10/02/16	7
Definir estructura básica del programa	2/02/16	12/02/16	9
Revisar documentación de los lenguajes	2/02/16	8/02/16	5
Análisis	15/02/16	26/02/16	10
Determinar dependencias requeridas	15/02/16	22/02/16	6
Determinar normas de calidad	15/02/16	17/02/16	3
Elaborar documentación para el desarrollo	23/02/16	26/02/16	4
Desarrollo	29/02/16	17/03/16	14
Desarrollar parte basada en llamadas. (Macro)	29/02/16	4/03/16	5
Script de procesos	29/02/16	4/03/16	5
Script de puertos	29/02/16	4/03/16	5
Desarrollar parte basada en búsqueda en archivos. (Macro)	29/02/16	15/03/16	12
Script de archivos	29/02/16	4/03/16	5
Script de binarios	7/03/16	11/03/16	5
Script de directorios	7/03/16	10/03/16	4

Ilustración 14. Diagrama de Gantt listado parte 1

[29] Definición Diagrama de Gantt. http://www.obs-edu.com/blog-project-management/diagramas-de-gantt/que-es-un-diagrama-de-gantt-y-para-que-sirve/. Vigente 17/07/2016.

Nombre	Fecha de inicio	Fecha de fin	Duración
Definir estructura básica del programa	2/02/16	12/02/16	9
Revisar documentación de los lenguajes	2/02/16	8/02/16	5
Análisis	15/02/16	26/02/16	10
Determinar dependencias requeridas	15/02/16	22/02/16	6
Determinar normas de calidad	15/02/16	17/02/16	3
Elaborar documentación para el desarrollo	23/02/16	26/02/16	4
Desarrollo	29/02/16	17/03/16	14
Desarrollar parte basada en llamadas. (Macro)	29/02/16	4/03/16	5
Script de procesos	29/02/16	4/03/16	5
Script de puertos	29/02/16	4/03/16	5
Desarrollar parte basada en búsqueda en archivos. (Macro)	29/02/16	15/03/16	12
Script de archivos	29/02/16	4/03/16	5
Script de binarios	7/03/16	11/03/16	5
Script de directorios	7/03/16	10/03/16	4
Script de empaquetados	7/03/16	15/03/16	7
Desarrollo de interfaz con el usuario	11/03/16	16/03/16	4
Desarrollo de capa de datos	14/03/16	17/03/16	4
Validación	18/03/16	22/03/16	3
Validación de requisitos	18/03/16	21/03/16	2
Validación de documentación	18/03/16	21/03/16	2
Validación de software	18/03/16	22/03/16	3
Pruebas	23/03/16	28/03/16	4
Realizar pruebas de software	23/03/16	28/03/16	4
Realizar pruebas de performance	23/03/16	25/03/16	3
Realizar pruebas de portabilidad	23/03/16	25/03/16	3
Capacitación	29/03/16	31/03/16	3
Elaboración de manual de usuario	29/03/16	31/03/16	3
Preparación de evento de presentación	29/03/16	31/03/16	3

Ilustración 15. Diagrama de Gantt listado parte 2

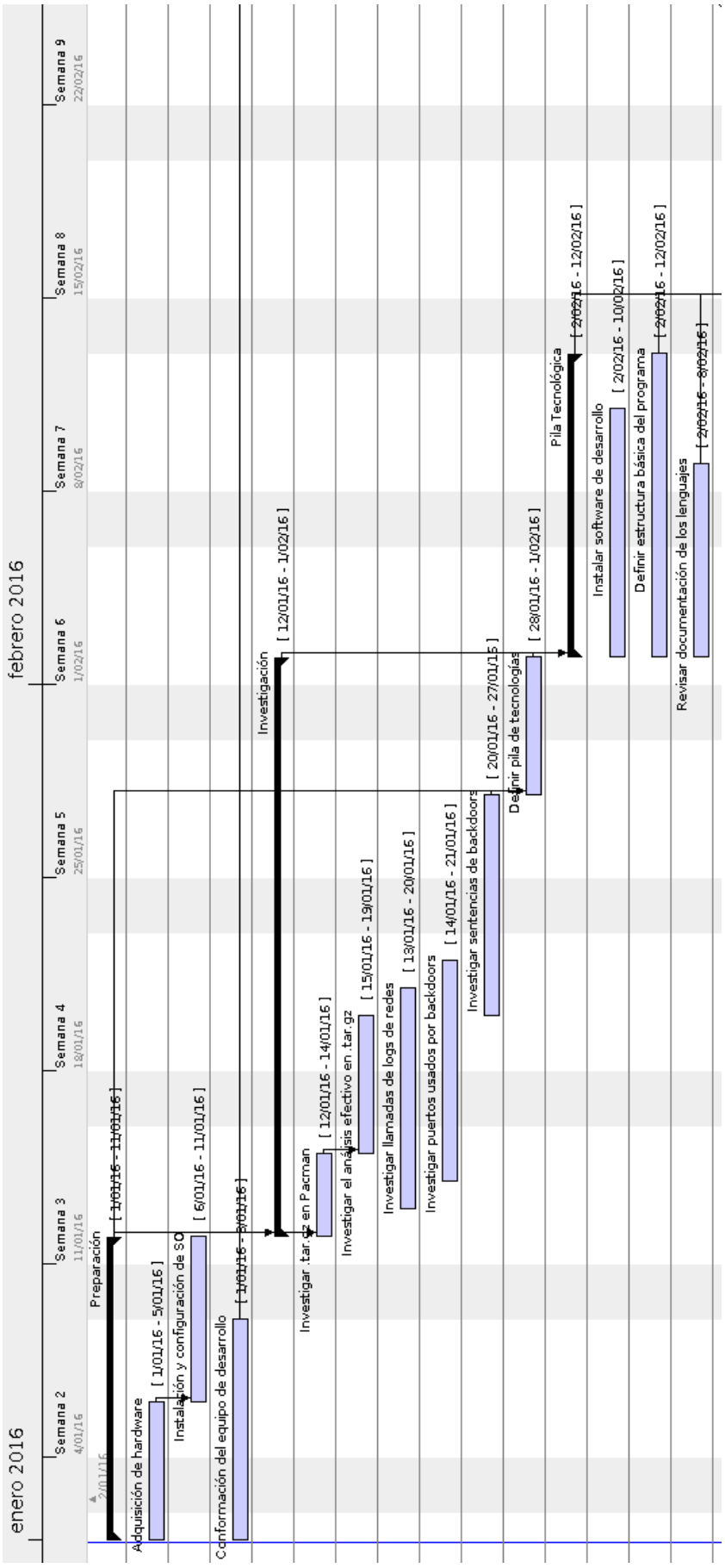

Ilustración 16. Diagrama de Gantt calendario parte 1

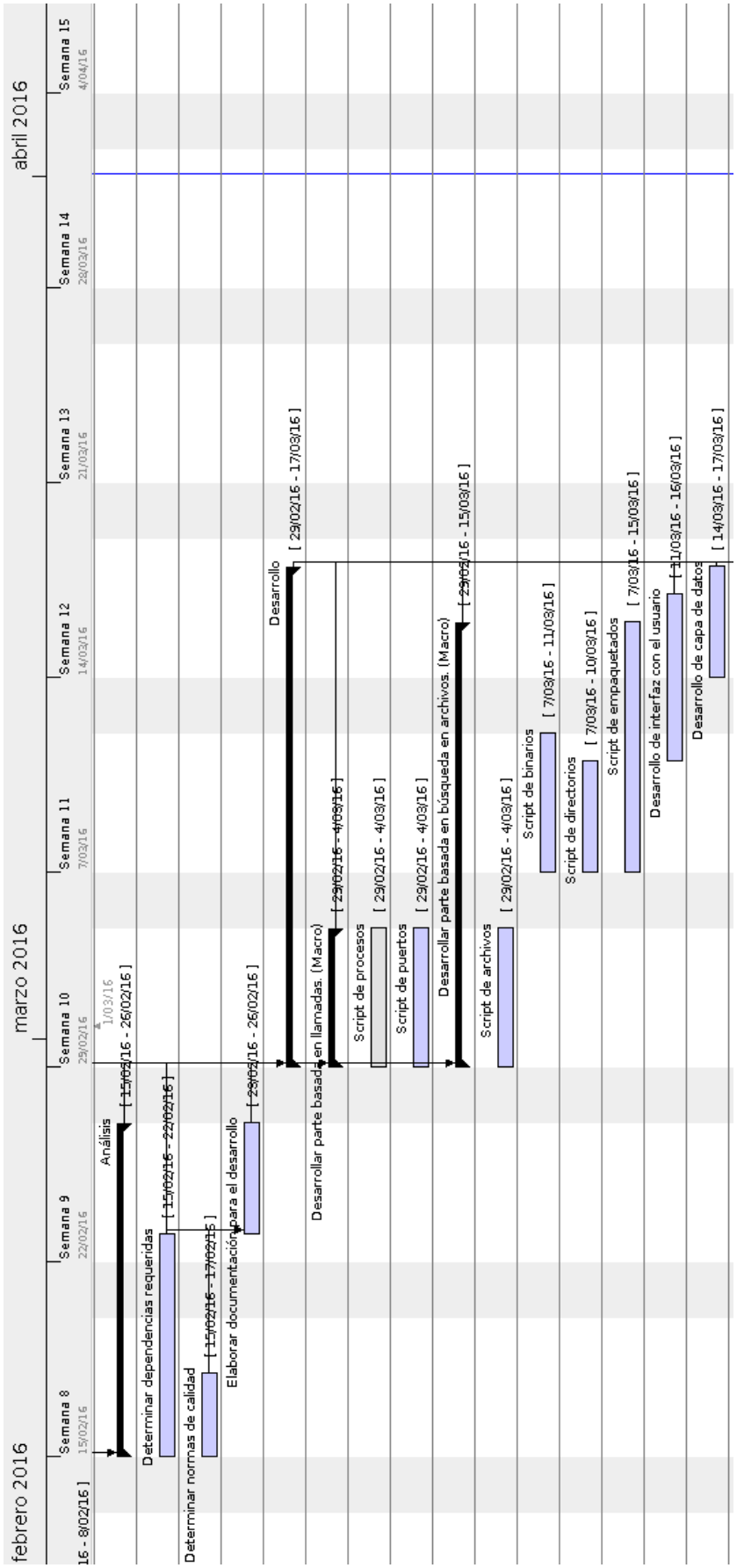

Ilustración 17. Diagrama de Gantt calendario parte 2

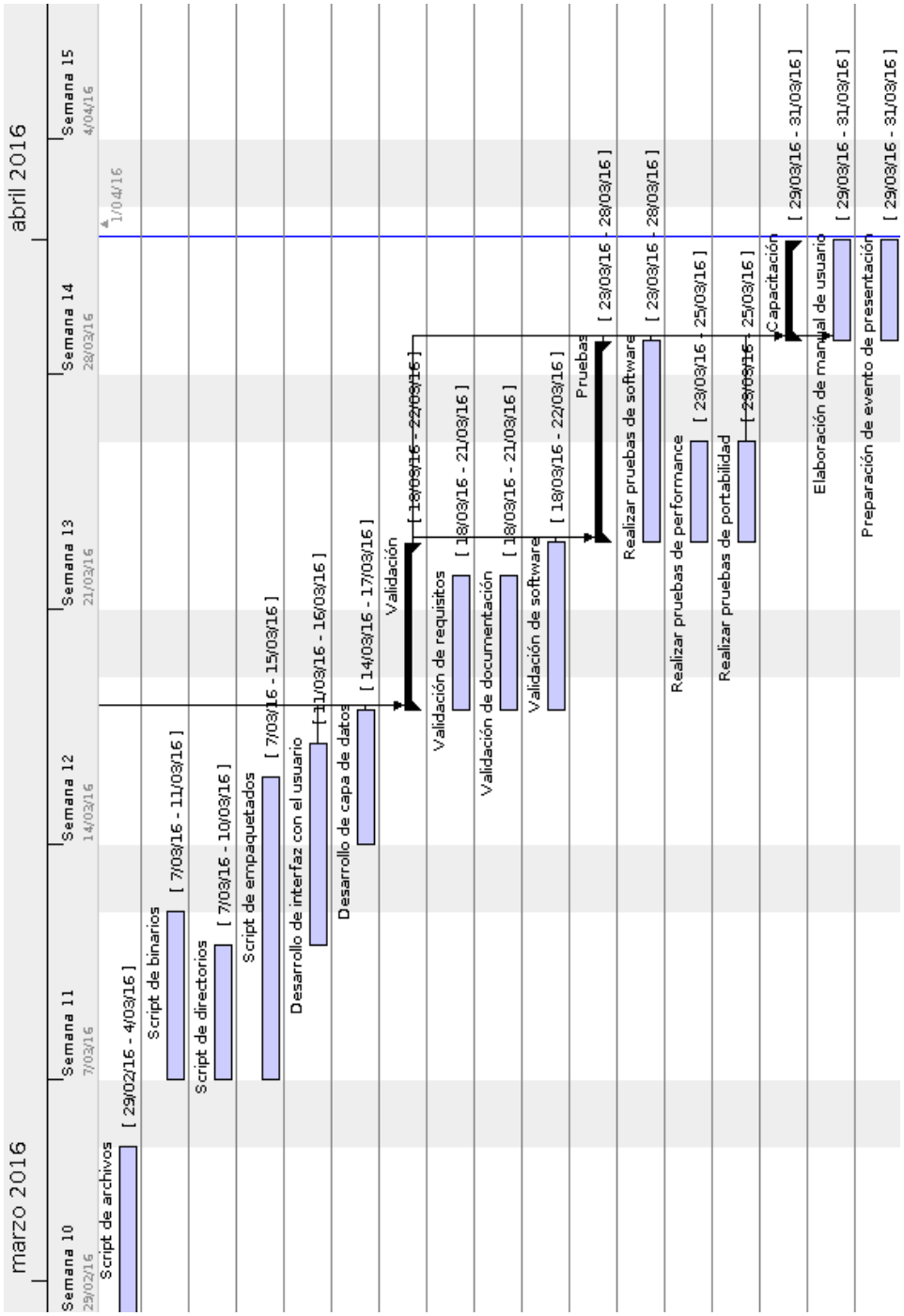

Ilustración 18. Diagrama de Gantt calendario parte 3

Matriz de asignación de responsabilidades

La matriz de asignación de responsabilidades permite describir la participación en las tareas de cada rol que tenga vinculación con el proyecto. Suele ocurrir que la relación entre una persona y un rol sea unívoca pero existen casos donde una persona desempeña dos o más roles. [30]

Referencias de la matriz:

"R" Significa que el rol es responsable del entregable. Usualmente hay una sola persona quien es responsable de crear un entregable, aunque muchas personas pueden proveer entrada.

"A" Significa que el rol aprueba el entregable.

"C" Significa que el rol es consultado sobre el entregable. Esto implica una discusión en ambos sentidos.

"I" Significa que el rol es informado del entregable. Esta es una comunicación en un sentido.

[30] Definición de Matriz de asignación de responsabilidades. http://www.proyectum.lat/2012/02/15/matriz-de-responsabilidades/. Vigente 17/07/2016.

ID	Tarea	Product Owner	Equipo de desarrollo	Cliente (SolAr)	Scrum Master
Sprint 1					
1.1.1	Conformación del equipo de desarrollo	R			I
1.1.2	Adquisición del hardware	I	R		
1.1.3	Instalación y configuración de SO	I	R		I
1.1.4	Investigar .tar.gz en Pacman	I	R		I
1.1.5	Investigar el análisis efectivo en .tar.	I	R		I
Sprint 2					
1.2.1	Investigar llamadas en logs de redes	I	R		I
1.2.2	Investigar puertos usados por backdoors	I	R		I
1.2.3	Investigar sentencias de backdoors	I	R		I
1.2.4	Definir pila de tecnologías	C	R	I	
1.2.5	Instalar software de desarrollo	I	R		I
Sprint 3					
1.3.1	Definir estructura básica del programa	R	C	A	I
1.3.2	Elaborar documentación para el desarrollo	R	I	A	
1.3.3	Determinar normas de calidad para el producto	R	I	C	I
1.3.4	Desarrollo de interfaz con el usuario	C	R	I	I
Sprint 4					
1.4.1	Script de puertos	A-C	R	I	I
1.4.2	Script de procesos	A-C	R	I	I
1.4.3	Determinar dependencias requeridas	A-C	R		
1.4.4	Revisar documentación de los lenguajes	I	R		
1.4.5	Desarrollo capa de datos	A-C	R	I	I
Sprint 5					
1.5.1	Script de binarios	A-C	R	I	I
1.5.2	Script de empaquetados	A-C	R	I	I
1.5.3	Script de directorios	A-C	R	I	I
1.5.4	Script de archivos	A-C	R	I	I
1.5.5	Realizar pruebas de software	A	R	I	I
Sprint 6					
1.6.1	Validación de requisitos	I		R	I
1.6.2	Validación de software	I		R	I
1.6.3	Elaboración de manual de usuario	A	R	C	
1.6.4	Realizar pruebas de performance	A	R	I	
1.6.5	Realizar pruebas de portabilidad	A	R	I	
1.6.6	Validación de documentación	I	I	R	I
1.6.7	Preparación de evento de presentación	R	I	I	I
C/sprint	Retrospectiva del sprint	I	C		R
C/sprint	Revisión del sprint	R	C	A	I

Tabla 8. Matriz de asignación de responsabilidades

Desarrollo del prototipo

A continuación se explicará la gestión y el uso de herramientas administrativas utilizadas conforme se desarrollaba el presente proyecto y el prototipo requerido para llevar a cabo las pruebas funcionales y de rendimiento de BackPearl.

Taiga

Taiga es una plataforma open source de gestión de proyectos ágiles.[31] Permite la creación de equipos para el desarrollo de un producto, asignación de tareas, seguimiento, elaboración de gráficos y estadísticas, brinda un medio de comunicación para el equipo y es gratuito para proyectos públicos. Brinda soporte específico para Scrum y Kanban.

El uso de Taiga permite que el equipo cuente con un soporte de fácil acceso en el que quedan asentadas las tareas que cada miembro debe realizar a cabo en cada uno de los sprints. Estas tareas son elegidas por cada miembro en las reuniones de planificación tal como lo plantea la metodología elegida para el desarrollo, Scrum.

Dentro de Taiga se pueden gestionar los sprints creando historias de usuario, las cuales se encuentran enunciadas en el Diagrama de Gantt que se ha realizado de forma estimativa para determinar el tiempo que requiere la implementación del proyecto. Tanto el mencionado diagrama como las herramientas proporcionadas por Taiga deben funcionar de forma articulada y tienen que estar sincronizadas. La información que proporcionan no es redundante y permiten tomar distintas decisiones.

Sprints

Una vez creado el proyecto en Taiga y cargadas las historias de usuario, se define que el número de sprints será 6, con una duración fija de 15 días cada uno. A su vez, la suma de los puntos de complejidad de las historias de usuario es de 182. Este número cobra importancia a la hora de elaborar estadísticas de seguimiento. Una vez que tenemos definida esta información, se procede a asignar las historias de usuario a cada sprint en cada reunión de planificación. Es decir, que las historias de usuario que se resolverán en el sprint se deciden justo antes de empezarlo.

Para ejemplificar, el primer sprint se inicia el 01/Enero/2016 y termina el 15/Enero/2016. En este sprint tenemos un total de 28 story points, puntos de complejidad o de esfuerzo, que fueron asignados por el equipo a las historias de usuario. Si la suma total de los

[31] Taiga página oficial. www.taiga.io. Vigente 01/07/2016.

puntos de complejidad es 182, repartidos entre 6 sprints obtenemos que en cada sprint deberíamos resolver poco más de 30 puntos.

La siguiente captura de pantalla proveniente de Taiga fue realizada tres días antes de terminar el sprint. En ella podemos ver información básica tales como fechas, puntos totales del sprint (28), puntos ya terminados (15) y todas las historias de usuario que formaron parte. En gris se encuentran las historias que ya fueron terminadas incluyendo su testeo. Escritas con negro se encuentran las historias que aún no fueron terminadas, tales como Investigar el análisis efectivo de .tar.gz y Adquisición hardware. Debido a que se finalizaron las tareas en tiempo y forma, si tomáramos una captura de pantalla al cierre del sprint veríamos que todas las historias se encuentran en gris. Por último, es importante aclarar que las historias de usuario de un sprint pueden finalizarse en cualquier orden; no es necesario seguir el criterio que figura en la lista. Esto es válido para todas las historias de usuario que no tengan dependencias. Si existiesen, Taiga y el Diagrama de Gantt ofrecen la posibilidad de indicarlo, impidiendo el comienzo de una historia hasta que no se finalicen todas aquellas otras de las cuales depende.

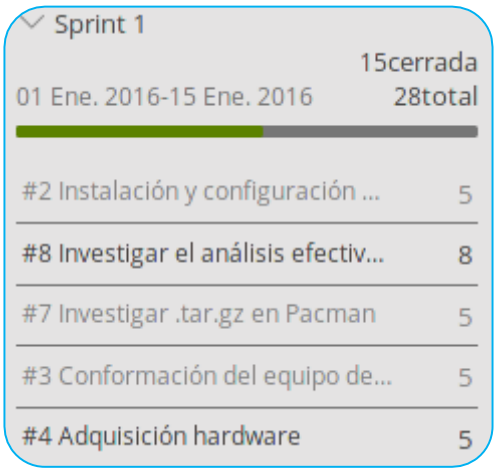

Ilustración 19. Sprint 1

Continuando con el análisis de los sprints a lo largo del desarrollo del prototipo de BackPearl, observamos una captura de pantalla realizada al segundo sprint, correspondiente al período comprendido entre el 15/Enero/2016 y el 31/Enero/2016. El total de puntos del sprint es de 28. Al momento de realizar la captura había 23 puntos cerrados, es decir que se habían finalizado historias de usuario que representan 23 puntos. La diferencia, 5, corresponde a la historia denominada Investigar puertos usados por backdoors. Como en el caso anterior, la captura se realizó un par de días antes a la fecha de cierre del sprint. En el desarrollo del prototipo se cumplimentó con los contenidos propuestos para este período, por lo que el

15/Enero/2016 todas las historias de usuario pertenecientes a los dos primeros sprints estaban terminadas.

Ilustración 20. Sprint 2

A continuación observamos una imagen del resumen del tercer sprint. En este caso tenemos una menor cantidad de historias de usuario, en total 4, pero dichas tareas son consideradas de mayor complejidad por los miembros del equipo. Este sprint tiene un total de 29 puntos y pertenece al período comprendido entre el 31/Enero/2016 y el 14/Febrero/2016, siguiendo la recomendación de Scrum de mantener una cantidad fija de días en todos los sprints. La captura se realizó el primer día, por lo tanto aún no hay historias de usuario finalizadas.

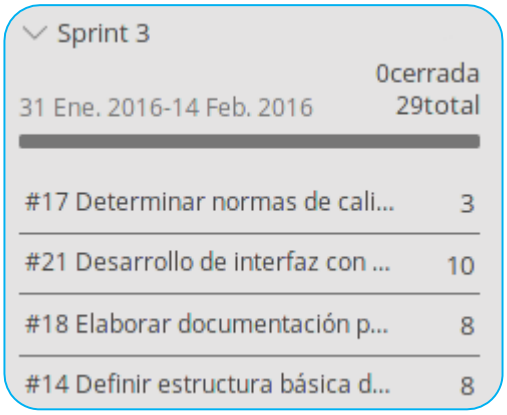

Ilustración 21. Sprint 3

En la siguiente captura podemos observar la información básica del cuarto sprint. Como en el caso anterior, la misma se realizó el primer día del mismo. Esta vez tenemos 34 puntos y 5 historias de usuario.

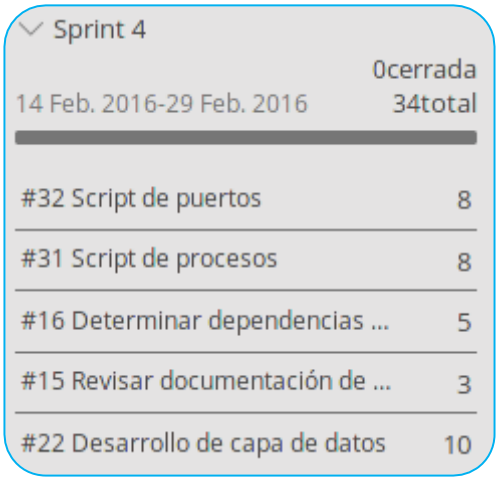

Ilustración 22. Sprint 4

En el quinto sprint se desarrollarán 32 puntos de esfuerzo durante los días comprendidos entre el 29/Febrero/2016 y el 14/Marzo/2016. Este sprint se caracteriza por tener casi la totalidad de sus historias de usuario relacionadas con la programación. Al igual que en los últimos anteriores, la captura se realizó al comienzo del mismo y por lo tanto no hay historias de usuario cerradas o finalizadas.

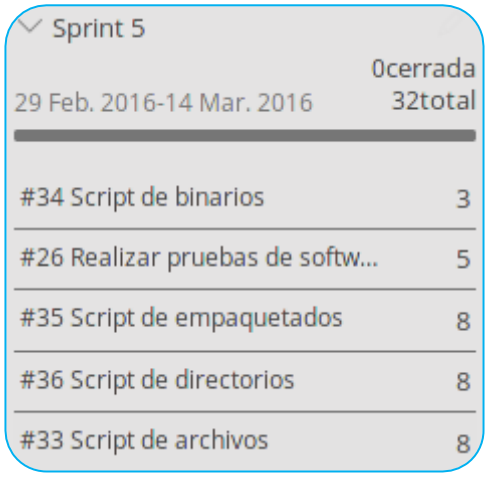

Ilustración 23. Sprint 5

Por último, el sexto sprint comienza el 14/Marzo/2016 y termina el 32/Marzo/2016. Está caracterizado por las tareas relacionadas con la culminación de un proyecto. Se continúa con las pruebas de software, tales como pruebas de performance y portabilidad. Además, se crea el manual de usuario y se valida que la documentación elaborada sea apropiada, a la vez que se comprueba que el software responda a las necesidades planteadas en un principio. También se prepara el evento de presentación del proyecto. Los 31 puntos de esfuerzo se

reparten entre 7 tareas, un número superior de tareas respecto a los anteriores sprints pero cuyas complejidades son menores, por lo tanto es posible llevarlas a cabo en un solo sprint.

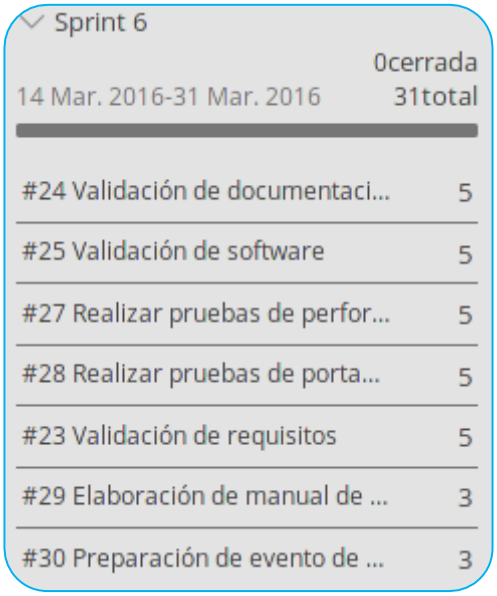

Ilustración 24. Sprint 6

Distribución de puntos de esfuerzo

En el siguiente gráfico se aprecia la distribución de los puntos de esfuerzo de cada sprint, también llamados story points. Utilizar estos puntos abstractos en lugar de unidades temporales otorga ciertas ventajas, tales como dedicar un menor tiempo de estimación, lograr un mayor concenso entre los miembros del equipo, menor estrés generado en las personas al no sentirse tan presionadas para estimar una ya que se reduce el miedo a equivocarse. Estas teorías están respaldadas por encuestas y estudios realizados a software factories.[32]

En el proyecto vemos cómo ha sido la distribución de los puntos de esfuerzo a lo largo de los sprints. En primer lugar se aprecia que la hay una clara similitud con la campana de Gauss. Una de las ventajas que proporciona este sistema es que en los primeros sprints se exige menor esfuerzo a los miembros del equipo. Es muy recomendable adoptar esta estrategia en equipos recién conformados o cuando se trabaja con una pila tecnológica nueva, ya que existe mayor probabilidad de que surjan contratiempos. Asignar muchos puntos de esfuerzo al primer sprint y considerando que es la etapa donde hay más imprevistos conlleva a que no se puedan completar a tiempo todas las historias de usuario planificadas para dicho sprint. Prosiguiendo con este proyecto, se aprecia que el cuarto sprint tiene asignado el mayor

[32] Estudios sobre el uso de story points. https://www.scruminc.com/story-points-why-are-they-better-than/. Vigente 04/07/2016.

número de puntos de complejidad porque se considera que los miembros del equipo ya se encuentran en un buen ritmo de trabajo. Luego la cantidad de puntos disminuye en el último sprint para que, de ser necesario, se puedan culminar tareas que quedasen pendientes de sprints anteriores porque ha ocurrido algún retraso.

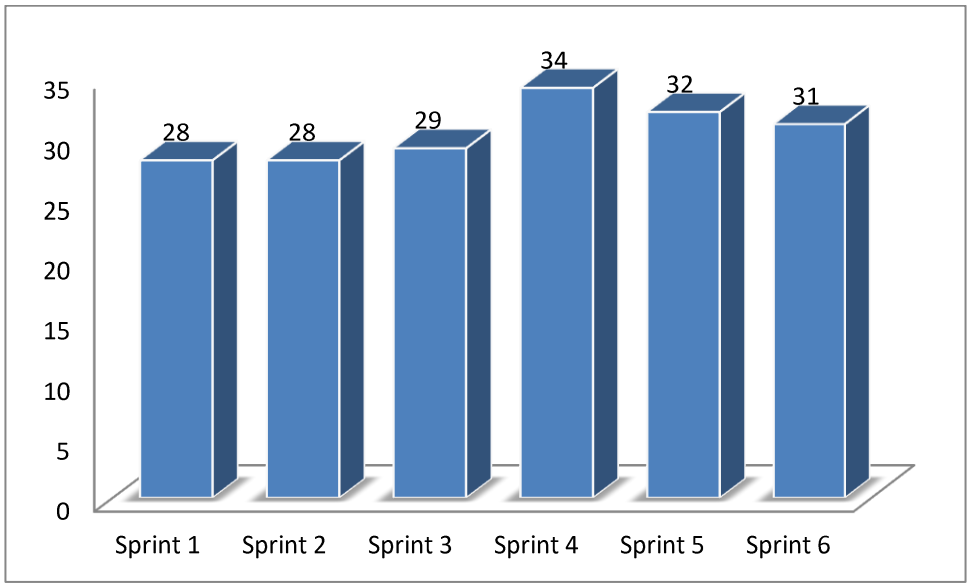

Ilustración 25. Distribución de puntos de esfuerzo

Burndown Chart

Burndown chart o gráfico de trabajo pendiente es un diagrama que muestra la velocidad a la cual se están completando los objetivos, requisitos o tareas que son reflejados a través de los puntos de esfuerzo. Otorga una rápida respuesta, excluyendo algunas variables, que permite determinar si la velocidad a la cual se está trabajando es suficiente para alcanzar las metas planificadas en el tiempo deseado. Este tiempo es representado a través de la cantidad de sprints que se ha calculado al inicio del proyecto y cuya duración es fija y ya está determinada. En BackPearl se consideran 6 sprints con una duración fija de 15 días. La utilización del burndown chart permite conocer la desviación temporal para realizar las correcciones necesarias que permitan aumentar la velocidad del equipo.

A continuación se muestra el burndown chart al momento de terminar el segundo sprint, es decir a los 30 días de iniciado el proyecto. En la barra de información que se encuentra en la parte superior podemos ver varios elementos. El primero de ellos es una barra de colores que indica, en verde, los puntos que se cerraron, es decir, que corresponden a historias de

usuario finalizadas. En blanco se representa la cantidad de puntos que faltan terminar. Estos puntos corresponden a historias de usuario que se desarrollan en otros sprints. Esta misma información se muestra en forma porcentual. El 31% indica que durante dos sprints se realizó aproximadamente un tercio del trabajo total. Se ha indicado que 182 es la cantidad de puntos que contendrá el proyecto y que ya se han asignado 182 a las historias de usuario que se han cargado. Estos valores pueden modificarse posteriormente. A continuación encontramos el valor 56 que corresponde a los puntos cerrados. Este valor es la suma de los puntos cerrados en los sprints hasta la fecha. Al momento de hacer la captura de pantalla se habían terminado los dos primeros sprints, cada uno con 28 puntos de esfuerzo, siendo 56 la suma de ambos. Luego tenemos la división entre este valor y la cantidad de sprints que se realizaron, coincidiendo con la mitad, 28. Esta es la velocidad por sprint hasta el momento.

Por último, el gráfico indica la relación entre los puntos en el eje de las ordenadas y los sprints en el eje de las abscisas. La línea verde y los pares coordenados que la determinan simbolizan los puntos reales pendientes para el próximo sprint. El primer punto verde se encuentra en (0, 182), lo cual significa que al momento de empezar el proyecto se declararon 182 puntos. El siguiente punto es (1, 154), que indica que al finalizar el primer sprint quedan pendientes 154 puntos de esfuerzo para ser distribuidos entre los siguientes sprints. La línea verde continúa de forma horizontal, sin pendiente, porque no se han asignado ni cerrado, hasta el momento, historias de usuario a los siguientes sprints. Formando una línea y área gris se encuentra la proyección sobre el número óptimo de puntos pendientes para los próximos sprints, considerando la duración total del proyecto. En otras palabras, determina la velocidad apropiada para concluir a tiempo (medido en cantidad de sprints) el proyecto. Por último, la línea roja representa los puntos de complejidad que fueron añadidos por los miembros del equipo. Por lo general estos puntos indicados en rojo reflejan nuevos requerimientos o correcciones en las complejidades de las tareas. La línea carece de pendiente porque no se agregaron nuevos requerimientos ni se modificaron las estimaciones a medida que se desarrollaron los sprints.

Ilustración 26. Burndown chart sprints 1-2

A continuación podemos apreciar una captura de pantalla extraída de Taiga que corresponde al burndown chart en el momento en que finalizaba el tercer sprint. Se observa un incremento en el porcentaje de avance del 16%. Terminada la tercera etapa el proyecto se encuentra casi a la mitad de su desarrollo. La velocidad se mantiene en 28 puntos/sprint y se finalizaron 85 puntos de esfuerzo. No se observan cambios en la proyección determinada por la línea gris comparado con el gráfico anterior.

Ilustración 27. Burndown chart sprint 3

La siguiente imagen corresponde al burndown chart al cierre del cuarto sprint. El avance porcentual comparado con la etapa anterior es del 18%. Aumenta la velocidad del equipo a 30 puntos/sprint, lo que representa un incremento en 2 unidades respecto al anterior. Dicha diferencia no modifica la proyección para la finalización del proyecto. Se encuentran cerrados 119 de 182 puntos de esfuerzo.

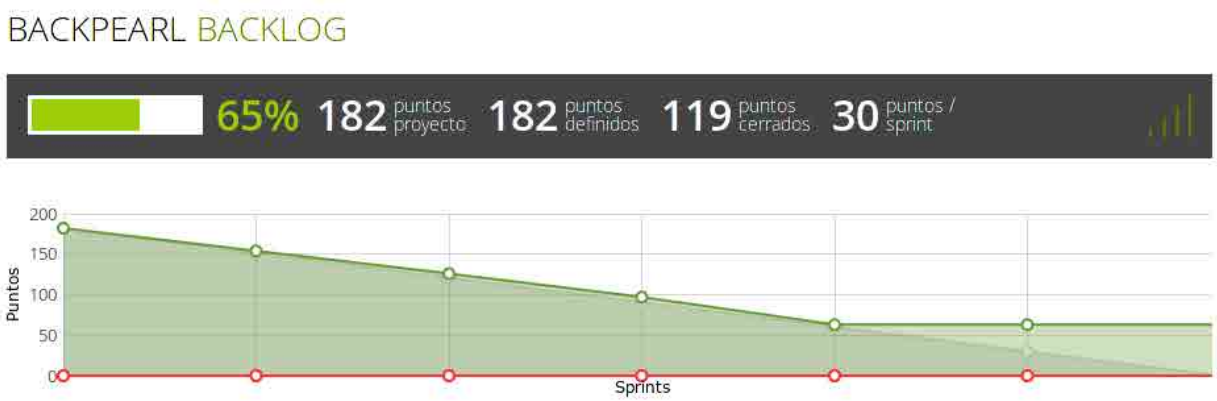

Ilustración 28. Burndown chart sprint 4

Con respecto al gráfico realizado al finalizar el quinto sprint, se puede notar un avance del 18%, lo cual habla de un progreso estable. Se han desarrollado 32 puntos de complejidad pero en promedio la velocidad del equipo continúa siendo 30 puntos/sprint.

Ilustración 29. Burndown chart sprint 5

El siguiente caso corresponde al último burndown chart, por lo tanto el porcentaje de avance es del 100%, esto implica que el proyecto ha finalizado porque se concluyeron todos los puntos establecidos en los inicios de los sprints y que no ha habido incrementos en los requerimientos por parte del equipo (línea roja) durante la última etapa, lo cual podría haber afectado la planificación y la fecha de entrega. En promedio, la velocidad del equipo culmina con 30 puntos/sprint. Este dato se transforma en información útil si el equipo continúa desarrollando proyectos similares. Se puede llegar a la conclusión de que hubo gran estabilidad, o dicho de otra forma, poca variabilidad a lo largo de todos los sprints. El avance porcentual promedio de cada etapa es de 17.5%. Como medida estadística se tiene que en el cuarto sprint hay un avance de 18%, en el quinto también, 18%, y en el último sprint es de 17%.

Ilustración 30. Burndown chart sprint 6

Dashboard

Se conoce por dashboard o tablero al soporte, físico o virtual, que permite acomodar dinámicamente las historias de usuario y sus tareas derivadas según las categorías que indican su estado. En Taiga, así como en la mayoría de los tableros online, existen las siguientes etapas o categorías en las que se pueden enmarcar las historias de usuario: *Nueva*, *En curso*, *Lista para testear*, *Cerrada* y *Necesita información*.

A lo largo de cada sprint, los miembros del equipo deben asignarse tareas (en esta primer instancia se etiquetarán como *Nuevas*). A medida que vayan trabajando con ellas se colocarán en la categoría *En curso*. Al terminar una tarea, ésta adquiere el estado de *Lista para testear* y, luego de ser aprobada por el Product Owner, se colocará bajo la categoría denominada *Cerrada*. Si la persona asignada para desarrollar una tarea requiere explicaciones o más detalles sobre lo que debe hacer entonces etiquetará a la misma como *Necesita información*. A dicha información se la proporcionará el Product Owner.

A continuación se observa una captura de pantalla dividida en dos secciones proveniente del dashboard de Taiga realizada a mitad del primer sprint. En ella podemos observar el porcentaje de avance del sprint, 54%; la cantidad de puntos totales del sprint, 28; la cantidad de puntos completados hasta el momento, 15; las tareas abiertas (se excluyen tareas listas para testear), 1; y las tareas cerradas o finalizadas, 3. Respecto al ítem denominado *Dosis de idocaína*, sirve para indicar tareas que requieren un esfuerzo adicional por parte de los miembros del equipo o que la velocidad de los mismos aumentará notablemente para llevar a cabo dicha tarea. En el desarrollo del proyecto no fue necesario utilizar este ítem ni aumentar notablemente la velocidad del equipo. En resumen, encontramos una tarea en curso, una lista para testear y tres cerradas. El tablero ha sido recortado para ser incorporado en el presente documento.

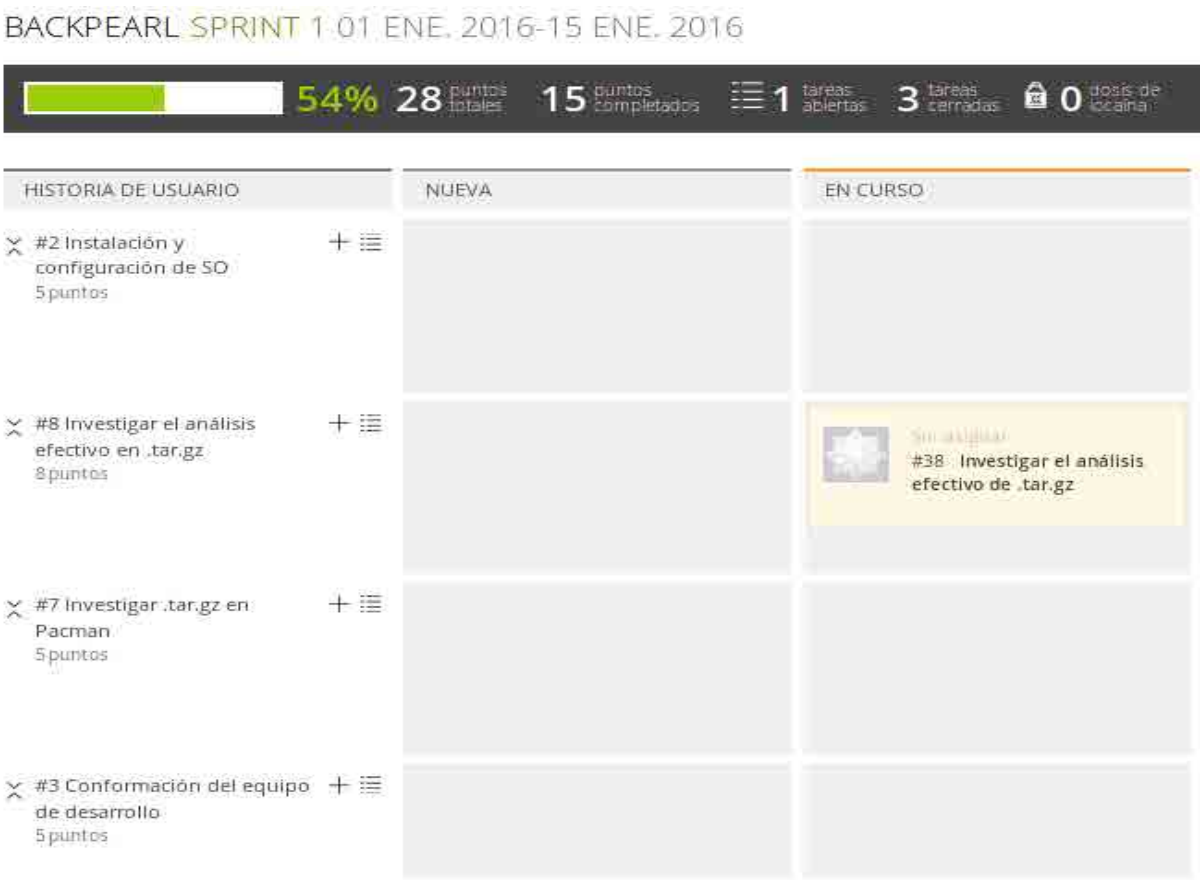

Ilustración 31. Dashboard sprint 1 parte 1

Ilustración 32. Dashboard sprint 1 parte 2

En la siguiente captura se observa el dashboard divido en dos imágenes que corresponde al segundo sprint. La misma se realizó cuando el avance era del 82%, se encontraba en curso una sola tarea, había dos listas para testear y cuatro tareas cerradas.

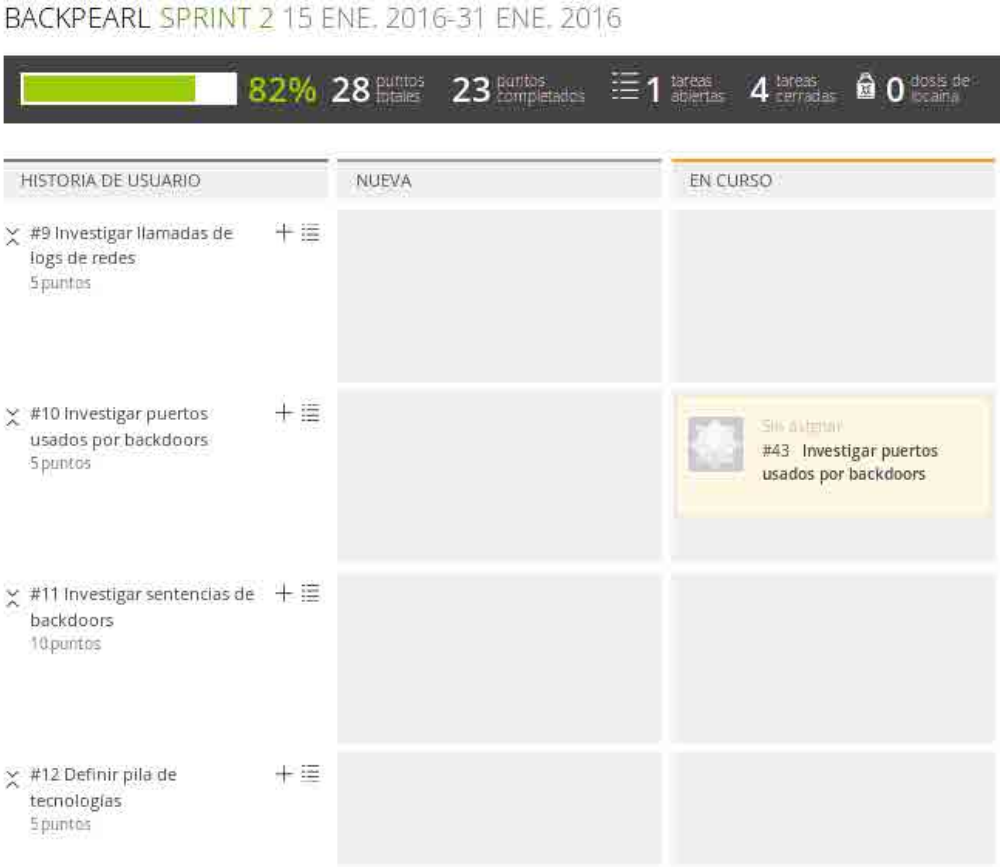

Ilustración 33. Dashboard sprint 2 parte 1

Ilustración 34. Dashboard sprint 2 parte 2

Kanban

Kanban es un método para la gestión de procesos que puede resumirse en un cuadro de mando o tablero similar al dashboard. La diferencia principal radica en que ahora se encuentran juntas todas las historias de usuario y las tareas que las componen. En el caso del dashboard, se pueden visualizar solamente las historias del sprint en curso. Con Kanban se consigue una visión global sobre el avance de las tareas a lo largo del tiempo y no hay interferencias con el uso de sprints si se realizan los cambios con cuidado. En Kanban encontramos las siguientes etiquetas: *Nueva, Preparada, En curso, Lista para testear y Hecha.*

A continuación observamos una captura realizada a través de Taiga, durante el primer día de trabajo ya con un equipo de desarrollo constituido. Podemos ver dos tareas en curso, una preparada, es decir, lista para ser comenzada y puesta en curso, y por último, el listado con todas las tareas nuevas o que aún no fueron empezadas.

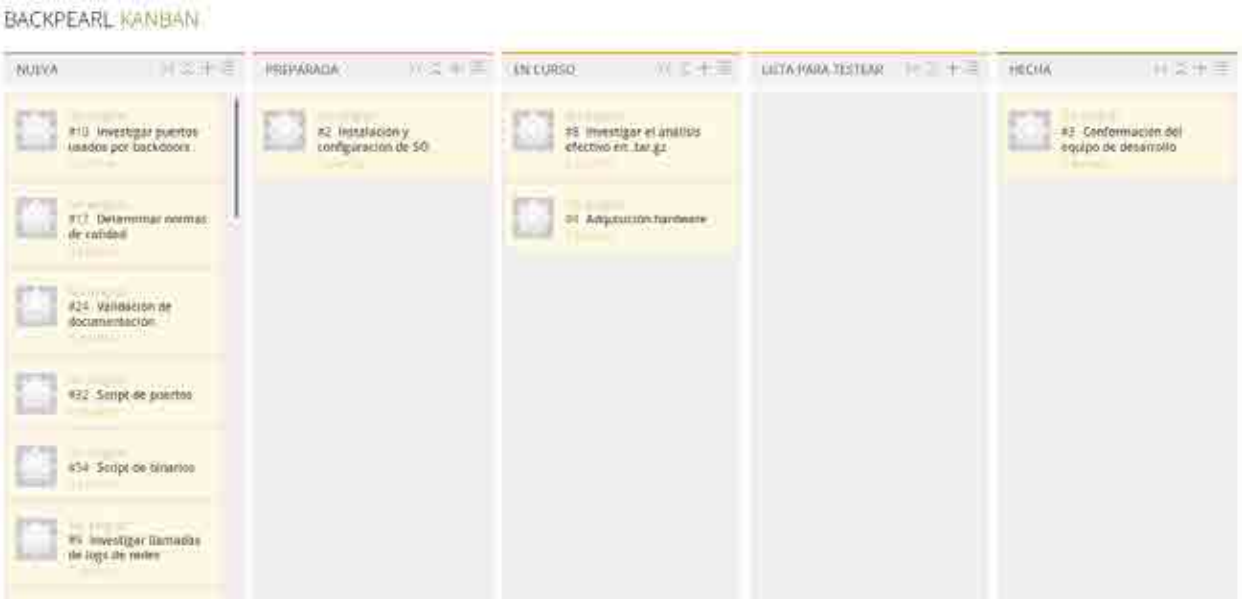

Ilustración 35. Kanban semana 1

En la siguiente captura se puede visualizar el estado del tablero realizada durante la tercera semana. Hay tareas distribuidas a lo largo de todas las etiquetas y refleja qué trabajo se está ejecutando en dicho momento. Si se agrega el nombre del responsable de cada tarea a las tarjetas entonces se puede controlar la actividad de cada miembro del equipo y evitar sobrecargas de trabajo.

Ilustración 36. Kanban semana 3

Interfaces de usuario

El usuario cuenta con interfaces de entrada y salida que son gráficas a través de la librería Zenity pero también cuenta con la opción de utilizar el programa sin interfaz gráfica a través de la terminal. Este último caso es aplicable cuando la distribución no tenga instalado un entorno de escritorio o gestor de ventanas. A continuación se mostrarán algunas interfaces que son representativas de BackPearl.

Inicio y proceso de carga

Como se observa, el proceso de inicio y carga muestra varias pantallas que le dan la bienvenida al usuario y le informan si su sistema cumple con los requisitos para ejecutar BackPearl.

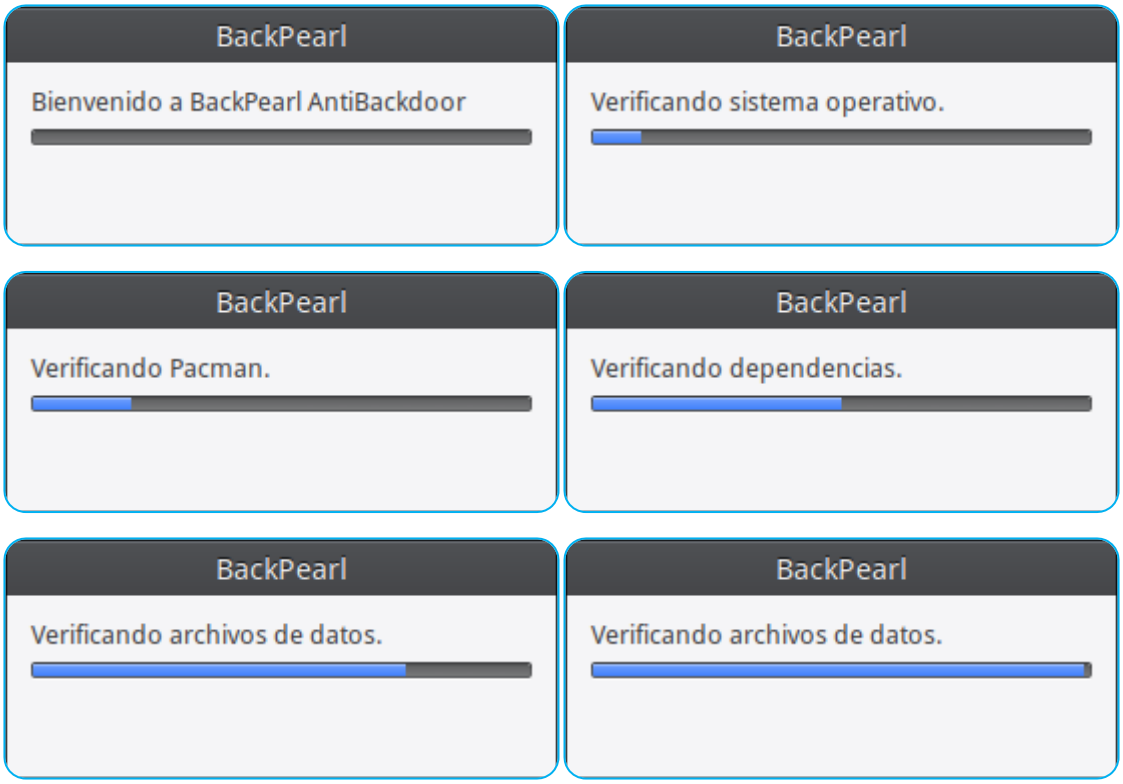

Ilustración 37. Inicio y proceso de carga

Menú principal

El menú principal proporciona acceso a todos los scripts y a la sección de Ayuda y Configuraciones. Permite seleccionar una opción seleccionándola con el mouse, con el teclado utilizando las flechas de desplazamiento o escribiendo el nombre de la opción a la que se quiere acceder. Posee una búsqueda dinámica, esto quiere decir que si ingresamos, por ejemplo, las letras "*Ayud*" se seleccionará la opción *Ayuda y Configuraciones*.

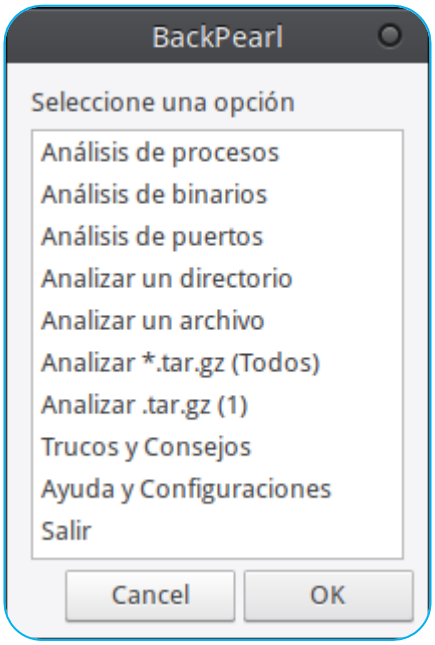

Ilustración 38. Menú principal

Análisis de procesos

En el *Análisis de procesos* se muestran las principales ventanas que surgen al trabajar en modo gráfico, sin salida de información a través de la terminal. Se muestra el listado de los procesos analizados con su correspondiente PID (número de identificación del proceso) y su nombre. Luego se informa si se ha detectado o no un backdoor. En el caso afirmativo, se mostrará la ubicación del mismo para que el usuario pueda tomar medidas al respecto.

BackPearl utiliza, en algunos análisis, notificaciones que el entorno de escritorio y/o sistema operativo deben tener activas, tales como la que se muestra a continuación y que informa al usuario que se están analizando procesos. El uso de notificaciones permite informar que el programa está funcionando correctamente.

Ilustración 39. Notificación

El listado de procesos muestra todos los procesos que están ejecutándose en el sistema al momento de realizar la consulta y todos ellos han sido analizados con el objetivo de encontrar un backdoor.

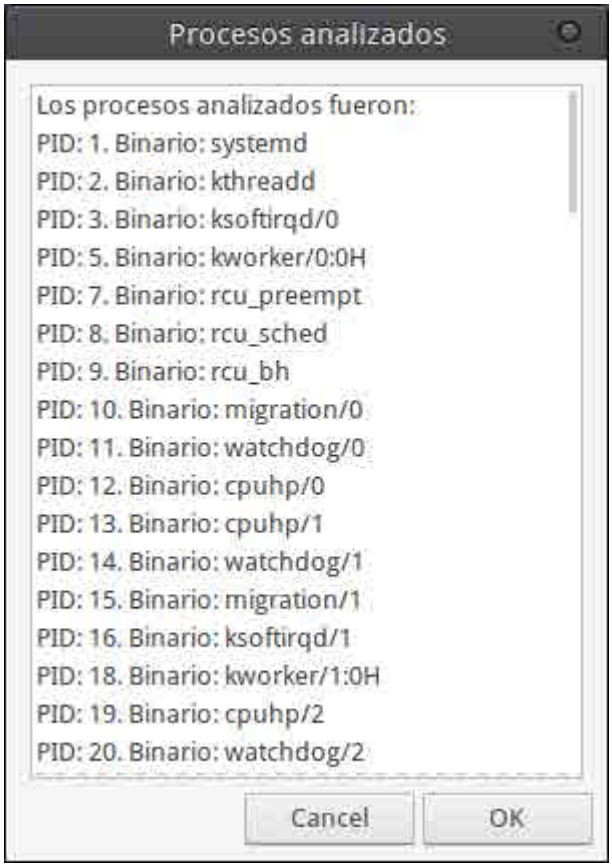

Ilustración 40. Muestra de resultado de procesos

Se muestra tanto una alerta positiva en términos de antivirus (se ha detectado un posible backdoor) como también un análisis en el cual no se ha podido detectar backdoors; esto no quiere decir que no existan, sino que los algoritmos no han podido detectar ninguno.

Ilustración 41. Alerta de detección

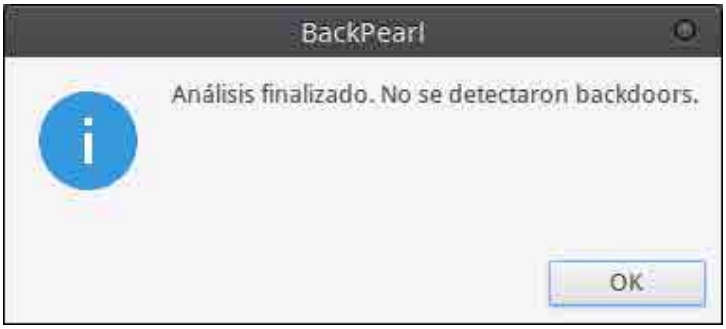

Ilustración 42. Alerta de finalización

A continuación se aprecia la ejecución de BackPearl en modo gráfico con terminal, es decir, se obtiene información a través de ventanas y de líneas en la terminal. Este segundo caso es útil cuando el sistema carece de un entorno de escritorio o gestor de ventanas. Además, la salida por terminal, en algunos casos, otorga mayor información sobre el procesamiento que lleva a cabo BackPearl.

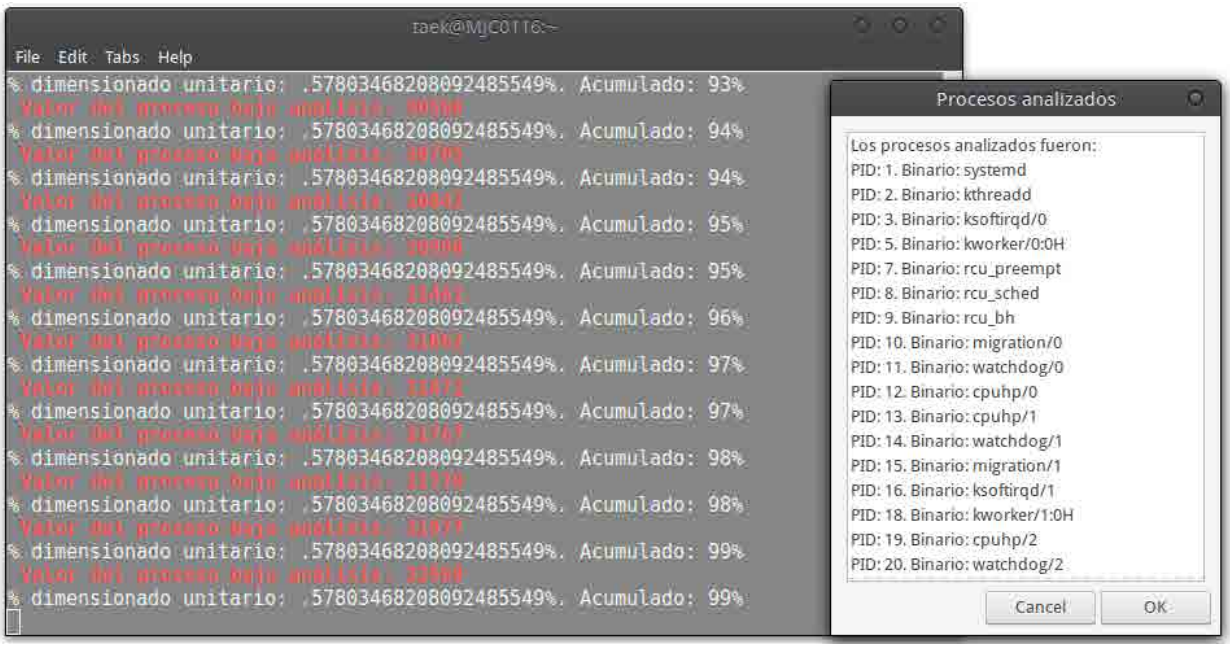

Ilustración 43. Modo gráfico con terminal

Análisis de archivo

El script basado en el análisis de un archivo cuenta con las siguientes pantallas básicas.

En la primera imagen se detecta un archivo sospechado de contener un backdoor. En la siguiente pantalla se proporciona más información sobre el archivo y los comandos que dan origen a las sospechas.

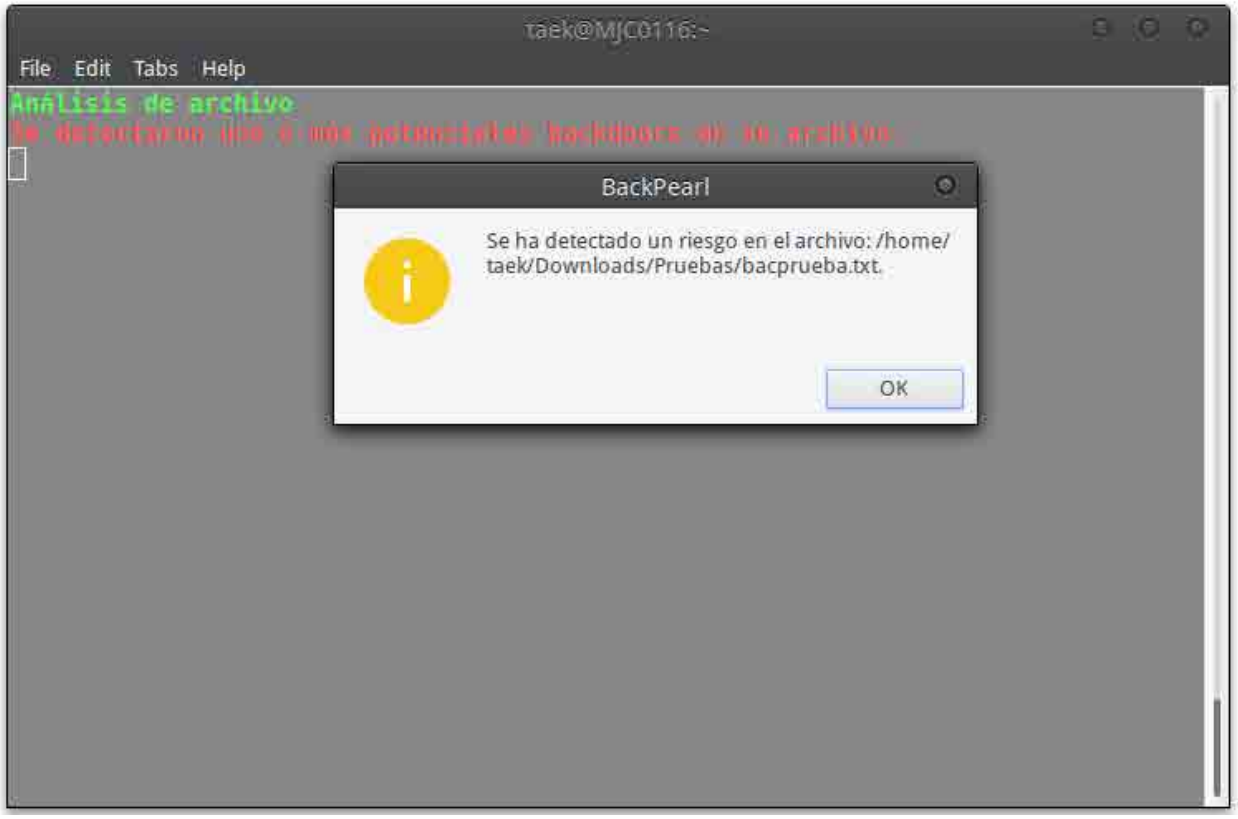

Ilustración 44. Resultados análisis archivo

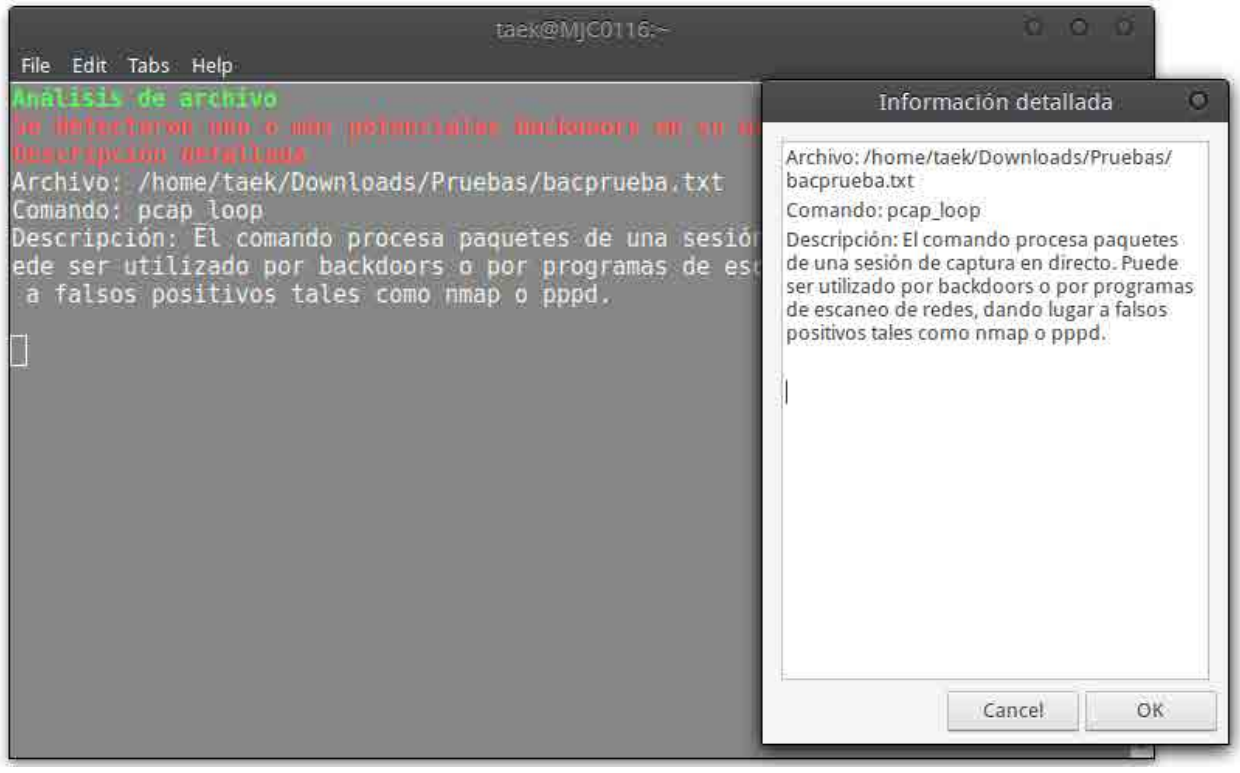

Ilustración 45. Información detallada

En la tercera imagen, a continuación, se lleva a cabo un análisis de un archivo que no contiene backdoors.

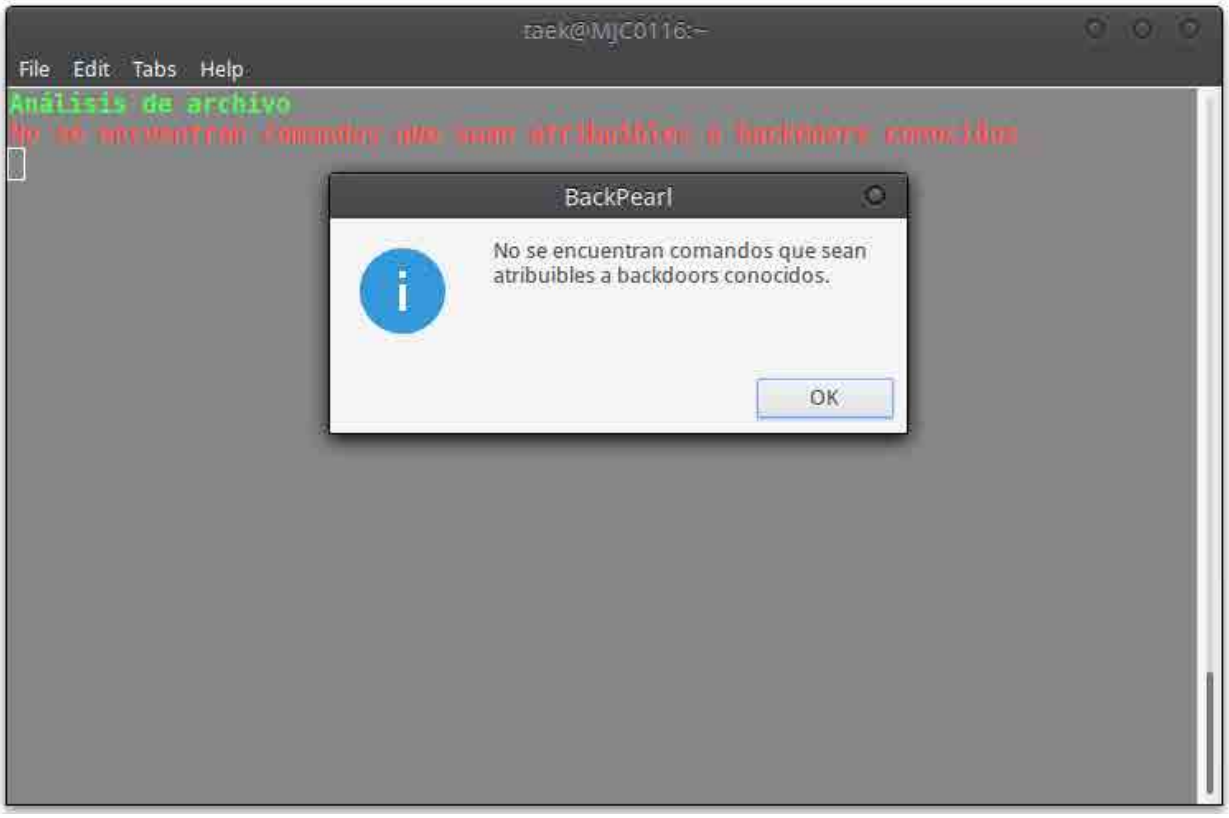

Ilustración 46. Resultados análisis archivo

Análisis de directorio

Lo que distingue el análisis de un directorio de la mayoría de los scripts es que al usuario se le permite seleccionar de forma gráfica la ubicación del directorio que quiere analizar. En la ilustración nos encontramos navegando entre las carpetas del usuario y el programa solamente nos permitirá seleccionar una de ellas, impidiendo que se ingrese un archivo.

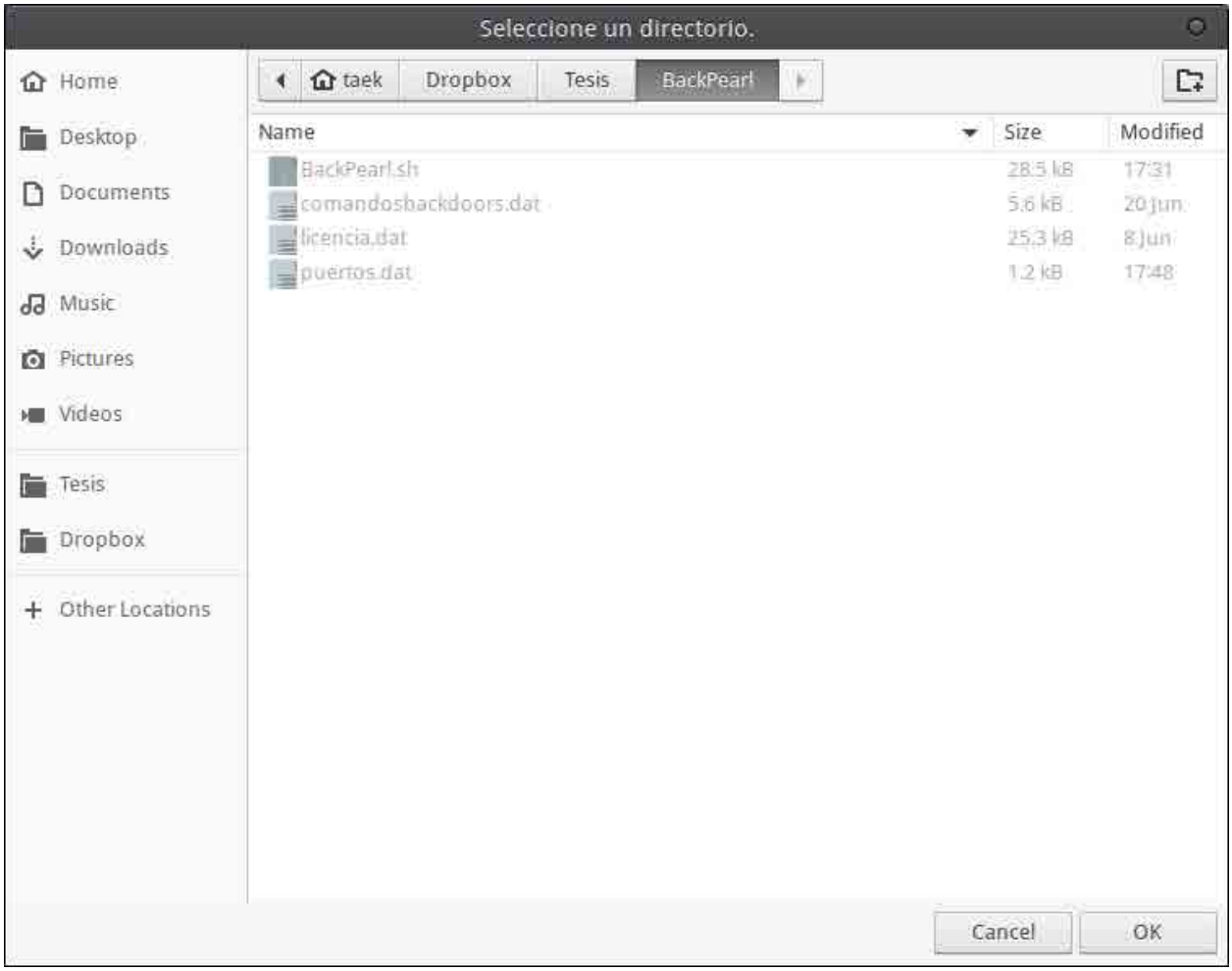

Ilustración 47. Selección de directorio

En la siguiente imagen se ha realizado un análisis del directorio /boot/, el cual contiene los archivos necesarios para realizar el proceso de arranque de un sistema operativo.

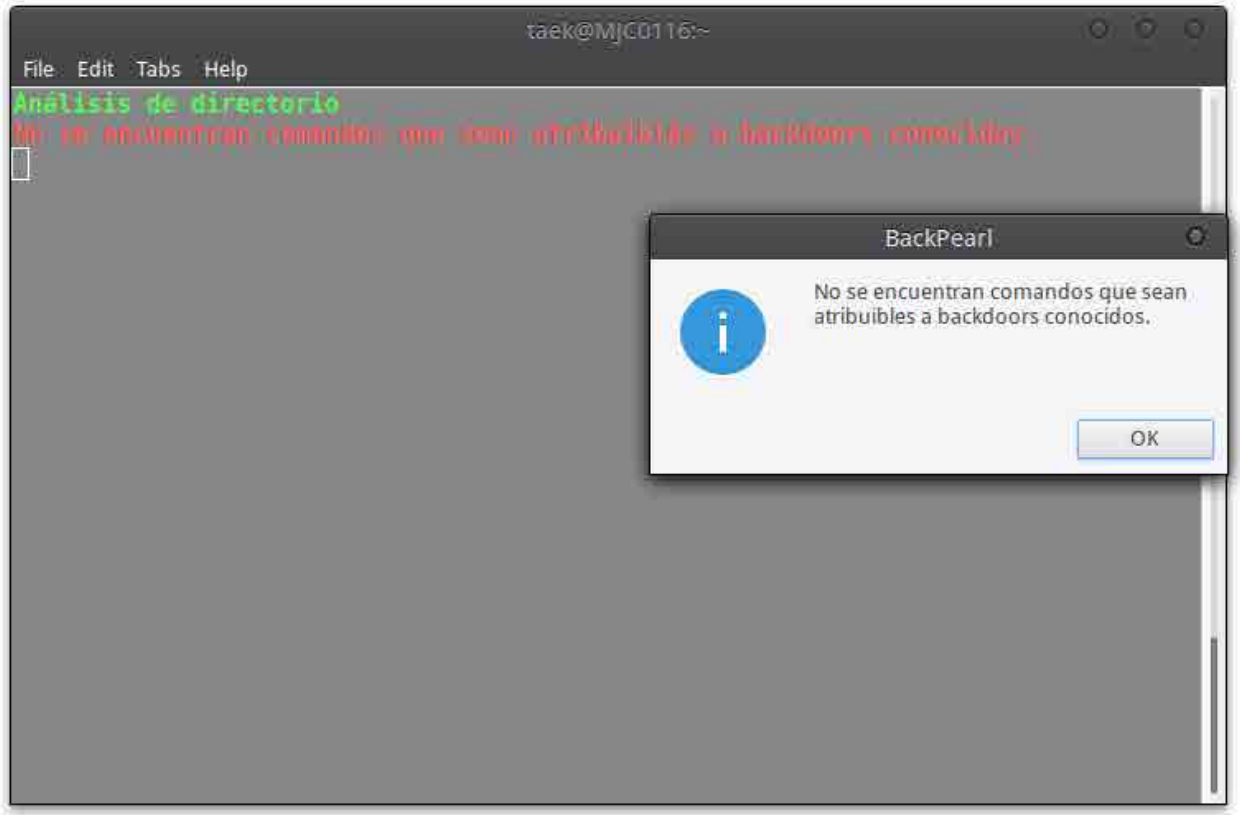

Ilustración 48. Resultados análisis directorio

Análisis de puertos

En el análisis de puertos no es necesario que el usuario ingrese datos. El programa sabe dónde debe realizar el análisis y el mismo comenzará apenas dicho proceso sea invocado. En el caso que se muestra a continuación no hay puertos abiertos que estén relacionados a backdoors según la base de conocimientos.

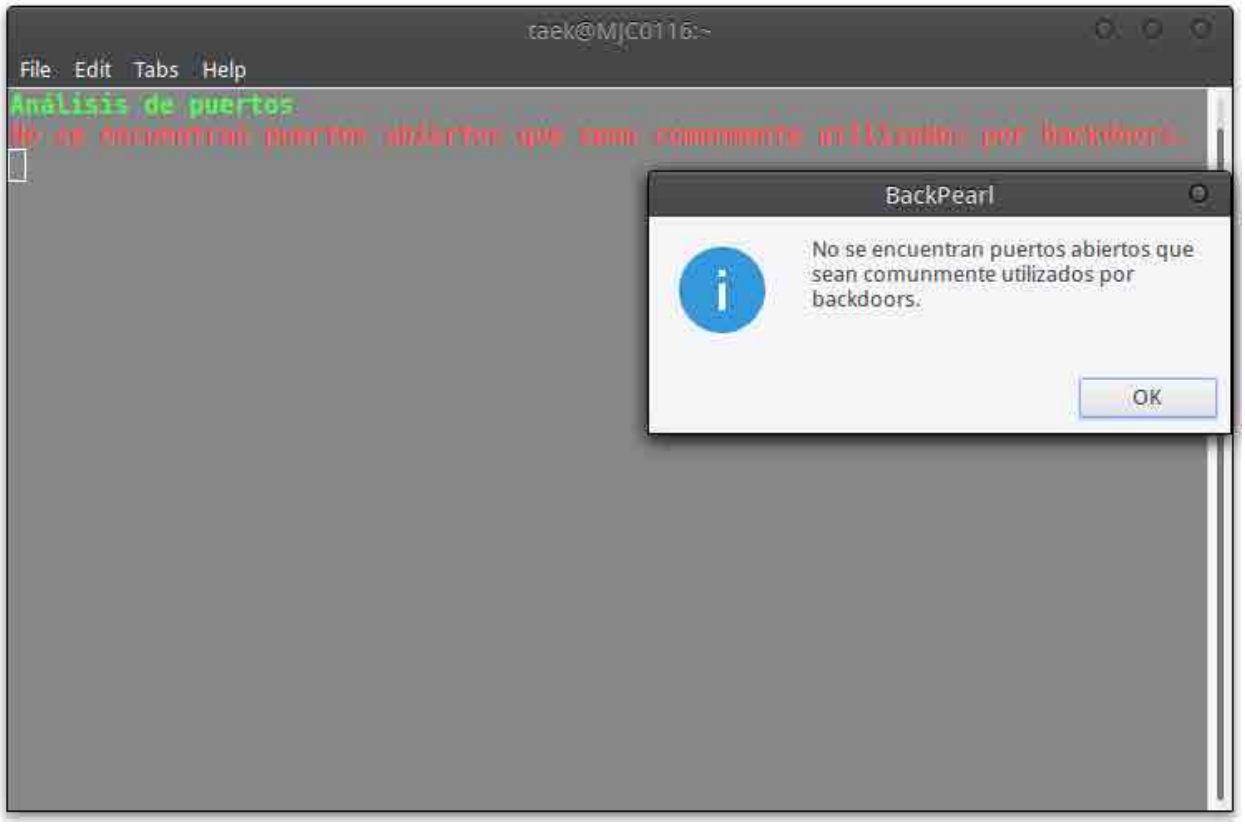

Ilustración 49. Resultados análisis puertos

Análisis de binarios

En el siguiente análisis de binarios se muestra que se han detectado dos posibles backdoors en los archivos mencionados en la primera imagen, y, en la segunda, se muestra más información sobre los comandos descubiertos que generan las sospechas de que en dichos archivos se encuentren backdoors. Respecto al primer archivo, *hedgewars,* se ha modificado intencionalmente el código del binario para que aparezca como una amenaza detectada. En el segundo caso, el binario *nmap* es un falso positivo tal como lo indica la segunda imagen.

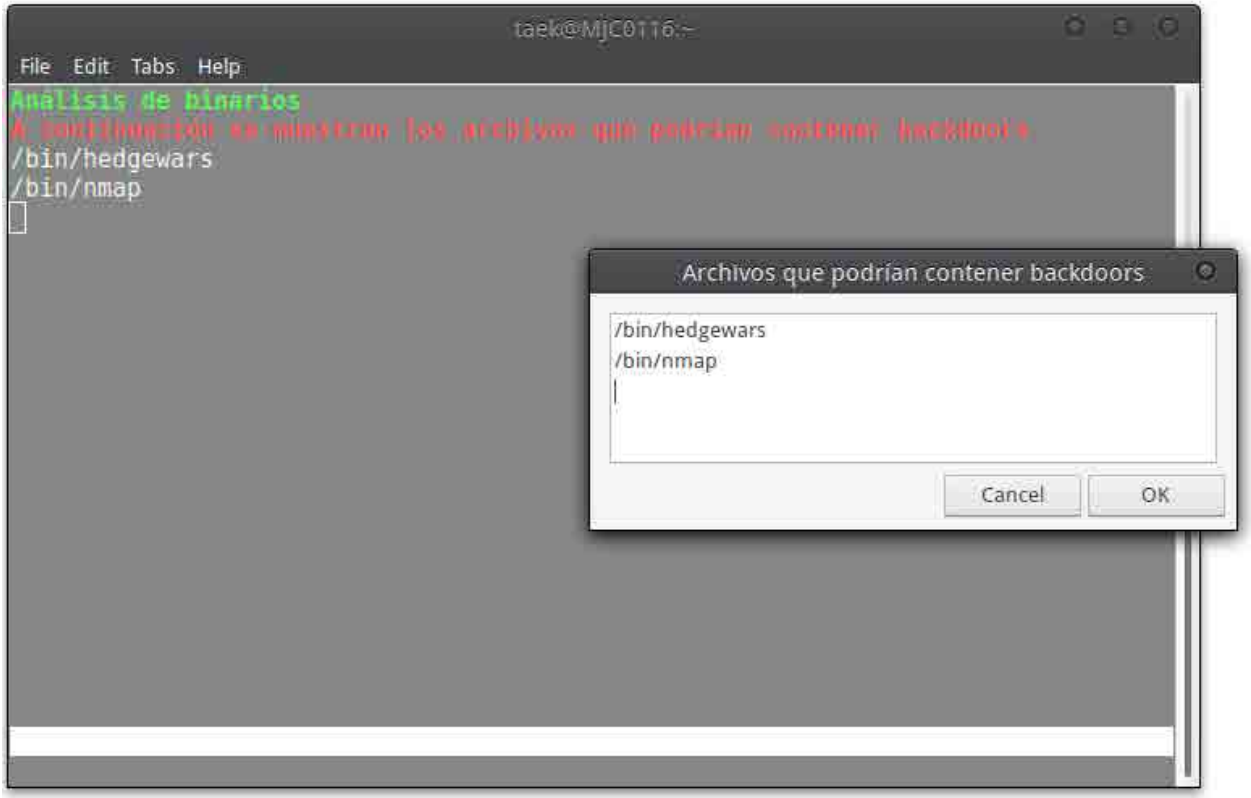

Ilustración 50. Resultados análisis binarios

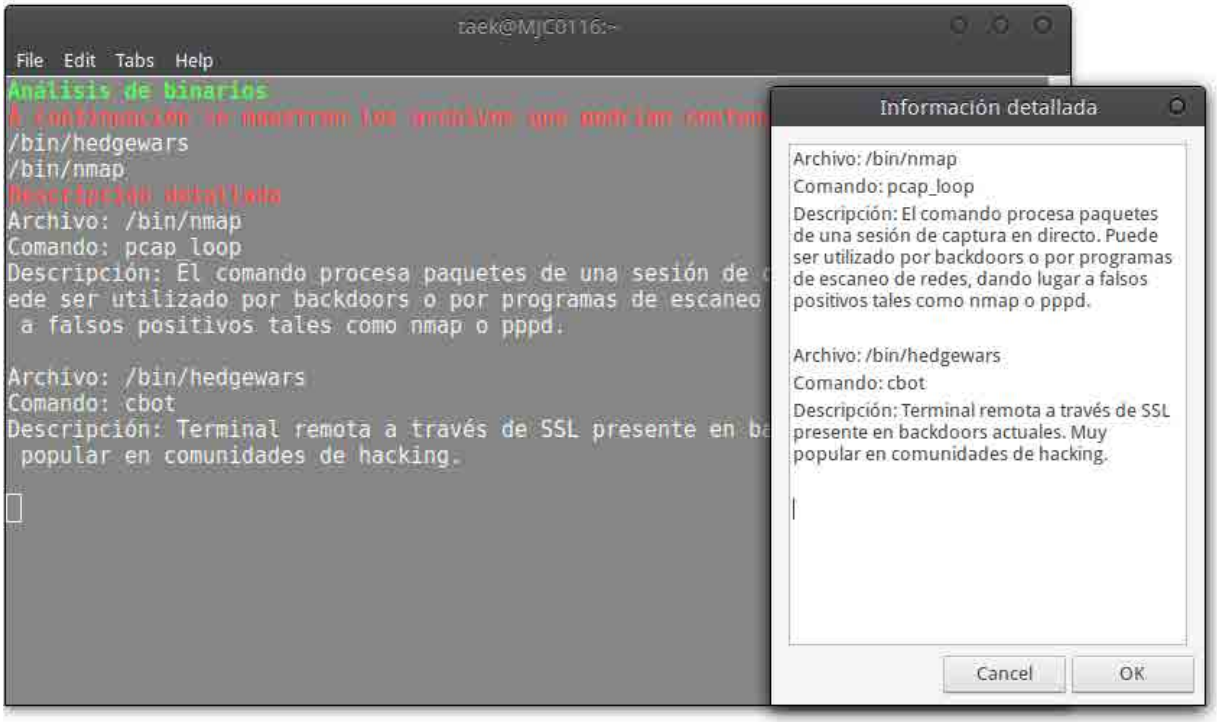

Ilustración 51. Información detallada binarios

Análisis de .tar.gz

A continuación, en el análisis de empaquetados, se detecta un backdoor en un archivo comprimido como .tar.gz. Se le proporciona al usuario la ubicación del mismo, el nombre de la sentencia que genera la sospecha y una descripción de la misma.

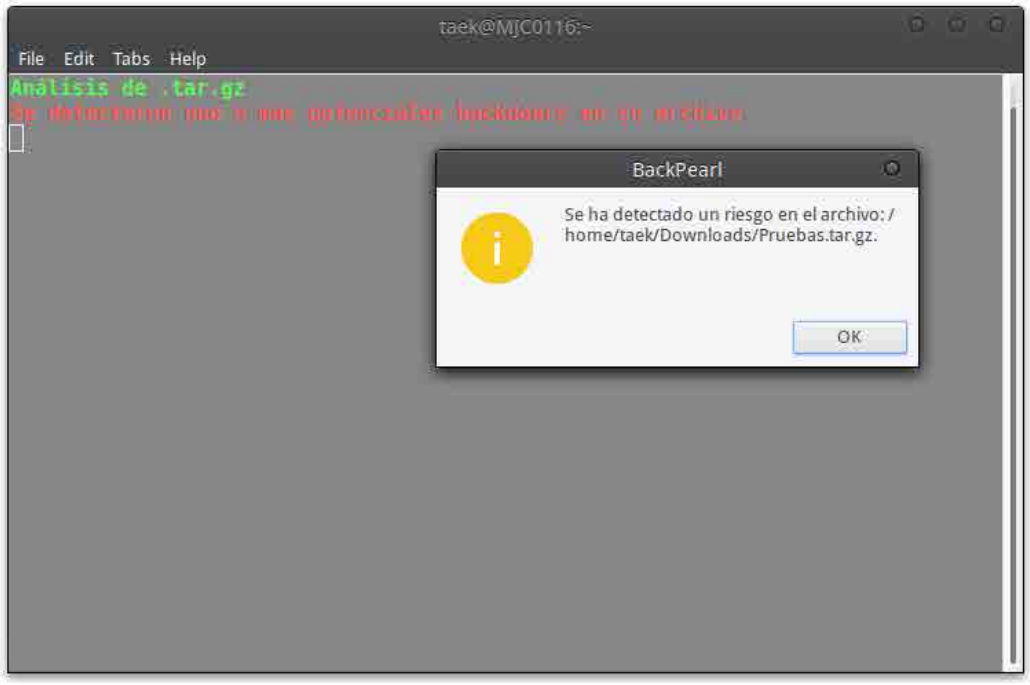

Ilustración 52. Resultados análisis .tar.gz

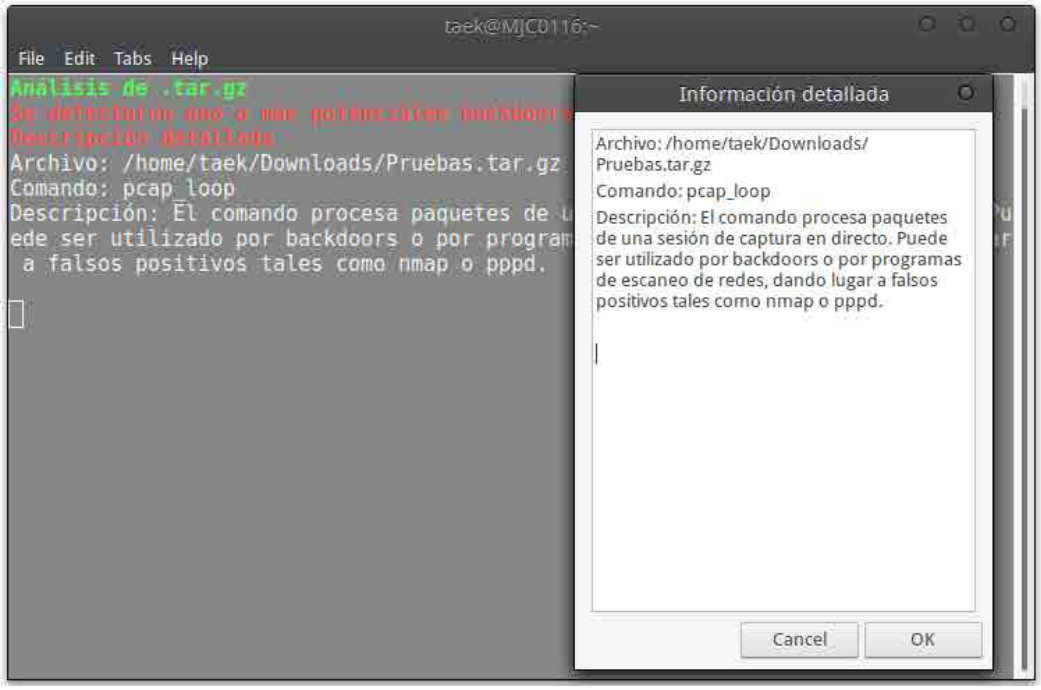

Ilustración 53. Información detallada .tar.gz

Trucos y Consejos

BackPearl ofrece una sección denominada *Trucos y Consejos* tal como lo hacen muchos programas de la misma naturaleza (antivirus, administradores del sistema, limpiadores, etc.). Estos consejos, otorgados de forma aleatoria en cada llamada a dicha opción del menú, le permiten al usuario aprender sobre su sistema basado en Pacman y sobre backdoors. Principalmente se trata de consejos sobre mantenimiento preventivo y optimización del sistema operativo.

Ilustración 54. Trucos y consejos

Ayuda y Configuraciones

La sección de *Ayuda y Configuraciones* cuenta con su propio menú.

Ilustración 55. Menú Ayuda y configuraciones

La primera opción, *Sobre BackPearl,* le brindará una definición sobre el programa y su objetivo.

Ilustración 56. Sobre BackPearl

La siguiente opción, *Ayuda*, equivale a un manual de usuario en donde se encuentran las definiciones de los scripts, su utilización y las configuraciones del programa.

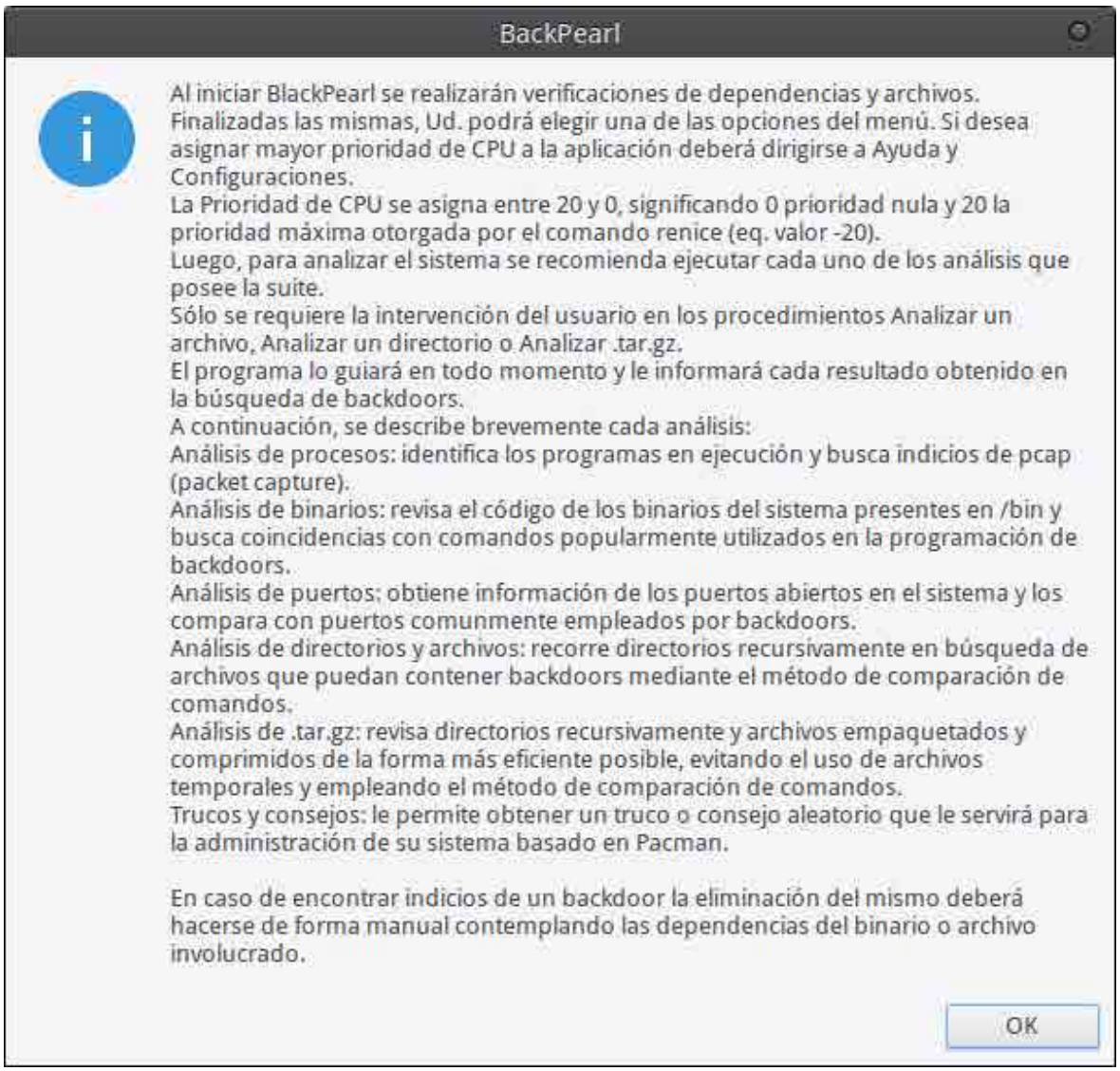

Ilustración 57. Ayuda

Otro ítem del menú es *Licencia*. Esta opción le mostrará una pantalla donde se puede leer la licencia que posee BackPearl, la cual es GPL v3.

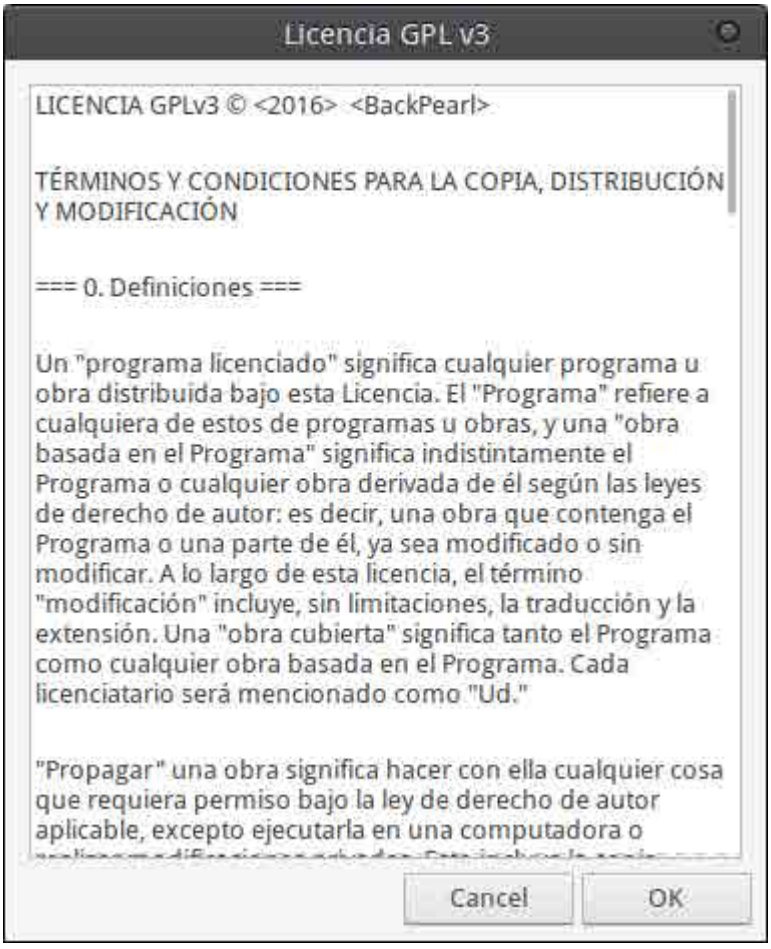

Ilustración 58. Licencia

Por último, se puede configurar la *Prioridad de CPU*, lo cual permite asignar un valor al proceso BackPearl que le permitirá contar con una mayor cuota de CPU. Este valor, medido en números negativos por la llamada al sistema *renice*, puede ser editado como se muestra en las siguientes imágenes.

Se le informa al usuario que al ser un script portátil los cambios no permanecerán al cerrarse el programa, por lo que debe realizarlos antes de iniciar los análisis. Además, el cambio de prioridad de CPU requiere permisos de súper usuario, por lo que se requiere la contraseña del mismo.

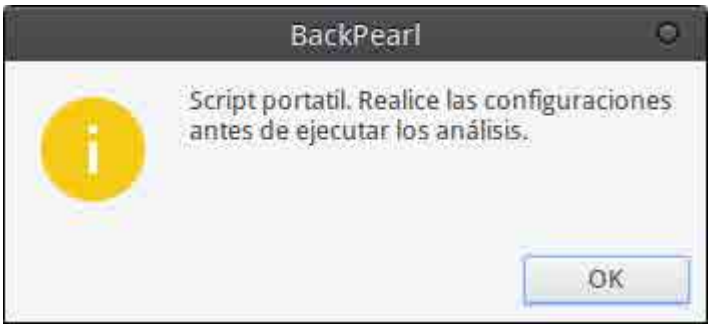

Ilustración 59. Mensaje de alerta

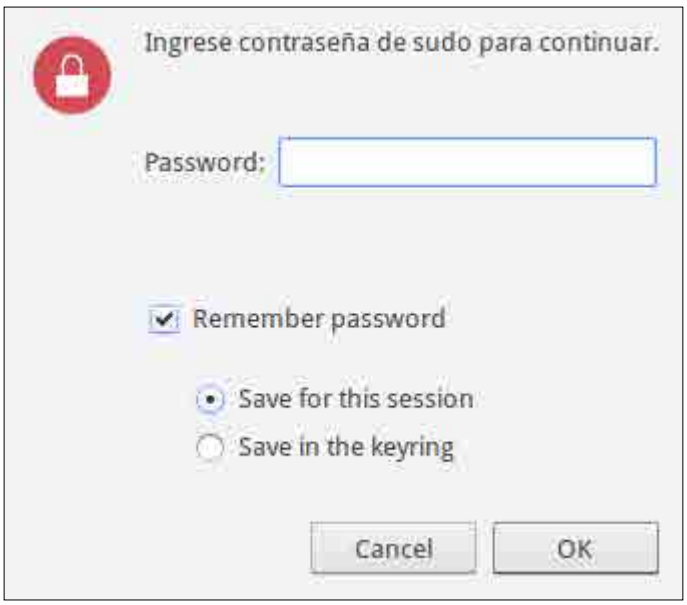

Ilustración 60. Petición de contraseña

Ilustración 61. Selección de prioridad

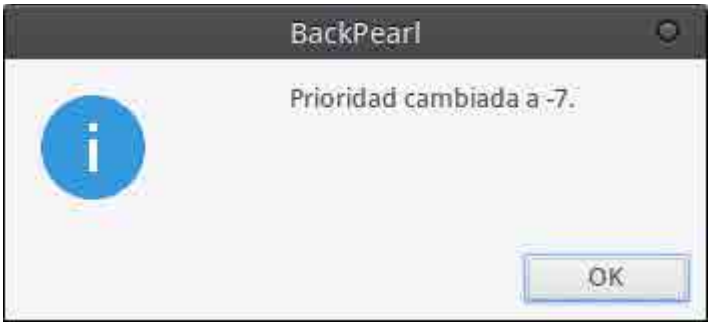

Ilustración 62. Mensaje de cambio de prioridad

Para *renice* el valor -7 significa que la prioridad que tiene un proceso será 7 puntos (sobre un máximo de 20) mayor que el resto de los procesos.

Tratamiento de errores

BackPearl le informará al usuario cuando ocurran errores tales como falta de dependencias, inexistencia de los archivos que se requieren como fuente de datos y

advertencias relacionadas a la distribución GNU/Linux en la cual se ejecuta el programa. A continuación se observan algunos de estos mensajes.

Ilustración 63. Errores por falta de dependencias

Instalación

BackPearl es un programa portable pero requiere cierta configuración inicial. En primer lugar es recomendable comprobar que la variable *directtrabajo*, que hace referencia al directorio de trabajo, apunte a la ruta correcta, es decir, aquella donde se encuentren los archivos .dat necesarios para realizar los análisis. En dicha ruta se crearán y eliminarán, en el momento oportuno, archivos temporales con el sufijo .tmp que son requeridos para el correcto funcionamiento de la suite. El directorio de trabajo ya viene asignado por default en la versión de prueba adjuntada. Por otro lado, es importante que se hayan instalado las dependencias. De esta forma se denomina a un conjunto de programas y librerías que son requeridos para que el programa principal funcione apropiadamente. Las dependencias de BackPearl son: *nmap, zenity, bash y pacman.* El programa comprueba la instalación de estos paquetes y en caso de no encontrarlos le informará al usuario. Además, antes de poder ejecutar los scripts de análisis, se comprobará de forma automática que los archivos .dat ya mencionados se encuentren en el directorio de trabajo asignado. Si no se encuentran entonces se le brindará al usuario un

mensaje de alerta. De la misma forma se verificará si el programa se está ejecutando en las distribuciones GNU/Linux en las que ha sido testeado. Por último, es posible cambiar el nombre del archivo ejecutable pero se recomienda no hacerlo para evitar errores relacionados con variables del entorno. Si por algún motivo el usuario debe realizar modificaciones en el nombre del script entonces deberá editar el código fuente en las llamadas a *pkill* y modificar la variable *pidbd.*

En resumen, si obtiene una copia de BackPearl, la única configuración a realizar es establecer la variable *directtrabajo* antes de ejecutar el programa, indicando la ruta a la carpeta donde se descomprimió el archivo. Asegúrese de tener las dependencias establecidas para evitar los mensajes de alerta del programa.

Restricción

En esta versión de prototipo evite utilizar nombres de archivos y carpetas que contengan espacios ya que algunas llamadas al sistema no reconocen de forma eficiente el nombre completo del archivo o directorio al que se referencia.

Dependencias

BlackPearl comprueba, en su inicio y como se ha mostrado en la sección de *Interfaces de usuario*, que existan las dependencias necesarias para un correcto funcionamiento. Las mismas son:

- Pacman: se trata de un gestor de paquetes que es señalado como dependencia por dos motivos. En primer lugar, por restricciones del alcance del prototipo. Se desea brindar un sistema compatible con la familia de Archlinux y cada distribución que la compone utiliza dicho gestor de paquetes. En segundo lugar, todas las recomendaciones de administración, solución a falta de dependencias y el análisis de empaquetados se realizan teniendo como punto de partida la existencia de Pacman.
- Nmap: es un programa para la captura de paquetes, auditoría de redes y detección de puertos. Es utilizado en el *Análisis de puertos* para realizar una consulta interna del sistema con el fin de determinar los puertos abiertos. Esta es la forma más efectiva y simple de obtener información sobre los puertos considerando todas las alternativas que ofrece GNU/Linux.
- Zenity: es una librería compatible con Shell Script para desarrollar las interfaces gráficas. Permite crear ventanas, cuadros de diálogo, botones, listas y otros componentes visuales. A pesar de ser limitado, ofrece la posibilidad de crear

rápidamente las principales ventanas de un programa. Se requiere como dependencia para que la interfaz gráfica sea visible cuando se ejecuta BlackPearl.

- Bash: es un software intérprete de comandos. BackPearl ha sido diseñado para ser interpretado por Bash siguiendo sus nomenclaturas, normas y restricciones. Se requiere Bash como dependencia porque utilizar otro intérprete puede generar errores de sintaxis.
- Archivos de *comandosbackdoors* y *puertos*. Algunos análisis que lleva a cabo BackPearl requieren realizar comparaciones entre sentencias y puertos que utilizan los backdoors habitualmente. Sin la presencia de estos archivos estos análisis carecen de sentido, ya que no poseen los datos de entrada requeridos.
- Por último, a pesar de no ser una dependencia, se realiza la verificación del sistema operativo o distribución en la cual se está ejecutando BackPearl. Esta comprobación tiene como objetivo detectar si el programa se está ejecutando en distribuciones donde ha sido testeado su correcto funcionamiento, las cuales son Archlinux, Manjaro y Antergos. En caso de que se trate de otra distribución, se alertará al usuario de usar el programa con precaución.

Pruebas y verificación experimental

Pruebas de scripts

A continuación se presentan las pruebas de detección realizadas. Para llevar a cabo las mismas se ha infectado el laboratorio de pruebas con backdoors reales y también con desarrollos personales.

Análisis de archivo

Para la primera prueba se utiliza un backdoor disponible en el foro español de Ubuntu.[33] El mismo utiliza sockets del kernel de Linux por lo tanto tiene la capacidad de infectar a la familia de Archlinux. El código del backdoor se ha descargado y ejecutado correctamente en Manjaro Cinnamon. Se crea un archivo sin extensión para sumar complejidad a una búsqueda manual. A continuación, se ejecuta el módulo Análisis de archivo de BackPearl, el cual detecta efectivamente la amenaza y reporta los siguientes resultados:

Se ha detectado un riesgo en el archivo: /home/tesis/Pruebas/archivoinfectado.

Archivo: /home/tesis/Pruebas/archivoinfectado

Comando: sock_recvmsg

Descripción: Función del kernel para recibir mensajes a través de un socket con protocolos TCP y UDP. Puede generar falsos positivos en software de gestión de redes.

El usuario, a continuación, podrá tomar la decisión de poner el archivo en cuarentena o eliminarlo de forma manual. Al mismo tiempo, el archivo se subió a la suite de antivirus online VirusTotal, propiedad de Google Inc.[34]

El análisis produce los siguientes resultados:

SHA256: a3ccd9b91e85a8dddbb2a70f60ed53cc7fbbe12314d28662a1d7d19cf9d35d83

Nombre archivoinfectado

Detecciones 0 / 48

Esto quiere decir que ningún antivirus pudo detectar una amenaza. El listado de los antivirus, todos ellos actualizados, que evaluaron el backdoor:

Ad-Aware, AegisLab, AhnLab-V3, Alibaba, ALYac, Antiy-AVL, Arcabit, Avast, AVG, AVware, Baidu, BitDefender, Bkav, CAT-QuickHeal, ClamAV, CMC, Comodo, Cyren, DrWeb,

[33] Proyecto Backdoor en Ubuntu. http://www.ubuntu-es.org/node/137348#.V46HNIaIi00. Vigente 19/07/2016.
[34] Página oficial de VirusTotal. www.virustotal.com. Vigente 19/07/2016.

Emsisoft, eScan, ESET-NOD32, F-Prot, F-Secure, Fortinet, GData, Ikarus, Jiangmin, K7AntiVirus, K7GW, Kaspersky, Kingsoft, Malwarebytes, McAfee, McAfee-GW-Edition, Microsoft, NANO-Antivirus, nProtect, Panda, Qihoo-360, Sophos, SUPERAntiSpyware, Symantec, Tencent, TheHacker, TrendMicro, TrendMicro-HouseCall, VBA32, VIPRE, ViRobot, Yandex, Zillya, Zoner.

El análisis se encuentra de forma online en: https://goo.gl/COrYyu o https://www.virustotal.com/es-ar/file/a3ccd9b9-1e85a8dddbb2a70f60ed53cc7fb-be12314d-28662a1-d7d19cf9d-35d83/analysis/1468958808/ Vigente al 19/07/2016.

Análisis de directorio

Con el paquete *tree*, disponible en los repositorios de Archlinux, se puede visualizar un entorno de pruebas con una jerarquía de directorios que contienen cuatro archivos distribuidos aleatoriamente que son inofensivos para el sistema y tres archivos que contienen backdoors. El árbol de directorios es la siguiente:

```
/home/tesis/Pruebas/
── ./Pruebas/Carpeta1
│   ── ./Pruebas/Carpeta1/.Carpeta1.1
│   │   ── ./Pruebas/Carpeta1/.Carpeta1.1/archivocalc.xlsx
│   │   └── ./Pruebas/Carpeta1/.Carpeta1.1/backdoorsocketskernel
│   └── ./Pruebas/Carpeta1/documento.pdf
── ./Pruebas/.Carpeta2
│   ── ./Pruebas/.Carpeta2/archivoimagen.jpg
│   └── ./Pruebas/.Carpeta2/Carpeta2.1
│       └── ./Pruebas/.Carpeta2/Carpeta2.1/backdoorshellinversaphp
└── ./Pruebas/.Carpeta3
    ── ./Pruebas/.Carpeta3/archivoinofensivo.c
    └── ./Pruebas/.Carpeta3/backdoorcasus.c
```

Las carpetas denominadas “Carpeta1.1”, “Carpeta2” y “Carpeta3” están ocultas, es decir que no son visibles para el usuario si no activa la opción correspondiente para hacerlas visibles. Esto se logra, de forma genérica, con la combinación ctrl+h en cualquier gestor de archivos gráfico. En GNU/Linux, una carpeta adquiere el estado de *oculta* cuando existe un punto “.” al comienzo del nombre.

Se procede a analizar el directorio con BackPearl y se obtiene el siguiente resultado:

A continuación se muestran los archivos que podrían contener backdoors.

/home/tesis/Pruebas/Carpeta1/.Carpeta1.1/backdoorsocketskernel

/home/tesis/Pruebas/.Carpeta2/Carpeta2.1/backdoorshellinversaphp

/home/tesis/Pruebas/.Carpeta3/backdoorcasus.c

Descripción detallada

Archivo: /home/tesis/Pruebas/.Carpeta3/backdoorcasus.c

Comando: casus15

Descripción: Terminal remota a través de SSL presente en backdoors actuales.

Archivo: /home/tesis/Pruebas/.Carpeta2/Carpeta2.1/backdoorshellinversaphp

Comando: n3fa5t1ca

Descripción: Terminal remota a través de SSL presente en backdoors actuales. Muy popular en comunidades hispanas de hacking.

Archivo: /home/tesis/Pruebas/Carpeta1/.Carpeta1.1/backdoorsocketskernel

Comando: sock_recvmsg

Descripción: Función del kernel para recibir mensajes a través de un socket con protocolos TCP y UDP. Puede generar falsos positivos en software de gestión de redes.

Como se puede apreciar, se utilizan tres backdoors que son delatados por los siguientes comandos: *sock_recvmsg, n3fa5t1ca* y *casus15.*

Análisis de puertos

Para el análisis de puertos se decide abrir, a través del firewall *ufw*, el puerto 6668. La salida del script es la siguiente:

Alerta. Puerto: 6667 Comprometido. Descripción: Posible suplantador IRC bot. Protocolo: TCP

Si profundizamos la información del puerto encontramos que es utilizado por los siguientes backdoors:

Dark Connection Inside, Dark FTP, Host Control, NetBus worm, ScheduleAgent, SubSeven, Trinity, WinSatan, Vampire, Backdoor.IRC.Flood, Backdoor.Hacarmy.E, Backdoor.Sdbot.AC, Backdoor.Alnica Backdoor.Maxload.[35]

Se le recomienda al usuario cerrar el puerto estableciendo una regla DENY ALL BOTH DIRECTIONS port 6667 en *gufw* o en el firewall que se encuentre en el sistema.

[35] Información del Puerto 6667. http://www.speedguide.net/port.php?port=6667. Vigente 21/07/2016.

Análisis de binarios

Al ejecutar este análisis observaremos dos resultados. El primero, la detección de *nmap*, es un falso positivo tal como se explica en la *Descripción detallada. Nmap*, programa para la captura de paquetes, auditoría de redes y detección de puertos, utiliza *pcap_loop* para realizar un bucle automático de recepción de paquetes. En el segundo caso, se detecta el comando *cbot* en el binario *hedgewars*, un popular juego de GNU/Linux. En este caso, el paquete ha sido modificado para insertarle un backdoor, ya que la versión que se encuentra en Github está libre de malware. *Cbot* es una terminal inversa, un tipo de backdoor, que tiene gran popularidad en foros, comunidades y blogs referidos al hacking.

Los resultados arrojados por BackPearl son los siguientes:

A continuación se muestran los archivos que podrían contener backdoors.

/bin/nmap

/bin/hedgewars

Descripción detallada

Archivo: /bin/nmap

Comando: pcap_loop

Descripción: El comando procesa paquetes de una sesión de captura en directo. Puede ser utilizado por backdoors o por programas de escaneo de redes, dando lugar a falsos positivos tales como nmap o pppd.

Archivo: /bin/hedgewars

Comando: cbot

Descripción: Terminal remota a través de SSL presente en backdoors actuales. Muy popular en comunidades de hacking.

Análisis de .tar.gz

Para probar un caso positivo del análisis de .tar.gz se ha creado un archivo denominado Carpeta.tar.gz. En su interior existen dos carpetas, una visible y la otra oculta, y tres archivos, dos de ellos están limpios y el tercero es el backdoor que se ha estudiado en profundidad en la prueba de análisis de archivo.

BackPearl muestra el siguiente resultado:

Se ha detectado un riesgo en el archivo: /home/tesis/Pruebas/Carpeta.tar.gz.

Comando: sock_recvmsg

Descripción: Función del kernel para recibir mensajes a través de un socket con protocolos TCP y UDP. Puede generar falsos positivos en software de gestión de redes.

Pruebas de performance

Para tener un parámetro de la performance de cada script se recurre a la velocidad de ejecución medida a través de la herramienta *time* de la terminal. Concretamente, se debe ejecutar *time BackPearl.sh* para obtener el tiempo que requiere el programa para ser ejecutado.

Las pruebas se realizan con prioridad 0 de CPU y no se considera el tiempo de la primer ejecución de BackPearl por no encontrarse en la memoria caché. Se han corrido pruebas sin mensajes mostrados a través de la terminal (verbose) con el fin de reducir el tiempo de ejecución considerando las buenas prácticas en la programación en GNU/Linux. El script con mayor cantidad de verbose es el de análisis de procesos. Quitar los mensajes no genera una variación significativa en el tiempo consumido como puede apreciarse a continuación.

Análisis de procesos. Script con mayor verbose (mensajes mostrados por la terminal)	
Tiempo en segundos con verbose. Se promedian 5 pruebas.	Tiempo en segundos sin verbose. Se promedian 5 pruebas.
real 33.274s user 3.887s sys 1.577s	real 32.842s user 3.693s sys 1.757s

Tabla 9. Performance con verbose

El tiempo total de CPU, *sys*, es la combinación de la cantidad de tiempo que la CPU tarda en realizar alguna acción para un programa y la cantidad de tiempo que la CPU tarda en realizar llamadas al sistema para el kernel en nombre del programa. Cuando un programa recorre, por ejemplo, una matriz, se acumula tiempo de CPU de usuario, *user*. Por el contrario, cuando un programa ejecuta una llamada al sistema, por ejemplo un exec o fork, se está acumulando tiempo de CPU del sistema.

El término *tiempo real* en este contexto se refiere al tiempo transcurrido, como si fuese medido por un cronómetro. El tiempo de CPU total (tiempo de usuario + tiempo de sistema) puede ser mayor o menor que ese valor, debido a que un programa puede pasar algún tiempo de espera y no ejecutar nada (ya sea en modo usuario o modo sistema) el tiempo real puede ser mayor que el tiempo total de CPU.[36]

[36] Clasificaciones del tiempo en *time*. https://es.wikipedia.org/wiki/Time_(Unix). Vigente 21/07/2016.

Para el resto de las pruebas los resultados son:

Script	Tiempo en segundos
Análisis de binarios (3486 elementos)	real 56.866s user 37.010s sys 2.093s
Análisis de puertos	real 10.220s user 0.527s sys 0.100s
Análisis de directorio (mismo directorio que en prueba de detección)	real 14.950s user 1.137s sys 0.183s
Análisis de archivo (archivo *backdoorsocketkernel*)	real 10.465s user 1.330s sys 0.137s
Análisis de .tar.gz (archivo *Carpeta.tar.gz*)	real 14.148s user 1.533s sys 0.183s

Tabla 10. Performance de los análisis

Futuras líneas de investigación

El proyecto podría continuar con la investigación y el trabajo abocado a:

- Análisis de archivos multimedia: se pretende detectar backdoors en imágenes, videos, música y flash.
- Generación de interfaces de usuario más atractivas, enfocadas en ser *friendly-user*.
- Profundizar la investigación y la documentación de cuáles son los backdoors que poseen la mayor popularidad actualmente en foros y comunidades de hacking.
- Desarrollar un script de barrido automático que sea capaz de tomar sus propias decisiones sin la participación activa del usuario, análogo a un antivirus de comportamiento activo.
- Diseñar un plan de negocios en conjunto con Thawte Consulting y Symantec Corporation. Ambas son empresas internacionales que generan certificaciones y software de seguridad.
- Extender el alcance del sistema para cubrir todas las distribuciones GNU/Linux.

Dificultades no previstas

Durante el desarrollo del proyecto y del software se presentaron dos dificultades que no fueron correctamente previstas al inicio.

En primer lugar, se estimaba que existía mayor información sobre backdoors, el funcionamiento, la codificación, y principalmente, sobre la forma de detectarlos y erradicarlos del sistema. Para afrontar la situación se optó por buscar contenido en otros idiomas: inglés y, en menor medida, portugués. En inglés hay infinidad de referencias a backdoors, pero la mayoría de los enlaces llevan a noticias que informan de la aparición de malware y no aportan las características técnicas que requiere el presente proyecto. Además, en foros, comunidades y blogs, al hablar de backdoors, se aborda el tema desde un perfil ofensivo. Esto es, se enseña a cómo infectar una computadora, generalmente con *Windows*, utilizando software ampliamente difundido tal como *Meterpreter*. Sobre medidas defensivas y preventivas se encuentra poco contenido por lo cual se ha requerido una mayor cantidad de tiempo en la etapa de investigación.

En segundo lugar, la codificación ha sido más difícil de lo esperado debido a las reglas que existen en el lenguaje de desarrollo especificado para la elaboración del prototipo. Shell script es un lenguaje muy estructurado y estricto. Por ejemplo, un espacio en blanco puede generar un error tal como lo genera la falta de un punto y coma en lenguajes como C o Java.

El manejo de pipes o hilos de ejecución presenta complicaciones porque no existen las variables globales entre dos sesiones de la terminal, lo que conlleva al uso de archivos para poder almacenar datos que serán consultados por un subproceso. Por último, es necesario conocer y saber utilizar los parámetros de las llamadas al sistema y comandos que se utilizan. Estos presentan grandes diferencias con sus equivalentes en otros lenguajes de programación, por lo que saber programar en esos lenguajes no nos evita que se deban aprender las llamadas al sistema de Shell script.

Conclusiones

Encarar un proyecto prácticamente obliga a su realizador a ordenarse, planificar y seguir una metodología para evitar pérdidas de tiempo y dispersiones. Es fácil perder el rumbo, no tener objetivos claramente definidos y desviar la atención hacia detalles en vez de enfocarse en lo importante. Por eso es altamente recomendable adoptar una metodología que permita una buena planificación y gestión del tiempo, el principal recurso en un proyecto de estas características.

En cuanto al tema, la detección de backdoors, puedo decir que ha sido muy gratificante poder dedicar muchas horas al estudio de este tipo de malware. Cualquier esfuerzo por reforzar la seguridad en sistemas GNU/Linux es altamente apreciado por la comunidad del software libre y considero que, en una medida muy pequeña, pude contribuir a reforzar un sistema que se caracteriza por su solidez y confiabilidad.

Además, realizar un proyecto de este tipo es una buena oportunidad para aplicar los conocimientos adquiridos en múltiples cátedras debido a la diversidad de aspectos que se deben cubrir; sólo por enumerar algunos: sistemas operativos, seguridad, gestión de proyectos, dirección, programación, diseño, economía e ingeniería del software.

Bibliografía

Las principales fuentes de consulta son:

- Andew Tanenbaum, 2009. *Sistemas operativos modernos*. 3ª edición. Editorial Pearson. ISBN: 978-607-442-046-3.
- Anónimo, 2002. *Linux Máxima Seguridad.* Edición Especial. Editorial Prentice Hall. ISBN: 848-322-244-2.
- Carlos Prado, 2013. *Detectando Backdoors silenciosos*.
- Carlos Tori, 2008. *Hacking ético*. ISBN: 978-987-05-4364-0.
- Justin Clarke, 2009. *SQL Injection Attacks and Defense*. Syngress Publishing, Inc. EEUU. ISBN: 978-1-59749-424-3.
- Karina Astudillo, 2013. *Hacking ético 101*. ISBN: 978-149-228-878-7.
- Noah Gift, 2009. *Python para administración de sistemas UNIX y LINUX.* Anaya Multimedia. EEUU. ISBN: 978-8441525405.
- Raúl Duque, 2008. *Python para todos.* Versión digital S/ISBN.
- Umberto Eco, *Cómo se hace una tesis,* Edición 2008, Gredisa Editorial. ISBN: 9788474328967.
- Martín Alaimo, 2013, *Proyectos ágiles con #SCRUM*.
- Project Management Institute, 2008. *Guía de los fundamentos para la dirección de proyectos – Guía del PMBOK.* Cuarta edición. ISBN: 9781933890722.

Anexo

Glosario

- AUR: Arch User Repository: repositorio no oficial mantenido por la comunidad de Archlinux. Brinda un conjunto de paquetes en formato .tar.gz.
- Backdoor: un conjunto de sentencias incrustadas en el código de un programa o librería que posibilitan el acceso remoto por parte de una persona a la cual se le otorga la capacidad de tomar control del sistema.
- Distribución (GNU/Linux): recopilación de programas, configuraciones, diseños y adaptaciones que conforman un sistema operativo con el kernel Linux.
- Falso positivo: etiqueta que se le otorga a un archivo que ha sido detectado como malware por un antivirus cuando en realidad no lo es.
- GNU/Linux: combinación del núcleo Linux, encargado de la administración del hardware, y el conjunto de programas, aplicativos y librerías del proyecto de desarrollo de software libre denominado GNU.
- Malware: tipo de software malintencionado o malicioso que tiene por objetivo dañar un sistema, infiltrarse en él, comprometerlo o generar un comportamiento indeseado por el propietario.
- Mirror: es una réplica de datos o programas entre dos o más servidores y realiza la función de ser sistema de backup a la vez que permite mejores velocidades de descarga.
- Repositorio: conjunto de paquetes y librerías disponibles para una distribución GNU/Linux. Equivale a una tienda de la cual se pueden descargar los programas.
- Rootkit: malware de acceso privilegiado y continuo a un sistema que no depende de un programa principal ni es transmitido a través de otro.
- Tar.gz: algoritmos de empaquetado (tar) y compresión (gz), con el propósito de convertir un conjunto de archivos en uno sólo para facilitar su gestión y transferencia, reduciendo el tamaño de los mismos sin pérdidas, utilizando el algoritmo deflate.

Licencia

La licencia se encuentra en el archivo denominado licencia.dat adjunto con el presente trabajo. La misma es una copia exacta de la licencia GPL v3 © con referencias al software BackPearl.

Código del backdoor para prueba de *Análisis de archivo*

El código fuente del backdoor analizado en profundidad en la sección de *Pruebas y verificación experimental* se encuentra en el archivo denominado *CódigoBackdoor.pdf* adjunto con el presente trabajo.

Código de BackPearl

El código fuente de BackPearl se encuentra, junto con la documentación de sentencias, en el archivo denominado *BackPearl.sh.pdf* adjunto en el presente trabajo.